VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ APROXIMACE OBRAZOVÝCH FILTRŮ

EVOLUTIONARY APPROXIMATION OF IMAGE FILTERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ FOUKAL**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Foukal Tomáš, Bc.**
Program: Informační technologie Obor: Bioinformatika a biocomputing
Název: **Evoluční aproximace obrazových filtrů**
Evolutionary Approximation of Image Filters
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou evolučních algoritmů, aproximativního počítání a filtrace obrazů.
2. Navrhněte způsob využití evolučního algoritmu pro aproximaci obrazových filtrů.
3. Navržený způsob evoluční aproximace implementujte a experimentálně ověřte na vybraných filtrech.
4. Porovnejte původní a aproximované filtry dle zvolených kritérií.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

Abstrakt

Tato diplomová práce popisuje problematiku aproximativního počítání, filtrování obrazu v hardwaru a evolučních algoritmů. Představuje řešení problému evoluční aproximace mediánových filtrů, kdy je cílem snížit výpočetní a implementační náročnost filtrace a současně minimalizovat chybu výpočtu. Na základě získaných poznatků a návrhů byly vytvořeny implementace nutných programů. Experimentální vyhodnocení ukazuje, že navržená metoda může pro mediánový filtr poskytovat dobrý kompromis mezi kvalitou filtrování a implementační cenou.

Abstract

This master's thesis introduces the areas of approximate computing, image filtering in hardware and evolutionary algorithms. It proposes a new design solution to the problem of the evolutionary approximation of median filters, where the objective is to reduce computational and implementation requirements and simultaneously minimize the error of filtering. Based on the gained knowledge and proposals, the necessary programs have been implemented. Experimental evaluation shows that the proposed method can provide good tradeoffs between the quality of filtering and the implementation cost for median filters.

Klíčová slova

Umělá inteligence, evoluční návrh, zpracování obrazu, číslicové systémy, aproximativní výpočty, mediánový filtr

Keywords

Artificial intelligence, evolutionary design, image processing, digital systems, approximate computing, median filter

Citace

FOUKAL, Tomáš. *Evoluční aproximace obrazových filtrů.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční aproximace obrazových filtrů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana profesora Lukáše Sekaniny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Foukal
30. července 2019

Poděkování

Děkuji panu profesoru Sekaninovi za odborné vedení a pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Obrazové filtry se uplatňují v mnoha oblastech počítačové techniky – namátkou mohu připomenout průmyslové roboty, bezpilotní vozidla všeho druhu, lékařské přístroje i mobilní telefony. Jejich úspěšný vývoj probíhá už spoustu let. Je tedy evidentní, že jejich efektivní ale i například levná implementace bude podstatná. Tato práce se zaměřuje na rozpracování problematiky využití nepřesně počítajících – ale zato v jiných ohledech prospěšných – komponent pro zlepšení vlastností obrazových filtrů. Nastavuje základní pojmy, vysvětluje potřebnou teorii na pozadí. Připravuje podklady pro problematiku prohledávání stavového prostoru řešení, na kterou vedou některé automatizované metody návrhu obrazových filtrů. Diskutuje využití evolučních algoritmů pro řešení těchto problémů.

Cílem této práce je čtenáře seznámit s problematiku obrazových filtrů, aproximativního počítaní, některých číslicových systémů a evolučních algoritmů. Dále pak vytvořit návrh využití evolučních algoritmů pro optimalizaci obrazového filtru pomocí funkcionální aproximace a navrhnout řešení z tohoto cíle vyvstávajících problémů. Dalším cílem této práce je navrhnout experimenty s vytvořeným modelem a samotný model implementovat. Závěrečným cílem je pak provést detailní experimenty, vyhodnotit je a analyzovat možné pokračování této práce.

Kapitola 2 popisuje a vysvětluje teorii nutnou k pochopení aproximativních výpočtů, filtrování obrazu, potřebných číslicových systémů a evolučních algoritmů. Zavádí pojmy v těchto okruzích využívané a popisuje některé problémy a úskalí. Seznamuje čtenáře se standardními postupy pro řešení funkcionální aproximace, lokální filtrace obrazu a dalších relevantních problémů.

Kapitola 3 se zabývá problematikou filtrování šumu v poškozených obrazových datech. Popisuje inovativní způsob návrhu obrazových filtrů za využití technik funkcionální aproximace a evolučních algoritmů. Tuto problematiku rozebírá na jednotlivé podproblémy a zaobírá se jejich přesným popisem a návrhem potřebných řešení. Dále také představuje technologie, které budou při implementaci programů využity.

Tématem kapitoly 4 je popsání implementovaných programů, které bylo nutné vytvořit ke splnění cílů stanovených v návrhu.

Kapitola 5 představuje konkrétní výsledky, které byly získány po provedení experimentů s implementovanými modely a programy. Výsledky jednotlivých experimentů následně vkládá do společného kontextu a porovnává je.

Kapitola 6 shrnuje obsah této práce a nastiňuje její možné pokračování.

Kapitola 2

Současný stav poznání

Následující podkapitoly se zabývají teorií potřebnou k pochopení problémů řešených v této diplomové práci. Konkrétně se jedná o úvody do aproximativního počítání (kapitola 2.1), lokální filtrace obrazu (kapitola 2.2) a evolučních algoritmů (kapitola 2.3).

2.1 Aproximativní výpočty

V mnoha aplikacích není nutné získávat přesné výsledky výpočtů. Přesný výsledek mnohdy vyžaduje velké množství zdrojů nebo úsilí. Jako příklady z běžné lidské praxe můžeme uvést situaci, kdy se člověk snaží vypočítat nějaký výpočet „z hlavy“ a je ochoten přijmout určitou nepřesnost. Na otázku „Kolik je zhruba 1 001 krát 1 002?“, by odpověď mohla znít „Něco přes jeden milion.“ Samozřejmě správná odpověď zní 1 003 002, chybu 3 002 však zanedbáváme, protože přesnou odpověď nemusíme v tomto případě znát, tazatel ji nutně nevyžaduje. Dalším příkladem – tentokrát již z našeho oboru informačních technologií – může být uložení zvuku. Kvůli převodu zvuku ze spojitého prostoru do diskrétního prostoru nám unikají informace, stále však díky limitům lidského vnímání člověk nemusí být schopen zaznamenat jakýkoli rozdíl. Ostatně s nepřesností se setkáváme dennodenně, lidské vnímání není dokonale přesné.

V posledních letech dochází ke zvyšování zájmu o aproximativní výpočty, zvláště kvůli snaze o snižovaní spotřeby elektrické energie a zvyšování hustoty integrace komponent na čipech. Kromě zmíněného faktu, že člověk není často schopen rozpoznat rozdíl mezi ideálně vypočteným a mírně nepřesným výsledkem (obraz, zvuk), jsou aproximativní výpočty motivovány i jinými aplikacemi. Například v případě, kdy ideálně přesný výsledek je neznámý nebo nedosažitelný v rozumném čase. Zvláště v posledních 7 letech (diplomová práce psána v roce 2019) se aproximativní počítání postupně vyvíjelo a ovlivnilo způsob, jakým jsou počítačové systémy stavěny – od baterií napájených malých zařízení, přes stolní počítače, až po superpočítače [8]. Moderní systémy zvyšují své požadavky na výpočetní zdroje. Nabízí se využití aproximativních výpočtů pro optimalizaci nebo urychlení výpočtů. Nejedná se však o panaceum[1], je třeba tuto techniku využívat tam, kde je to vhodné a žádoucí. Využití při implementaci obrazových filtrů se skutečně samo nabízí. Člověk je těžko schopen rozeznat minimální rozdíly ve vizuálním obsahu a mnohdy to ani nepotřebuje. Přitom, jak naznačují články [8], [12], ze kterých tato práce čerpá, může být kompromis mezi implementační cenou a kvalitou filtrace velmi zajímavý.

[1] všelék, lék na všechno

2.1.1 Vyhodnocení citlivosti na aproximaci

Před samotnou aproximací je nutné nejdříve aproximovaný systém analyzovat a zjistit, které části jsou pro aproximaci vhodné. Je třeba odhalit, jak jednotlivé části systému ovlivňují námi zkoumané parametry jako je přesnost výpočtu, spotřeba zdrojů a další. Tyto části jsou pak označeny buď za citlivé, kdy by jejich úprava mohla vést k selhání systému, nebo za odolné. Odolné části jsou pak vhodné k aproximaci [7]. Výše zmíněné vyhodnocování lze provádět například tak, že jsou do systému vpraveny náhodné změny a jejich důsledky jsou pak statisticky vyhodnoceny. Lze také postupovat deterministicky – zvolit určitou míru aproximace z několika možných úrovní a nahrazovat přesné komponenty při procházení stavového prostoru hrubou silou. Následným krokem je pak vyhodnocení vlivu takto provedené úpravy na samotný systém. Získaná data pak mohou být využita pro přesnější a efektivnější aproximaci onoho systému.

2.1.2 Chybové metriky

Důležitou úlohou je dále měření chyby vnesené do systému kvůli aproximaci. Změřená úroveň chybovosti nám umožňuje porovnávat kvalitu jednotlivých řešení. Nejčastěji měření chyby probíhá tak, že je výstup aproximovaného systému porovnáván s výstupem systému přesného. Tento popis je však velmi obecný, způsob měření chyby je silně závislý na konkrétní aplikaci.

Rozlišujeme *rozsah chyby* e_{wst}, neboli největší možnou chybu:

$$e_{wst} = \max_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|, \tag{2.1}$$

kde $O^{(i)}_{orig}$ je výstup přesného řešení a $O^{(i)}_{approx}$ je výstup aproximovaného řešení. Pokud si tedy stanovíme určitou hodnotu ϵ a budeme dodržovat podmínku $e_{wst} \leq \epsilon$ při návrhu aproximovaného systému, budeme jistě vědět, že nejhorší chyba není vetší než hodnota ϵ.

Dále je dána *maximální relativní chyba* e_{rel}:

$$e_{rel} = \max_{\forall i} \frac{|O^{(i)}_{orig} - O^{(i)}_{approx}|}{O^{(i)}_{orig}}. \tag{2.2}$$

Průměrný rozsah chyby e_{avg}:

$$e_{avg} = \frac{\sum_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|}{n}, \tag{2.3}$$

kde n je počet testovacích vektorů.

Pravděpodobnost chyby e_{prob} je definována jako procentuální zastoupení vektorů, kdy se aproximovaný výstup lišil od původního neaproximovaného výstupu [7]:

$$e_{prob} = \frac{\sum_{\forall i}, O^{(i)}_{orig} \neq O^{(i)}_{approx}}{n}. \tag{2.4}$$

Kromě výše uvedených obecně použitelných chybových metrik se často využívají aplikačně specifické chybové metriky. Například v oblasti zpracování obrazu se jedná o Peak Signal to Noise Ratio (PSNR) [8]:

$$PSNR = 10 \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i,j} (v(i,j) - w(i,j))^2}, \tag{2.5}$$

kde N je výška obrázku v pixelech, M je šířka obrázku v pixelech, $v(i,j)$ je funkce vracející hodnotu pixelu zadaného souřadnicemi i a j z vyfiltrovaného obrázku a $w(i,j)$ je funkce vracející hodnotu pixelu zadaného souřadnicemi i a j z referenčního obrázku.

Další metrikou užívanou v oblasti zpracování obrazu je pak průměrná chyba na pixel obrazu (mean absolute error – MAE):

$$MAE = \frac{1}{MN} \sum_{i,j} |v(i,j) - w(i,j)|. \tag{2.6}$$

2.1.3 Aproximace na úrovni softwaru

Nejobecněji je možno pole využití aproximace v informačních technologiích rozdělit na dvě základní úrovně: na hardwarovou aproximaci a softwarovou aproximaci. V softwaru můžeme snižovat přesnost reprezentace čísel, upravovat kód například jeho zjednodušením, přeskakovat volání funkcí, nebo některé iterace cyklů. Dále nemusíme vyžadovat striktní synchronizaci procesů, můžeme tolerovat chyby v uložení dat... V této práci se však více budu zabývat aproximací na úrovni hardwaru.

2.1.4 Voltage Over-Scaling

Metoda aproximace pomocí *Voltage Over-Scaling* pracuje s nějakým číslicovým obvodem, který v normálních podmínkách (podmínkách, na které byl obvod navržen) funguje přesně. Na jeho vstup je pak přivedeno nižší napětí, než na které byl obvod stavěn. Toto nižší napětí způsobí snížení příkonu (dynamický příkon roste kvadraticky s napájecím napětím), ale mohou vzniknout chyby v časování a tudíž nekorektní výstupní hodnoty [7].

2.1.5 Funkcionální aproximace číslicových obvodů

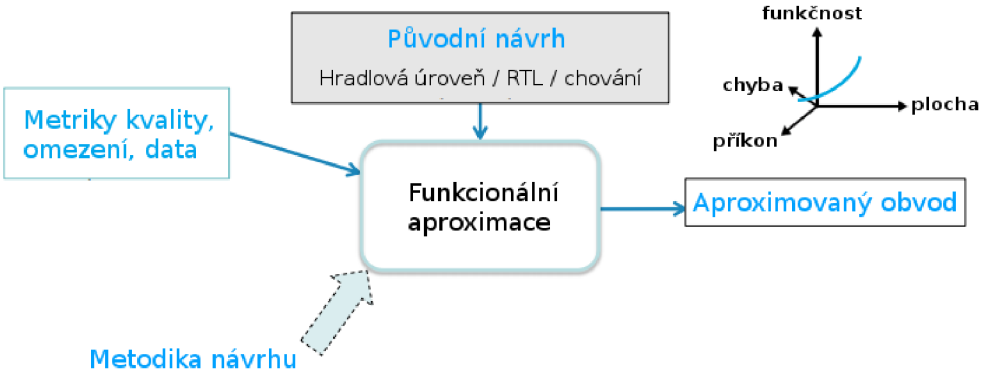
Principem funkcionální aproximace je náhrada přesných komponent v původním obvodu aproximovanými komponentami. Začínáme tedy s obvodem popsaným na úrovni tranzistorů, hradel, jednolivých vyšších komponent, nebo s obvodem popsaným pomocí jeho chování. Cílem je upravit obvod tak, aby se zlepšily některé parametry jako spotřeba energie nebo potřebná plocha při zachování přijatelné chyby [6] (viz obrázek 2.1). Tyto chyby musí být měřitelné – například pomocí v kapitole 2.1.2 uvedených chybových metrik.

Existuje několik metod provádění funkcionální aproximace číslicových obvodů. Nejjednodušší je prosté odstranění komponent. Další možností je náhrada původních komponent aproximovanými komponentami s požadovanými vlastnostmi. Dále je možno přistoupit k resyntéze obvodu, kdy jsou přeskládány samotné komponenty obvodu tak, aby se optimalizoval určitý parametr (například zpoždění obvodu) a chyba obvodu byla přijatelná. Zjištění idealní úpravy tak, aby bylo dosaženo požadovaného výsledku, však nemusí být triviální záležitostí.

Dále rozlišujeme dva rozdílné přístupy, jak mohou být výše zmiňované metody prováděny.

Ruční přístup

Odborník na dané téma zvolí prvky, které budou odstraněny, popřípadě nahrazeny aproximovanými. Dále musí být určeny metriky výsledku, musí být porovnán s původním obvodem



Obrázek 2.1: Schéma vstupů a výstupů funkcionální aproximace. Zdroj: [11]

a musí být vyhodnoceno, zda je výsledek nějakým způsobem přínosný. Je evidentní, že takový proces je náročný – vyžaduje znalost problematiky, čas, určitou dávku štěstí a hlavně onoho odborníka, který tento proces vykoná. Uchylujeme se proto k principiálně jinému přístupu, a to k automatizaci tohoto postupu.

Automatický přístup

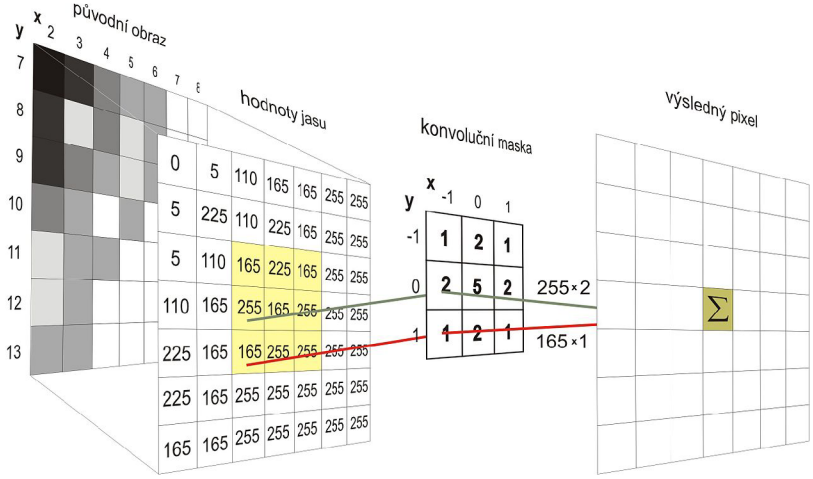
Odborníka nahrazuje sofistikovaný algoritmus. Problém odpojování, nahrazování komponent, resyntézy atd. je formalizován a algoritmicky vyřešen. V mnoha případech řešení problému vede na nutnost prohledávání prostoru možných řešení. Ten je mnohdy příliš velký na prohledávání konvenčními způsoby [6]. Na řadu přicházejí heuristiky a optimalizační metody, jimiž se zabývá kapitola 2.3.

2.2 Lokální filtrace obrazu

Filtrování je způsob úpravy obrazu za účelem jeho zlepšení. Obrázek můžeme pomocí filtrování rozmazávat, ostřit, snižovat v něm přítomný šum nebo zvýrazňovat v něm hrany. Obrazový filtr je struktura provádějící algoritmus filtrování obrazu. *Lokální* obrazový filtr je obrazový filtr, který se nedívá na obraz jako na celek, ale vždy zpracovává pouze jeho určitou část – například okolí určitého daného pixelu [13]. Pro tento přístup může existovat hned několik důvodů. Daná aplikace může vyžadovat rychlé zpracování, nemusí mít dostatek paměti pro uložení kompletně celého obrázu nebo může být limitována energetická náročnost a velikost aplikace.

2.2.1 Obecný popis

Obraz je chápán jako matice hodnot. Ta je u lokálních filtrů zpracovávána po částech. Je třeba stanovit velikost okolí, které bude filtrem bráno v potaz. Toto okolí může být například 3 x 3, 5 x 5 nebo 7 x 7 pixelů [10]. S narůstajícím okolím narůstá samozřejmě i složitost provedeného výpočtu. Zároveň je během výpočtu vytvářen nový obrázek, protože při aplikaci výstupu přímo na původní obrázek by došlo k jeho poškození. Okolí bývá nazýváno posuvné okno (*Moving Window*). Výpočet probíhá posunem posuvného okna po matici pro každý jeden prvek matice [3]. Uvedený princip je ilustrován na obrázku 2.2.



Obrázek 2.2: Ukázka filtrování obrazu pomocí posuvného okna. Jedná se o aplikaci operace konvoluce. V tomto případě je uvedena suma součinů, operace mohou být však obecně různé.[2]

Pro lokální filtrování obrazu byla vyvinuta řada konvenčních algoritmů. Některé z nich v následujících podkapitolách popíši. Také se zaměřím na jednotlivé druhy šumu, které je možné těmito filtry potlačit.

2.2.2 Typy šumu

Při přenosu, snímání a dalších operacích s daty dochází k chybám, které se projevují jako šum. Je naší motivací tento šum potlačovat a dosahovat tím lepší kvality obrazu. Ve zpracování obrazu se zajímáme o různé typy šumu. Některé z jeho druhů jsou následující [13] [10]:

- *šum sůl a pepř*: projevuje se extrémními hodnotami, kdy poškozené pixely nabývají maximální nebo minimální možné hodnoty,
- *náhodný výstřelový šum*: pixely mohou nabývat libovoných hodnot, identifikovat takto poškozený pixel je složitější, než u šumu sůl a pepř,
- *impulsní dávkový šum*: poškozené pixely vytvářejí sledy, nabývají maximální nebo minimální možné hodnoty.

2.2.3 Konvoluční filtr

Konvoluční filtr využívá operaci konvoluce (viz obrázek 2.2). Na matici obrazu přikládáme matici posuvného okna, každý bod násobíme zadaným koeficientem a vytvoříme sumu. Například při matici posuvného okna 5 x 5 pixelů a koeficientech 1/25 provádíme průměrování a obraz rozmazáváme [5]. Vzorec konvoluce pro filtrování obrazu zní:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t), \tag{2.7}$$

[2]Zdroj: Autor: grt – Vlastní dílo, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=969064.

kde x a y jsou souřadnice filtrovaného pixelu, a a b je rozpětí souřadnic v posuvném okně, $w(x, y)$ funkce vracející hodnotu v daném místě posuvného okna, $f(x, y)$ funkce vracející hodnotu v daném místě původního obrazu.

Na hodnotách koeficientů v posuvném okně velmi záleží, podle jejich hodnot a pozic můžeme například provádět detekci hran, rozostřování, nebo zaostřování (viz obrázek 2.3) [13].

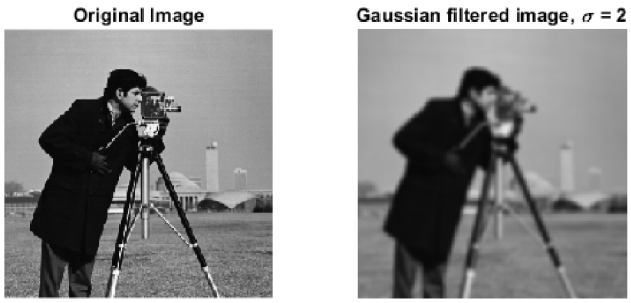
Obrázek 2.3: Výsledek aplikování posuvného okna na obrázek – detekce hran. Zdroj: https://docs.gimp.org/2.8/en/plug-in-convmatrix.html.

2.2.4 Gaussův filtr

Gaussův filtr cíleně rozmazává obrázek (viz obrázek 2.4) a potlačuje tím v něm šum. V 2D prostoru je Gaussův filtr definován pomocí funkce dvou proměnných ve tvaru:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2.8}$$

kde σ je směrodatná odchylka normálního rozdělení prvků uvnitř posuvného okna.



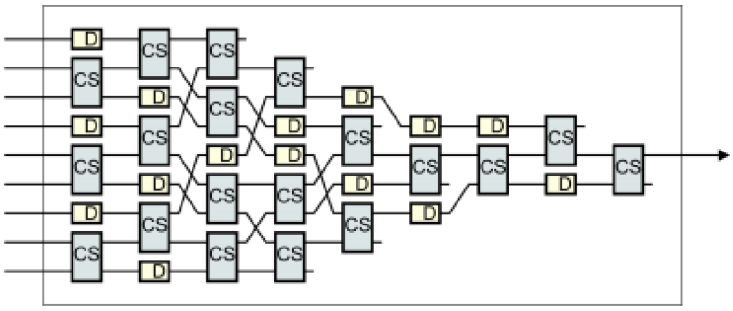
Obrázek 2.4: Výsledek aplikování Gaussova filtru na obrázek. Zdroj: https://www.mathworks.com/help/images/ref/imgaussfilt.html

2.2.5 Mediánový filtr

Výpočet mediánu se řadí mezi nelineární operace. Nelineární metody filtrace obrazu nám mohou pomoci se specifickými úlohami. Mezi ně se řadí potlačení určitých typů šumu při zachování detailů v obraze. Mediánový filtr se standardně používá k potlačení šumu sůl a pepř [13].

Medián se vypočítá seřazením hodnot v posuvném okně a vybráním prostředního prvku jako výstupu. Takto se posuvné okno postupně aplikuje na všechny pixely obrazu. Výsledný

obraz je však menší než vstupní obraz (např. u mediánového filtru 3 x 3 je výsledný obraz menší o jeden pixel na všech stranách – nahoře, dole, vlevo a vpravo). Tento fakt nemusí být na první pohled evidentní, může však být třeba si se s ním vypořádat. Nejčastějším řešením tohoto problému bývá rozšíření vstupního obrazu o požadovaný počet pixelů, kdy jsou hodnoty hraničních pixelů rozkopírovány do nově vzniklých vrstev. Mediánový filtr dobře potlačuje „salt and pepper noise“, tedy šum sůl a pepř. Základní mediánový filtr s posuvným oknem 3 x 3, nebo 5 x 5 je schopen potlačit šum intezity nižší než přibližně 10 – 20 % [10]. V hardwaru se mediánový filtr často implementuje jako řadicí síť (viz obrázek 2.5). Řadicí síť seřadí pixely uvnitř posuvného okna a vrátí hodnotu uprostřed. Nepotřebné komponenty výstupu jsou vynechány. Se zvětšováním posuvného okna nabývá řadící síť na složitosti [12].

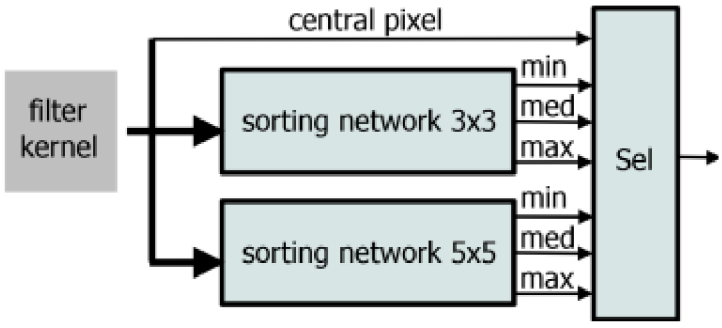


Obrázek 2.5: Hardwarová implementace řadicí sítě mediánového filtru pomocí prvků *Compare and Swap* a *Delay*. Zdroj: [8].

2.2.6 Adaptivní mediánový filtr

Adaptivní mediánový filtr poskytuje lepší výsledky než jeho jednodušší předchůdce. Během provádění algoritmu je vyhodnocováno, které pixely byly poškozeny šumem. To se zjišťuje porovnáváním bodů s jejich okolím. Velikost okolí je nastavitelná (obrázek 2.6). Obvykle se využívají dvě posuvná okna velikosti 3 x 3 a 5 x 5 [8].

Tento způsob filtrace se osvědčuje zvláště, pokud je obraz velmi zašuměný. Nevýhodou adaptivního mediánového filtru je ale jeho složitost, zvláště pokud je implementován hardwarově.



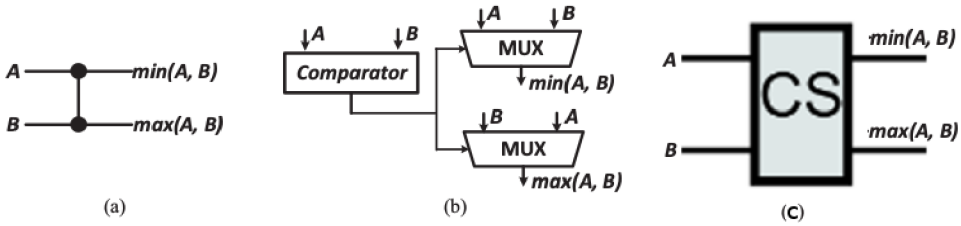
Obrázek 2.6: Adaptivní mediánový filtr. Zdroj: [8].

Obrázek 2.7: Ukázka aplikace mediánového a adaptivního mediánového filtru na obrázek ze 40 % zašuměný šumem sůl a pepř. Byly využity různé velikosti okolí posuvného okna. Zdroj: [10].

2.2.7 Komponenta Compare and Swap

Komponenta *Compare and Swap* je základním stavebním prvkem řadicích sítí a tím pádem podstatnou součástí např. mediánového filtru. V hardwaru bývá konvečně implementována jako komparátor a dva multiplexory vždy s dvěma vstupy – ony vstupy jsou porovnávaná čísla [4]. Výstup komparátoru je pak přiveden do multiplexorů a přepíná, který ze vstupů bude jejich výstupem. Tento výstup je pak vyšší, respektive nižší hodnota. Komparátor se skládá z osmi úplných jednobitoých sčítaček, přičemž pro implementaci jedné sčítačky je třeba 24 tranzistorů [2], jednobitový multiplexor se pak skládá z 6 tranzistorů. Snadno lze tedy provést odhad ceny implementace pro celou osmibitovou komponentu *Compare and Swap*: $24 * 8 + 6 * 8 * 2$, tedy 288 tranzistorů na komponentu. Tento hrubý odhad ceny neuvažuje případnou různou velikost tranzistorů nebo nějaké další pomocné tranzistory.

Standardní označení pro tuto komponentu bývá takové, jako je uvedeno na obrázku 2.8 (a). Toto označení však nevyhovuje potřebám práce, kdy bude třeba jednotlivé komponenty označovat a odkazovat na ně. Proto jsem pro totu práci zvolil označení uvedené na obrázku 2.8 (c), které vychází ze zdroje [8].



Obrázek 2.8: (a) Standardní označení pro komponentu *Compare and Swap*, (b) implementace komponenty *Compare and Swap*, (c) označení pro komponentu *Compare and Swap* v této práci. Zdroje: [4] [8].

2.3 Evoluční algoritmy

Pojem evoluční algoritmus zastřešuje velkou skupinu stochastických prohledávacích algoritmů. Tyto algoritmy se až do devadesátých let vyvíjely v postatě odděleně od sebe [1]. Řeší problém optimalizace. Zakladním cílem evolučního algoritmu je hledání optima zadané funkce f z množiny poteciálních řešení dané problémové domény [1]. Mějme například účelovou funkci f takovou, že:

$$f : R^n \rightarrow R, \tag{2.9}$$

Pak x_{opt} je optimem funkce f, pokud je cílem minimalizovat:

$$x_{opt} = arg\ min\ f(x). \tag{2.10}$$

Mezi evoluční algoritmy se řadí zvláště:

- genetické algoritmy,
- evoluční strategie,
- evoluční programování,
- genetické programování,
- diferenciální evoluce.

Společným rysem těchto algoritmů je to, že využívají množinu (v teminologii EA „populaci“) kandidátních řešení (dochází tedy k paralelnímu prohledávání stavového prostoru řešení) a že pro vytvoření nových jedinců populace používají oprátory, které jsou inspirovány biologickou evolucí. Popisem těchto oprátorů se zabývají následující kapitoly.

Při použití EA je v prvním kroku třeba navrhnout, jak budou jednotlivá řešení zadaného problému (jedinci) reprezentována. Je třeba, aby bylo řešení možno zakódovat jako řetězec symbolů (chromozom). Na tento řetězec jsou později aplikovány genetické operátory. Druhou podmínkou pro použití EA je schopnost ohodnocení kvality jedince (fitness hodnota).

2.3.1 Základní pojmy EA

Problematika zahrnující EA je značně rozsáhlá. Proto se v této podkapitole omezuji pouze na pojmy, které budou v další práci případně využity [9] [1]:

- *Jedinec, kandidátní řešení, fenotyp* – potenciální řešení daného problému.
- *Chromozom, genotyp* – struktura, jejíž interpretací získáváme kandidátní řešení[1]. Jedná se o jeden stav v prohledávaném prostoru řešení daného problému.
- *Fitness hodnota* – kvalita jedince.
- *Ohodnocovací fitness funkce* – funkce přiřazující jedinci fitness hodnotu. Jedná se o fukci, která je silně závisla na daném problému.
- *Populace* – multimnožina (může obsahovat několik stejných prvků) chromozomů.
- *Inicializace* – způsob, kterým je vytvořena první populace. Může být vytvořena náhodně, často je však vyžadována určitá struktura. První populaci lze také vytvořit vybráním již známých řešení. Tím je do programu zanesena určitá znalost o problému, která může jeho průběh zefektivnit.
- *Selekce* – mechanismus, pomocí kterého jsou vybíráni rodiče pro generování nových jedinců další populace.
- *Generace* – jedna iterace EA, kdy se z původní populace vytváří nová. Využívá se selekce, genetické oprátory pro tvorbu nových a nahrazení původních jedinců.
- *Genetické operátory* – zbůsob, jakým jsou z rodičů vytvářeni noví jedinci. Využívá se mutace genů a křížení chromozomů rodičů.
- *Ukončovací kritérium* – podmínka, při jejímž splnění dojde k ukončení algoritmu. Může se jednat o dosažení dostatečně kvalitního řešení, uplynutí nastaveného času, nebo například počtu generací.

2.3.2 Reprezentace problému

Reprezentace problému je silně závislá na konkrétní aplikaci. Využívá se binární, nebo reálný vektor, permutační kódování (které vyžaduje speciální genetické operátory). Dále může být problém zakódován jako strom (kódování výrazů a programů), nebo jako graf (obvody). V podstatě se jedná o to, nalézt takové kódování, které se nám pro reprezentaci problému nejvíce hodí. Dále se práce bude zabývat reprezentací problému řetězcem hodnot.

2.3.3 Genetické operátory

Jak bylo již řečeno, v EA dochází ke generování nových jedinců (potomků) na základě původních jedinců (rodičů). Tímto je zajištěna evoluce a tedy hledání kvalitního řešení problému. Proto, aby se vytvářeli potomci s potenciálně lepšími vlastnostmi, se využívá genetických operátorů. Následující podkapitoly tyto operátory představují.

[1] Některé evoluční algoritmy mezi fenotypem a genotypem nerozlišují. Jedná se o případy, kdy je již samotný genotyp zároveň řešením problému.

Selekce

Selekce je mechanismus zajišťující vybrání jedinců z původní populace, kteří budou sloužit jako rodiče pro novou generaci potomků. V podstatě jde o přidělení určitých pravděpodobností, s kterými budou vybráni. Je možno tuto pravděpodobnost stanovit *náhodně*, nedochází pak ale k u EA požadovanému selekčnímu tlaku. Další možností je technika nazývaná *Fitness proportional selection*, kdy je pravděpodobnost závislá na fitness hodnotě jedince, která je porovnávána s celkovou fitness hodnotou populace. Jedním ze způsobů implementace této techniky je tzv. vážená ruleta. U vážené rulety se sečte suma všech fitness hodnot všech jedinců v populaci. Na základě fitness hodnoty jedince a sumy je jedinci přiřazeno procentuální zastoupení v ruletě. Procentuální zastoupení může být vyjádřeno následovně:

$$p_i = \frac{f_i}{\sum_1^N f_i}, \tag{2.11}$$

kde p_i je pravděpodobnost, s kterou bude jedinec i vybrán, f_i je fitness hodnota jedince i a N je počet jedinců v populaci.

Dále je iterval $< 0, 1 >$ rozdělen tak, aby každá vypočítaná pravděpodobnost zabírala proporčně část intervalu. Pro zjištění rodiče pak stačí vygenerovat náhodné číslo z intervalu $< 0, 1 >$ s rovnoměrným rozdělením. Dalším způsobem selekce může být *Ranking*, kdy populaci seřadíme podle fitness hodnoty. Následně je jedincům přiřazen *rank*, což je hodnota kvality jedince v porovnání s ostatními jedinci. Pravděpodobnost výběru je následně vypočítána podle hodnoty *ranku*. Posledním zde popsaným způsobem selekce je *turnaj*. Při turnaji se náhodně vybere určité množství jedinců a ti jsou spolu porovnáváni [1]. Vítězové turnaje budou poté rodiči následující generace jedinců.

Mutace

Mutace přináší do populace nové vlastnosti, je tedy zásadním genetickým operátorem. Je řízena parametrem modelu p_{mut} v rozsahu <0, 1>, který stanovuje pravděpodobnost, že na daném místě v chromozomu dojde k mutaci. Pro každé místo v chromozomu je náhodně vygenerováno číslo v rozsahu <0, 1> a pokud je toto číslo menší než p_{mut}, dojde na daném místě k mutaci – původní hodnota je nahrazena novou (je však nutné, aby nová hodnota byla platnou hodnotou). U binárního vektoru může například dojít k překlopení hodnoty bitu na dané pozici. Vysoká pravděpodobnost mutace způsobuje, že se evoluční algoritmus změní v náhodné prohledávání stavového prostoru.

Křížení

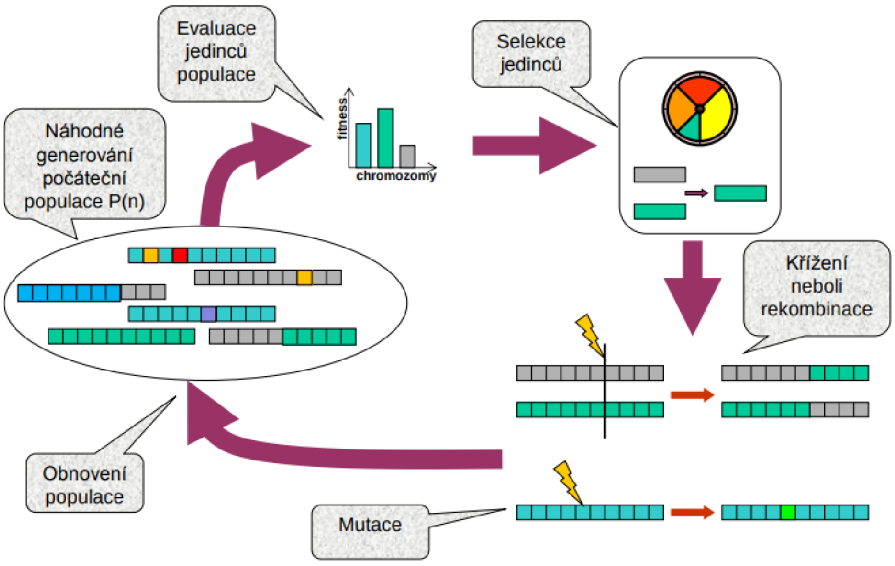
Jedná se o genetický operátor vycházející z párovaní jedinců v přírodě. Autory algoritmů byly vyvinuty speciální varianty křížení pro různé reprezentace problémů (stromem, permutacemi, grafy, ...). Pro reprezentaci problému řetězcem rozlišujeme dále popsané varianty.

Jednobodové, kdy se zvolí bod v chromozomech rodičů. V tomto bodě jsou jejich chromozomy rozděleny na dvě části, z nichž jednu si rodiče vymění a vytvoří tím dva nové potomky s novým unikátním chromozomem (pokud tedy rodiče nejsou dva stejní jedinci). Další možností je *vícebodové* křížení, kdy je bodů rozdělujících chromozom více. Poslední možností je *uniformní* křížení, kdy se podobně jako u mutace zavede parametr p_{cross} v rozsahu <0, 1> určující pravděpodobnost, že na daném místě v chromozomu dojde k výměně hodnot mezi rodiči.

2.3.4 Průběh EA

Evoluční algoritmy probíhájí následujícím způsobem (viz obrázek 2.9):

1. Generování počáteční populace jedinců (nejčastěji náhodně)
2. Ohodnocení každého jedince pomocí fitness funkce
3. Provádění následujících kroků, dokud není splněna ukončovací podmínka:
   (a) Selekce rodičů
   (b) Vytvoření potomků z rodičů
   (c) Vytvoření nové populace z původních a nových jedinců
   (d) Ohodnocení každého jedince pomocí fitness funkce
4. Vrácení nejlépe ohodnoceného jedince jako výstupu algoritmu



Obrázek 2.9: Průběh EA. Zdroj: [9]

2.3.5 Jednokriteriální vs. multikriteriální algoritmy

Standardní evoluční algoritmus hodnotí jedince na základě jediného kritéria. Tímto kritériem může být například vzdálenost nalezeného řešení od referenčního. Reálně řešené problémy však vyžadují pohled na problémy v závislosti na různých kritériích. Vyvstává však otázka: „Jak jednotlivá kritéria porovnat?“ Jednoduchým řešením může být vyčíslení těchto kritérií a nastavení určitých koeficientů „důležitosti“. Toto řešení však neodpovídá realitě, kdy pro nás mohou být zajímavé i kombinace, které bychom výše zmíněným způsobem nenalezli. Z tohoto důvodu byly vytvořeny sofistikovanější algoritmy [1].

Optimalizace jednoho kriteria

Zvolíme kritérium, které pro nás má největší důležitost. Pomocí EA optimalizujeme pouze toto kritérium. Ostatní kritéria jsou při této optimalizaci stanovena pouze jako omezení, která musí být splněna.

Agregovaná účelová funkce

Jedná se o určitý způsob váhování kritérií. Vezmeme všechny pro nás podstatná kritéria a přiřadíme jim váhy. Takto upravená kritéria sčítáme. Provádíme tím tedy redukci několika kritérií na jediné, tj.:

$$fitness = \sum_{i=1}^{k} w_i f_i(x), \tag{2.12}$$

kde k stanovuje počet kritérií a w_i je váha daného kritéria (cílem je maximalizovat fitness).

Nalezení Pareto fronty

Vícekriteriální přístup narozdíl od výše zmiňovaných přístupů dovoluje z jediného běhu evolučního algoritmu nalézt vícero různých řešení. Algoritmus hledá Pareto optimální řešení, která nejsou dominována žádným jiným řešením z pohledu použitých kritérií [1].

2.3.6 Pareto optimalita

Jedinec je Paretovým optimem, pokud neexistuje žádné kritérium, ve kterém by se mohl zlepšit, aniž by však snížil ohodnocení ostatních kritérií. Citace formálních definic z [1]:

1. Nechť $x, x' \in \Omega$ jsou různá řešení z uvažovaného prostoru řešení. Řekneme, že vektor $u = F(x) = (f_1(x), ..., f_k(x) = (u_1, ..., u_k))$ *dominuje* vektoru $v = F(x') = (f_1(x'), ..., f_k(x') = (v_1, ..., v_k))$, píšeme $u \preceq v$, právě když $\forall i \in \{1, ..., k\} : u_i \leq v_i \wedge \exists i \in \{1, ..., k\} : u_i < v_i$ (zde uvedeno pro případ minimalizace funkcí f_i).

2. Řešení $x \in \Omega$ nazýváme *Pareto optimem* na Ω, právě když neexistuje žádné $x' \in \Omega$, $x' \neq x$, pro které $v \preceq u$, kde $v = F(x')$, $u = F(x)$.

3. Pro daný MOP[3], $F(x)$, *Pareto množina* je definována jako $P^* := \{x \in \Omega \mid \neg\exists x' \in \Omega : F(x') \preceq F(x)\}$.

4. Pro daný MOP, $F(x)$ a P^* definujeme *Pareto frontu* jako $PF^* := \{u = F(x) | x \in P^*\}$.

V průběhu multikriteriálního EA určujeme, zda jedinec Pareto dominuje nad jinými jedinci. Dle výše uvedených definic tedy musí být *dominující* jedinec ve všech kriteriích alespoň stejně dobrý jako *dominovaný* jedinec a v alespoň jednom kritériu lepší. Hledáme jedince, kteří nejsou dominováni žádným jiným jedincem. Výsledkem jednoho běhu MO EA je tedy množina řešení.

Dobrý multikriteriální EA by měl mít také nasledující vlastnosti [1]:

- udržování diverzity populace,
- zachovávání nedominovaných jedinců v populaci,
- spění k Pareto optimálním řešením.

[3] Multikriteriální optimalizační problém

2.3.7 Genetické programování

Genetické programování vzniklo na konci 80. let minulého století a o jeho rozvoj se nejvíce zasloužil John Koza. Jedná se o verzi EA, která vytváří počítačové programy (spustitelné struktury) pro řešení určitých úloh. Genetické operátory jsou užívány přímo na tyto spustitelné struktury. Při zjišťování fitness hodnoty je program kandidátního řešení proveden pro určitou množinu vstupů a vyhodnocují se výsledky. Program může být reprezentován kódem programovacího jazyka, instrukcemi, syntaktickým stromem, grafem, nebo posloupností čísel.

Základní kostra programu genetického programování je následující:

1. Generování počáteční populace jedinců
2. Ohodnocení všech jedinců v populaci za užití fitness funkce
3. **while** není splněna ukončovací podmínka **do**:
   (a) Vytvoření nové populace
      i. Vybrání rodičů z populace a vytvoření jejich potomků za užití genetických operátorů
      ii. Vytvoření nové populace z původní populace a potomků
   (b) Ohodnocení všech jedinců v populaci za užití fitness funkce
4. Jedinec s nejlepší fitness hodnotou je výstupem algoritmu

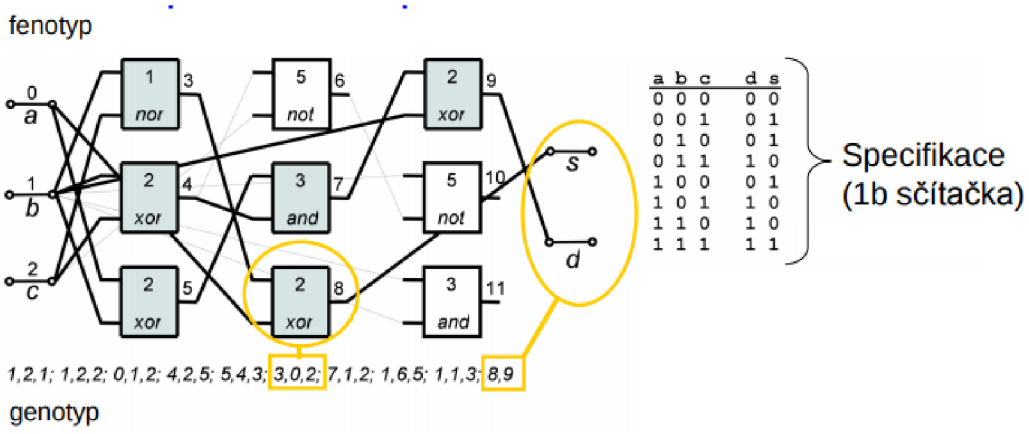
2.3.8 Kartézské genetické programování

Jedná se o variantu genetického programování, která se využívá zvláště na návrh obvodů. Fenotyp je reprezentován jako acyklický orientovaný graf v dvojrozměrné mřížce uzlů, chromozom je pak posloupnost celých čísel, kódující, jak je graf propojen a jaké prvky obsahuje. Při vyhodnocování je z genotypu vytvořen fenotyp, ve kterém jsou pouze využité uzly grafu. V úloze návrhu malých kombinačních obvodů je zadána cílová tabulka, která je ideální odezvou obvodu. Na vstup kandidátního obvodu jsou pak přiváděny jednotlivé kombinace vstupů a výstup je porovnáván s cílovou tabulkou (obrázek 2.10). Čím přesnější je odezva obvodu, tím je jedinci přiřazena lepší fitness hodnota. U větších obvodů je pak při evaluaci vybrán reprezentativní vzorek dvojicí vstup a výstup, který je poté využit k výpočtu fitness, protože evaluace všech možných kombinací je časově nezvládnutelná.

Před spuštěním algoritmu je CGP potřeba nastavit následující paramtry [9]:

- n_i – počet vstupů,
- n_o – počet výstupů,
- v – počet sloupců,
- u – počet řádků,
- n_a – počet vstupů uzlu,
- Γ – množina funkcí,
- *L-back* parametr – počet sloupců předcházejících j-tý sloupec, ze kterých může být vybrán vstup pro uzel v j-tém sloupci.

Bylo dokázáno, že v CGP hrají velkou roli mutace a to i mutace, které nezmění fitness hodnotu jedince – tzv. neutrální mutace. U křížení nebyla prokázána užitečnost [9].



Obrázek 2.10: Ukázka kandidátního řešení – genotyp (reprezentovaný trojicemi čísel vstup, vstup, funkce), jeho fenotyp (acyklický orintovaný graf) a referenční tabulka výstupů pro vyhodnocení fitness hodnoty. Nastavení parametrů $u = 3, v = 3, n_i = 3, n_o = 2, n_a = 2, L = 3, \Gamma = \{NAND(0), NOR(1), XOR(2), AND(3), OR(4), NOT(5)\}$. Zdroj: [9]

Kapitola 3

Evoluční aproximace obrazových filtrů – návrh

Následující kapitola se zabývá nastíněním problému řešeného v této diplomové práci. Formuluje problém, navrhuje způsoby jeho řešení, navrhuje a popisuje experimenty, zdůvodňuje vybrané postupy.

Kapitola 3.1 popisuje motivaci, proč je vhodné téma této práce vůbec řešit a odkazuje na zdroje, zabývající se podobnou problematikou. V kapitole 3.2 se zabývám podrobnějším popisem metody, stanovuji otázky, které bych v této práci rád zodpověděl a zhruba popisuji způsob provedení aproximace. Tématem kapitoly 3.3 je pak popsání jednotlivých kroků nutných k dosažení stanovených cílů. Kapitola 3.4 pak rozebírá implementaci řadicí sítě, analyzuje možnosti a způsoby aproximace jednotlivých komponent a následně vybírá způsob aproximace. Následuje kapitola 3.5, která hlouběji popisuje návrh evolučního algoritmu, který bude využit. Diskutuje způsoby prohledávání stavového prostoru řešení, navrhuje řešení základních problémů, se kterými je třeba se při implementaci evolučního algoritmu vypořádat. Kapitola 3.6 popisuje objektivní způsob ověření kvality u v budoucnu nalezených implementací obrazových filtrů pomocí testovací datové sady. Tématem kapitoly 3.7 jsou technologie, které budou použity pri implementaci všech nutných programů. Dále zde popisuji další využité podpůrné technologie. Poslední kapitola 3.8 navrhuje experimenty, které budou v rámci této práce provedeny, a dále vypisuje podrobný postup při jejich vykonávání.

3.1 Motivace

Na obrazové filtry implementované jako vestavěné obvody klademe určité požadavky. Měly by zabírat malý prostor, mít co nejmenší spotřebu energie a zároveň by měly mít dostatečnou propustnost. Dále by měly využívat malé posuvné okno, protože velké posuvné okno znamená významné zvyšování zabrané plochy na čipu [12]. Samozřejmostí je, že by měly umět co nejlépe filtrovat.

Během let docházelo k pokusům filtry implementovat jinak než konvenčním návrhem. Byly vyvinuty a úspěšně otestovány postupy, které dosahovaly dobrých výsledků. Aproximovány byly mediánové, Gaussovy a Sobelovy filtry, kdy byly na úrovni trasistorů, hradel a RTL obvodů aproximovány sčítačky, násobičky komparátory, nebo samotný kód simulace obvodu. Byly použity ruční i automatizované techniky. Za využití evolučních algoritmů –

které se snažily optimalizovat určité kritérium – byly nalezeny nekonvenční obrazové filtry s vlastnostmi mnohdy lepšími, než mají konvenční implementace [8].

3.2 Návrh metody

Mějme konveční mediánový obrazový filtr 3 x 3. Úkolem je upravit stávající implementaci tohoto filtru za využití technik aproximativních výpočtů. Cílem je najít lepší kompromis mezi kvalitou filtrace a cenou implementace než vykazují stávající řešení. Navržená metoda využije aproximované implementace základních komponent filtrů.

Otázky zní:

- Jaké komponenty nahrazovat?
- Na jakých místech komponenty nahrazovat?
- Kterými implementacemi komponent původní komponenty nahrazovat?
- V jakých implementacích obrazových filtrů bude mít nahrazovaní nejlepší výsledky?
- Jakým způsobem komponenty nahrazovat?

Výše zmíněné nahrazování bude prováděno pomocí evolučních algoritmů a pro potřeby porovnání i hrubou silou. V tomto případě bude vytvořeno několik implementací metod aproximace rozdělených podle způsobu aproximace. Budou rozděleny podle způsobu prohledávání stavového prostoru řešení. Tyto implementace a přístupy pak budou experimentálně porovnány.

Zjištění co nejvýhodnějšího nastavení parametrů výše popsaných implementací bude součástí experimentální práce.

3.3 Kroky navržené metody

Vstupem metody bude obrazový filtr a množina aproximovaných komponent, které budou použitelné místo původních komponent. Výstupem pak bude aproximovaný filtr. Metoda bude ověřena pomocí standardního mediánového filtru.

Zvolím funkce, které budou nahrazovány aproximovanými implementacemi. V potaz připadají funkce *Add*, *Compare and Swap* a multiplexory. Dále bude zvolen způsob aproximace komponent. Následně bude implementován evoluční algoritmus a všechny potřebné skripty pro jeho řízení. Tyto programy použiji pro aproximaci referečního filtru, provedu experimenty. Konečně vyhodnotím výsledky a srovnám všechny využité přístupy z různých hledisek.

3.4 Volba aproximovaných komponent a způsob aproximace

Jak bylo uvedeno v teoretické části této práce, kritickou částí mediánového filtru je nepochybně řadicí síť. Ta se skládá z komponent *Compare and Swap* a *Delay*. Komponentu *Delay* není vhodné aproximovat, zato však *Compare and Swap* ano. Na aproximace této komponenty se můžeme dívat z různých úrovní. Například na nejnižší úrovni lze provádět odpojování některých tranzistorů. Dále víme, že *Compare and Swap* se skládá z komparátoru a dvou multiplexorů. Nabízí se tedy aproximovat komparátor pomocí aproximovaného

sčítání. V rámci této práce jsem se ale rozhodl provádět aproximaci redukcí bitové šířky, na kterou komparátor reaguje. Tímto v podstatě simuluji aproximaci na jakékoli úrovni a konkrétní implementace je pak ponechána k diskuzi.

Je třeba zmínit, že bude aproximován pouze komparátor. Multiplexory zůstávají aproximací nedotčené. Tento fakt dobře ilustruje obrázek 2.8. I když tedy porovnání hodnot nebude muset proběhnout přesně, na výstupu z komponenty budou vždy stejně přesná čísla jako ta, která byla na vstupu.

Pojďme se nyní zamyslet nad úsporou, kterou aproximace komponenty *Compare and Swap* přináší. Předpokládejme implementaci komparátoru vytvořenou z osmi úplných jednobitových sčítaček, kdy každá hardwarová implementace takové sčítačky vyžaduje 24 tranzistorů [2]. Implementace jednoho osmibitového multiplexoru pak zabírá 48 tranzistorů [2], tedy pro dva multiplexory v komponentě hovoříme o 96 tranzitorech pro ně vyhrazených. Kompletně komponenta *Compare and Swap* vyžaduje na svou implementaci odhadem 288 tranzistorů. Podle výše uvedeného návrhu aproximace pomocí redukce bitové šířky komparátoru se tedy lze zabývat až 192 tranzistory, které mohou být odstraněny. Vždy při redukci bitové šířky o jeden bit dochází k odstranění jedné sčítačky tedy 24 tranzitorů. U osmibitových čísel můžeme rozlišit několik konkrétních úrovní aproximace. Nulová aproximace vyjadřuje přesnou komponentu, při postupné redukci bitové šířky od nejméně významného bitu se pak lze dostat přes úrovně 1, 2, ... až na úroveň aproximace 8, tedy až k úplnému odpojení komponenty z řadicí sítě. V této situaci zůstane pouze 96 tranzitorů implementujících multiplexory. Pokud budeme mít přesný popis aproximace celé řadicí sítě, můžeme výše uvedeným způsobem vypočítat úsporu tranzistorů pro celý filtr.

3.5 Návrh evolučního algoritmu

Tato poněkud rozsáhlejší kapitola má za úkol navrhnout řešení základních problémů, které se v této práci týkají evolučních algoritmů. Odkazují na rozebírané teoretické informace a využívají je k popsání návrhu všech řešení.

Podkapitola 3.5.1 rozebírá prohledávaný stavový prostor řešení, navrhuje optimalizace jeho prohledávání. Druhá podkapitola 3.5.2 řeší reprezentaci problému, přirovnává ji k již používaným řešením a popisuje návrh v této práci použité reprezentace. Poslední podkapitola 3.5.4 se zabývá popsáním návrhu výpočtu fitness hodnoty u jedince. Dále popisuje detaily tohoto problému a navrhuje jejich řešení.

3.5.1 Možnosti prohledávání stavového prostoru

V předcházející kapitole jsem nastínil možnosti aproximace řadicí sítě mediánového filtru. Tím vzniká stavový prostor možných implementací za užití redukce bitové šířky. Při 9 úrovních možné aproximace[1] a 19 místech[2] získáváme 19^9 možných implementací – vyčísleno přesně máme 322 687 697 779 možností. To je stavový prostor o významné velikosti, úplné prohledávání hrubou silou tedy nepřipadá v úvahu. Je třeba do procesu prohledávání prostoru zavést heuristický přístup.

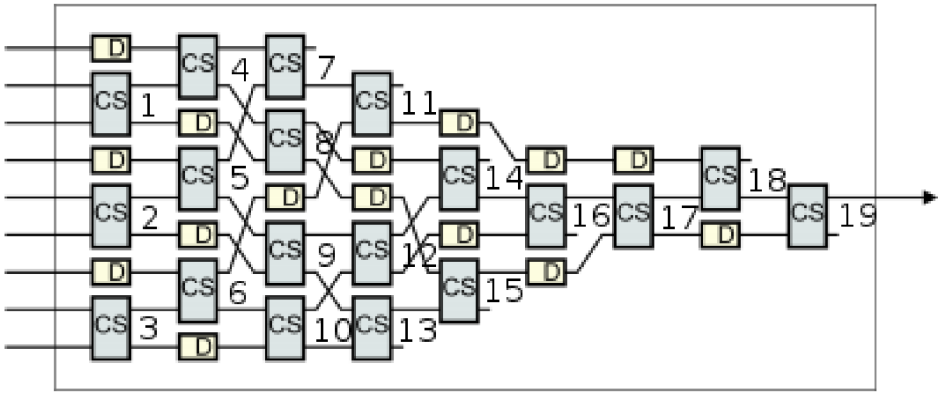
Jednou z možných optimalizací je například úprava prohledávání stavového prostoru tak, že se komponenty neaproximují, ale přímo odpojují. Tím se prostor značně omezí na 2^{19} stavů, tedy na 524 288 stavů. Takto velký prostor již lze prohledat hrubou silou. Další možností, která je pro tuto práci absolutně zásadní, může být použití evolučního algoritmu.

[1] Možné aproximace jsou odstranění 0 – 8 bitů.

[2] V implementaci na obázku 2.5 je 19 *Compare and Swap* obvodů.

3.5.2 Reprezentace řešení

Základním problémem při návrhu evolučního algoritmu je způsob zakódování řešení do chromozomu jedince. V kapitole 2 bylo popsáno, jak se tento problém řeší u CGP. Návrh zakódování pro tuto práci je jednodušší než CGP – není třeba hledat propojení komponent. U řadicí sítě se nebudou měnit žádné jiné komponenty než ty, které budou nahrazovány, a propojení zůstane zachováno. Problém tedy může být řešen genetickým algoritmem. Řešení pak může být zakódováno jako posloupnost celých čísel – pro každou komponentu jedna hodnota. Každé číslo bude určovat konkrétní úroveň aproximace, kterou byla původní komponenta prošla. To znamená, že chromozom bude tvořit 19 číslic, kde se budou vyskytovat na každé pozici hodnoty od 0 do 8. Chromozom se tedy omezí tak, že bude obsahovat pouze a jen skutečně nutné informace. Označení jednolitvých pozic je ukázáno na obrázku 3.1, kdy označování probíhalo shora dolů a poté zleva doprava. Číslo označující komponentu zároveň určuje polohu její reprezentace v chromozomu jedince.



Obrázek 3.1: Číselné označení jednotlivých aproximovatelných komponent. Čísla vyjadřují i pozici komponenty v chromozomu.

3.5.3 Genetické operátory

Mutace bude prováděna následujícím způsobem. Bude nastavena pravděpodobnost mutace, pokud pak v evolučním algoritmu k mutaci dojde, budou mutovány 2 náhodně zvolené geny. *Selekce* pak bude provedena turnajovou selekcí se zavedeným elitizmem. To znamená, že nejlepší nalezený jedinec bude vždy součástí následující populace a stejně tak i jeho potomek – mutant. Dále budou z populace vybráni vždy 4 náhodní jedinci vstupující do turnajové selekce. Z těchto 4 jedinců pak budou vybráni 2 nejlepší, kteří projdou operátorem křížení (při využití pravděpodobnosti křížení) a následně pak operátorem mutace (při využití pravděpodobnosti mutace). *Křížení* bude probíhat jednobodově, kdy si rodiče vymění část chromozomu.

3.5.4 Výpočet fitness

Za účelem vyhodnocení kvality filtrace kandidátního filtru byl zvolen referenční obrázek (viz obrázek 3.2). Tento obrázek bude následně zašuměn určitým druhem šumu – t.j. poškozen. Z jedince, u kterého budeme hodnotit jeho fitness hodnotu, bude vytvořen filtr,

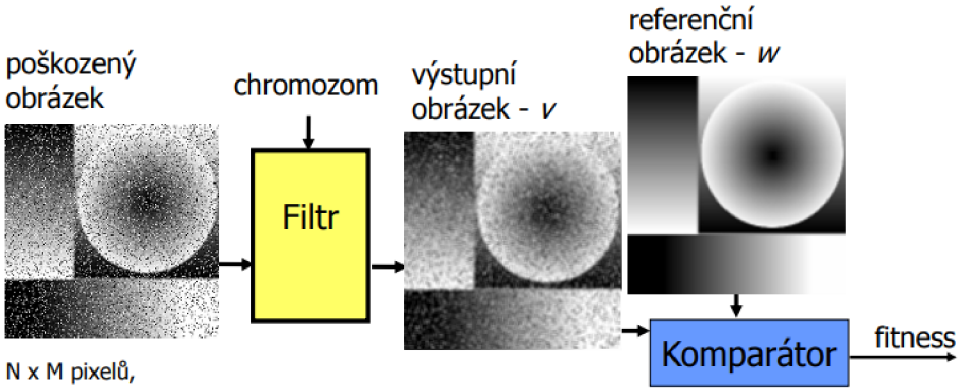
který bude použit na odstranění šumu v poškozeném obrázku. Následně bude vyfiltrovaný obrázek porovnán s referečním obrázkem a bude vypočítán rozdíl. Tento rozdíl bude suma suma rozdílů hodnot pixelů přes všechny pixely obrázku. Výsledek bude následně odečten od hodnoty maximálního možného rozdílu (velikost referenčního obrázku 256 x 256 pixelů násobeno maximálním rozdílem 256, tedy 16 777 216) tak, aby při běhu evolučního algoritmu docházelo k hledání maximální fitness. Například ideální fitness hodnota tedy bude 16 777 216, znamenající žádný rozdíl vyfiltrovaného obrazu oproti referenčnímu obrázku. Cílem evoluční optimalizace bude maximalizovat tuto fitness. Vyhodnocování fitness hodnoty ilustruje obrázek 3.2. Vzorec pro výpočet fitness jedince tedy bude:

$$fitness = F_{MAX} - \sum_{i=1}^{N}\sum_{j=1}^{M}|v(i,j) - w(i,j)|, \tag{3.1}$$

kde F_{MAX} je konstanta 16 777 216 (maximální rozdíl mezi dvěma obrázky v tónech šedé o rozměrech 256 x 256 pixelů), N je výška obrázku v pixelech, M je šířka obrázku v pixelech, $v(i,j)$ je funkce vracející hodnotu pixelu zadaného souřadnicemi i a j z vyfiltrovaného obrázku a $w(i,j)$ je funkce vracející hodnotu pixelu zadaného souřadnicemi i a j z referenčního obrázku.

Další možné zlepšení výpočtu fitness hodnoty by mohlo být zavedení teoreticky lepšího způsobu hodnocení kvality filtru, jako například Peak Signal to Noise Ratio (PSNR):

$$PSNR = 10\log_{10}\frac{255^2}{\frac{1}{MN}\sum_{i,j}(v(i,j) - w(i,j))^2}. \tag{3.2}$$



Obrázek 3.2: Způsob vyhodnocení fitness hodnoty kandidátního filtru. Zdroj: [9], upraveno pro vlastní potřeby. Poškozený i refereční obrázek byl skutečně použit při implementaci evolučního algoritmu.

Evoluční algoritmus s takto nastavenou fitness funkcí však bude hledat nejkvalitnější možnou implementaci bez podmínek na aproximaci. Lze očekávat, že pokud bude spuštěn nalezne konveční řešení tedy řešení, kdy žádná z komponent není aproximovaná. Je třeba zavést optimalizaci dalšího kritéria. Tímto kritériem je cena, kterou řadicí síť filtru potřebuje ke své implementaci. V teoretické části této práce byly zmiňovány způsoby, kterými je možné další kritérium vnést do evolučního algoritmu. Ze zmíněných možností jsem pro potřeby této práce zvolil techniku *fitness penalization*, tedy agregovanou účelovou funkci (viz vzorec

2.12), spočívající v penalizaci finess hodnoty při nedodržení omezujicího kritéria, kterým je v tomto případě předepsaná maximální cena filtru.

3.6 Způsob ověření kvality řešení

Jak už bylo uvedeno, výstupem algoritmu bude implementace mediánového filtru, splňující omezující požadavky. U takového filtru však bude třeba ověřit jeho kvalitu. K tomuto účelu poslouží datová sada testovacích obrázků [3]. Tato sada obsahuje 24 různých obrázků, které jsou poškozeny šumem sůl a pepř. Intenzita tohoto šumu se pohybuje od 5 % do 75 %, vždy s 5% odstupem. To znamená, že celá sada obsahuje celkově 360 obrázků s 15 úrovněmi šumu. Nalezený aproximovaný filtr bude využit k profiltrování celé této sady, výsledky filtrování budou porovnány s originálními, nepoškozenými obrázky a bude vypočítána průměrná chyba na pixel obrazu. Takto získáme hodnotu, pomocí které budeme moci zjistit kvalitu filtrace a jednotlivé filtry mezi sebou objektivně porovnávat.

3.7 Technologie

Pro efektivní práci s obrazovými daty bude využita knihovna *OpenCV*[4] zaměřená na zpracování obrazu. Jedná se o multiplaformní knihovnu, která je zdarma použitelná pod open-source BSD licencí. Je možné ji využívat v jazyce C/C++ a také v jazyce Python a Octave. Potřebné programy budu psát v jazyce C++ a dále budu ve výpočetně nenáročných úlohách používat jazyk Python.

Existuje řada použitelných implementací evolučních algoritmů. Pro potřeby této práce jsem se rozhodl využít již existující implementaci genetického algoritmu, se kterou jsem se setkal v předmětech *Aplikované evoluční algoritmy* a *Biologií inspirované počítače* při mém studiu na Fakultě informačních technologií Vysokého učení technického v Brně[5]. Tuto implementaci pak budu upravovat podle potřeb této práce. Pro následné vyhodnocování výsledků experimentů využiji program LibreOffice Calc[6] a jeho statistické funkce.

3.8 Návrh experimentů

Následující podkapitoly popisují návrh experimentů prováděných v rámci této diplomové práce.

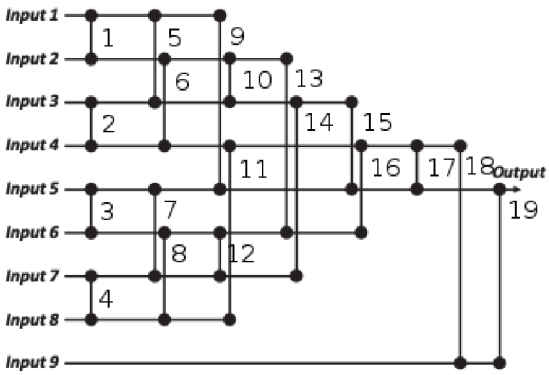
3.8.1 Experiment 1 – Analýza citlivosti jednotlivých komponent na aproximaci

Prvním navrhnutým experimentem bude zjištění citlivosti jednotlivých komponent na aproximaci v závislosti na její poloze v řadicí síti. Je předpokládatelné, že vliv pozice na důležitost komponenty při řazení bude růst s přibližováním se k výstupu filtru. Zároveň bude cílem zjistit, která pozice ve filtru je nejvíce citlivá na aproximaci. Druhým cílem tohoto experimentu pak bude porovnání dvou implementací mediánového filtru a zjištění, která z nich je vhodnější pro aproximaci.

[3] Datová sada dostupná na adrese: http://www.fit.vutbr.cz/~vasicek/imagedb/.cs?lev=5&knd=corrupted
[4] https://opencv.org/
[5] Implementace GA použita s laskavým svolením autora Dr. Michala Bidla.
[6] https://cs.libreoffice.org/

Průběh experimentu

Komponenty na každé pozici (jedna po druhé 1 - 19) budou aproximovány všemi úrovněmi aproximace (1 - 8). Tímto způsobem vznikne 8 * 19 tedy 152 rozdílných implementací aproximového mediánového filtru. Každá takto vzniklá implementace pak bude použita k profiltrování testovacího datasetu. Bude vypočítána průměrná chyba na pixel obrazu při aproximaci určitého místa ve filtru (tedy jedné konkrétní komponenty v úrovních aproximace 1 - 8). Bude zjištěno, která místa ve filtru jsou citlivá na aproximaci a tato místa budou porovnána s ostatními místy. Takto popsaný experiment proběhne u dvou implementací mediánového filtru a bude zjištěno a odůvodněno, který z těchto filtrů je vhodnější pro aproximaci. Bude využit filtr uvedený na obrázku 2.5 a také implementace filtru uvedená na obrázku 3.3. Lépe vyhodnocený filtr pak bude použit u experimentů, které budou na zde popisovaný experiment navazovat.



Obrázek 3.3: Druhá implementace mediánového filtru použitá v experimentu 1 s očíslovanými pozicemi komponent *Compare and Swap*. Zdroj: [4].

3.8.2 Experiment 2 – Prohledání části prostoru hrubou silou

Druhým experimentem pak bude pokus o prohledání alespoň části prostoru řešení pomocí hrubé síly. Prostor bude omezen tím, že budou uvažovány pouze úrovně aproximace 0, nebo 8. To znamená, že daná komponenta nebude aproximována vůbec, nebo bude odpojena. Cílem bude nalezení nejlepších tímto způsobem hledaných řešení, při omezení maximální ceny implementace.

Průběh experimentu

V experimentu 1 vybraný filtr bude použit v zde popisovaném i ve všech dalších popisovaných experimentech. Postupně budou získávány všechny možné implementace mediánového filtru, které mohou být získány odpojováním komponent *Compare and Swap*. U těchto filtrů bude vyhodnocena odezva na obrázek využívaný při výpočtu fitness hodnoty. Poté bude vybráno 20 filtrů, s nejlepší odezvou, zároveň však splňujících 20 omezení na cenu, která je definována počtem využitých sčítaček nutných pro realizaci implementace filtru (152, 144, ... až 0 – každé snížení ceny o 8 sčítaček v postatě odpovídá odpojení jedné komponenty *Compare and Swap*). Takto získané implementace budou opět použity k filtrování zašuměných obrázků v testovacím datasetu. Budou vypočítány průměrné chyby na jeden

pixel obrazu pro každou implementaci. Nalezené filtry budou použity ve vyhodnocení experimentů a pro další porovnání.

3.8.3 Experiment 3 – Aproximace všech komponent současně stejnou úrovní aproximace

Třetí experiment pak bude proveden z praktických technologických důvodů. Je technologicky jednodušší vytvořit filtr, kde mají všechny prvky (ze kterých se filtr skládá) stejnou hardwarovou implementaci. Proto zde přistoupím k zjištění kvality filtrů, kde budou mít všechny komponenty stejnou úroveň aproximace.

Průběh experimentu

Mediánový filtr bude postupně aproximován přes všechny úrovně aproximace. Budou však vždy aproximovány úplně všechny komponenty *Compare and Swap*. Vznikne tím 8 implementací (6 unikátních, neaproximovaný filtr a filtr s úplně odpojenými komponentami už byly analyzovány, budou však součástí i toho experimentu pro možnost porovnání). Tyto implementace pak budou vyhodnoceny a závěrem použity pro porovnání všech nalezených implementací filtrů.

3.8.4 Experiment 4 – Optimalizace parametrů evolučního algoritmu

Jako čtvrtý experiment jsem zvolil optimalizaci parametrů evolučního algoritmu, který bude následně použit pro optimalizované prohledávání stavového prostoru. Nastavení počátečních parametrů je věc zásadně ovlivňující efektivitu běhu evolučního algoritmu. Parametry, které mohou být optimalizovány, jsou následující:

- pravděpodobnost mutace vybraného chromozomu,
- počet mutovaných genů,
- pravděpodobnost křížení,
- počet jedinců v turnajové selekci,
- velikost populace,
- maximální počet generací.

Po základních experimentech s modelem byly nastaveny následující parametry: počet mutovaných genů – 2, pravděpodobnost křížení – 70 %, počet jedinců v turnajové selekci – 4. Optimalizované parametry pak budou pravděpodobnost mutace a velikost populace.

Průběh experimentu

Počet evaluací u experimentu bude omezen na 10 000. V záslosti na velikosti populace se pak dopočítá a nastaví potřebný počet generací tak, aby došlo po běhu evolučního algoritmu ke splnění této podmínky. Poté bude zkoumán vliv úpravy pravděpodobnosti mutace a velikosti populace na výstupní fitness hodnoty. Základní experimenty s modelem mne přesvědčily, že hodnoty mutace pod 40 % nejsou z hlediska maximalizace fitness zajímavé. Proto budou experimenty začínat na 40% pravděpodobnosti mutace a po deseti procentech se v každém kole experimentů zvyšovat až na hodnotu 100 %. Stejně tak jsem zjistil,

že velikost populace, nižší než 30 jedinců nepřináší žádané výsledky a vyžaduje větší počet evaluací. Proto hodnota velikosti populace v experimentech bude začínat na 30 jedincích a po deseti jedincích se bude zvyšovat až na hodnotu 80 jedinců. Evoluční algoritmus s každým unikátním nastavením parametrů bude spuštěn 50krát. Následně budou získané maximální hodnoty fitness zprůměrovány a bude nalezena nejvyšší průměrná hodnota. Tato hodnota bude představovat nejlepší nalezené nastavení parametrů algoritmu a bude využita ve finálním experimentu.

3.8.5 Experiment 5 – Prohledávání prostoru pomocí genetického algoritmu

Nejdůležitějším a závěrečným experimentem bude spuštění samotného evolučního algoritmu pro určené scénáře aproximace. Parametry algoritmu budou nastaveny podle hodnot získaných v experimentu 4. Dále bude do parametrů evolučního algoritmu přidán parametr maximální ceny, kterou by vyevolvované řešení nemělo přesáhnout – `MAX_AREA`. Také bude třeba nastavit hodnotu penalizace fitness za nedodržení maximální ceny (cena bude moci být menší, ale ne větší, než nastavená hodnota). Tato hodnota bude určena empiricky při experimentech s již hotovým modelem.

Průběh experimentu

Počet evaluací finess hodnot jediců bude omezena na 50 000. Budou nastaveny optimální hodnoty parametrů evolučního algoritmu. Hodnota maximální ceny bude na počátku experimentu nastavena na $8 * 19$, tedy 152 (jedná se o maximální cenu, kterou může zabírat jedna komponenta a celkový počet komponent ve filtru). Bude provedeno vždy 10 běhů evolučního algoritmu se stejným nastavením. Po deseti bězích se hodnota maximální přípustné ceny sníží o 8, dokud nebude dosažena hodnota 0. To znamená, že celkově bude provedeno 200 běhů evolučního algoritmu přes 20 unikátních nastavení maximální přípustné ceny. Následně nazené filtry použiji k profiltrování tastovacího datasetu a pro každé nastavení maximální ceny vyberu jeden filtr, který bude mít nejlepší výsledek. Tím budu aproximovat Pareto množinu Pareto optimálních řešení.

Kapitola 4

Implementace

Tato kapitola se zabývá implementací všech potřebných programů. Popisuje jejich vývoj, zdůvodňuje rozhodnutí, která byla učiněna při vývoji, a vysvětluje jejich význam. Informuje o důležitých částech programů.

Kapitola 4.1 popisuje implementaci lokálního filtru, simulaci řadicí sítě a aproximovatelných komponent *Compare and Swap*. Kapitola 4.2 se zabývá implementací evolučního algoritmu, hlouběji probírá fitness funkci. Tématem kapitoly 4.3 je popsání pomocných programů, které byly implementovány v rámci řešení experimentů.

4.1 Implementace filtru, řadicí sítě a komponent

Podstatnou částí implementace programů v této práci bylo vytvoření softwarové simulace hardwarové implementace řadicí sítě. Zároveň však bylo třeba, aby softwarová implementace jednolitvých komponent *Compare and Swap* byla aproximovatelná.

Komponentu *Compare and Swap* simuluje funkce `aproxCmpswp()`. Jejím vstupem jsou dvě osmibitová čísla `a` a `b`. Dalším vstupem do funkce je ale i hodnota `aproxLevel` určující stupeň aproximace komponenty. Simulace aproximace v komponentě probíhá bitovým posunem vpravo o hodnotu `aproxLevel`. Následuje porovnání výsledků a případné prohození hodnot, pokud tomu tak má být. Tím je dosaženo simulace přesně takové funkce, kterou provádí hardwarová komponenta *Compare and Swap* na úrovni aproximace `aproxLevel`.

Samotnou řadicí síť jsem implementoval ve funkci `getMedianSWP()`. Vstupy do této funkce jsou následující: ukazatel na pole hodnot vstupů `i` a ukazatel na pole hodnot urovní aproximace jednolivých komponent `aproxLvl`. Výstupem této funkce je pak hodnota získaná postupnou aplikací funkcí `aproxCmpswp()` podle schématu řadicí sítě. Při nulové aproximaci u všech komponent se jedná o medián vstupních hodnot, při aproximaci však hodnota mediánu na výstupu není garantována. Úplně stejným způsobem jsem implementoval i druhou řadicí síť, využívanou v experimentu 1. Tuto implementaci simuluje funkce `getMedianSWP2()`.

Celkovou funkci filtru provádí funkce `filter()`. Ta na vstup dostává chromozom jedince, což jsou v podstatě úrovně aproximace pro všechny komponenty *Compare and Swap*. Dále připravuje všechna potřebná pole a prochází filtrovaný obraz pomocí posuvného okna. Spouští simulaci aproximované řadicí sítě a její výstupy ukládá jako profiltrovaný obraz.

4.2 Evoluční algoritmus

Implementaci samotného evolučního algoritmu jsem se souhlasem jeho autora přejal. Skládá se ze souborů `main-opt.cpp`, kde nachází implementace potřebných funkcí, a `params.h`, kde můžeme nalézt parametry genetického algoritmu, definici typu chromozomu a prototypy funkcí. Tuto implementaci bylo potřeba upravit pro potřeby této práce. Mezi změny patří nastavení parametrů genetického algoritmu a změna délky chromozomu na 19 prvků. Dále bylo třeba změnit způsob generování počáteční populace tak, aby chromozomy jedinců obsahovaly pouze validní hodnoty, a stejným způsobem i mutaci prvků v chromozomu. Následně byla do implementace evolučního algoritmu přidána funkce `init()`, která načítá potřebný zašuměný i přesný referenční obrázek a připravuje struktury pro uložení výsledku. Nevýznamnější změnou však byl výpočet fitness.

4.2.1 Fitness funkce

Výpočet fitness hodnoty na základě chromozomu jedince jsem implementoval ve funkci `fitness()`. Tato funkce má na vstupu vyhodnocovaný chromozom a výstupem z této funkce je vypočítaná fitness hodnota. V této funkci je volána funkce pro výpočet ceny, kterou výsledná implementace zabírá a tato hodnota je uložena. Dále je zde spuštěna výše popisovaná funkce `filter()`, díky ní je získán vyfiltrovaný obraz, který je následně porovnán s referenčním. Od výsledné hodnoty rozdílu je pak odečtena penalizace za nedodržení předepsané vyžadované ceny, kterou implementace filtru zabírá. Tato penalizace je však odečtena pouze v případě, že cena filtru přesahuje maximální stanovenou cenu. Fitness jedinců zabírajících menší než požadovanou cenu není penalizována (tento způsob penalizace finess hodnoty se projevil jako zásadní, viz výsledky experimentů).

4.3 Pomocné programy

V rámci implementace všech potřebných programů jsem kromě upraveného evolučního algoritmu vytvořil i několik pomocných programů pro provádění experimentů a automatizaci procesů, které by mi při ručním provádění zabraly příliš času. Jedná se zejména o následujicí programy:

- `filterByChromosome.cpp` – program pro provedení filtrování celého testovacího datasetu na základě chromozomu, který je mu předán v parametrech. Jeho výstupem je suma všech rozdílů všech testovaných obrazů. Z této hodnoty lze vypočítat např. průměrnou chybu na pixel obrazu.
- `bruteSearch.cpp` – program pro provedení experimentu 2. Vypočítává rozdíly referenčního a zašuměného referenčního obrazu pro jednotlivé možné odpojení komponent. Výsledky vypisuje do souboru.
- `testByComponent.py` – skript v jazyce python, který provádí experiment 1. Spouští program `filterByChromosome` a do parametrů mu zadává potřebné chromozomy.
- `findBruteFilters.py` – skript pro prohledání filtrů získaných hrubou silou a nalezení nejlepších implementací pro jednotlivé požadované velikosti ceny. Tento skript je využíván v experimentu 2.

- `replaceAndLaunch.py` – skript využívaný v experimentu 4. Má za úkol vypočítávat počet generací v závisloti na velikosti populace a počtu požadovaných evaluací. Tyto hodnoty i s pravděpodobností mutace nastavuje jako parametry evolučního algoritmu v souboru `params.h`, kód programu kompiluje a tento algoritmus následně s těmito parametry spouští. U experimentu 5 plní stejnou funkci, mění však jiné parametry.
- `readFiltersAndCheck.py` – skript pro zpracování dat získaných při experimentu 5.

Výše uvedené programy jsou vyžívány v případě potřeby napříč všemi experimenty. Ve výčtu jsou však explicitně uvedeny experimenty, pro které jsem ony programy vytvořil.

Kapitola 5

Výsledky experimentů

Následující kapitoly 5.1, 5.2, 5.3, 5.4 a 5.5 se zabývají výsledky provedených experimentů. Kapitola 5.6 porovnává filtry nalezené v jednotlivých experimentech.

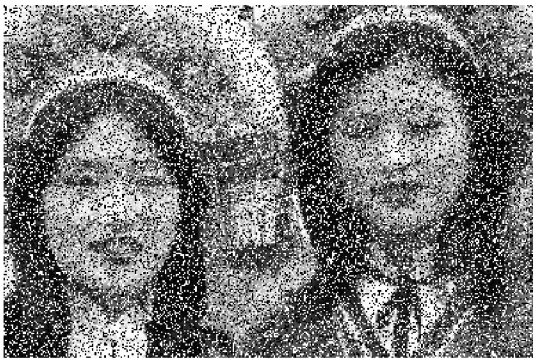
5.1 Experiment 1

Prvním a intuitivním způsobem, jak zhodnotit kvalitu filtrace při aproximaci komponent v mediánovém filtru v tomto experimentu, bylo podívat se na výstupy a vizuálně zhodnotit kvalitu filtrace při aproximaci. Na obrázku 5.1 můžeme vidět šest různých obrázků. Už na první pohled lze vidět, že nízká úroveň aproximace u jediné komponenty nevnáší do filtrace okem zaznamenatelné změny. Pokud však přecházíme do vyšších úrovní aproximace (6 a výše), aproximace se na vizuální kvalitě filtrace začíná hlouběji podepisovat. Zároveň však významě záleží na pozici aproximované komponenty ve filtru. Obrázek 5.1 ukazuje, že při vyšší úrovni aproximace u komponenty 16 dochází k horší filtraci než při aproximaci komponenty 1.

Tato empiricky zjištěná fakta dokazuje statistické vyhodnocení výsledků experimentu. Graf na obrázku 5.2 ukazuje, že vliv aproximace na kvalitu filtrace je dramatický a s vyšší úrovní aproximace dochází u mediánového filtru ke zhoršení schopnosti filtrace obrazu. Zároveň na tomto obrázku můžeme vidět, že aproximace má u různých komponent rozličný vliv na filtraci. Pro lepší představu jsem vytvořil graf na obrázku 5.4, kde je uvedena průměrná chyba na pixel obrazu v závisloti na komponentě. Tento graf částečně potvrzuje intuitivní myšlenku, že vyšší vliv bude mít na kvalitu filtrace aproximace komponent blíže k výstupu. Tato myšlenka však neplatí naprosto vždy, protože, jak můžeme z grafu vyčíst, například aproximace komponent 17 a 18 zhoršuje kvalitu filtrace více než aproximace poslední komponenty č. 19, tedy komponenty, která má k výstupu z řadicí sítě nejblíže. Pro lepší představu jsem vytvořil heatmapu pro tuto řadicí síť, která na obrázku 5.6 přehledně ukazuje, na jakých místech jsou komponenty na aproximaci málo citlivé (modrá barva) a naopak, na kterých místech jsou komponenty velmi citlivé (červená barva).

Dalším zajímavou věcí v tomto experimentu bylo porovnání dvou implementací řadicí sítě. Graf na obrázku 5.3 ukazuje, že vliv aproximace na komponenty druhé implementace řadicí sítě (M2) je v podstatě stejný jako u první implementace. Pokud však porovnáme graf na obrázku 5.4 s grafem 5.5, vidíme významný rozdíl. Pro potřeby této práce jsou však důležité následující informace. U prvního testovaného filtru jsem u všech aproximací přes všechny komponenty vypočítal průměrnou chybu na pixel obrazu 19,7856. U druhé testované implementace však tato hodnota byla 19,8381. Druhý mediánový filtr (M2) tedy

při aplikaci aproximace vykazuje větší chybovost. Důsledkem toho faktu tedy je, že jsem v dalších experimentech jako referenční implementaci využíval první testovanou implementaci(M1).



(a) Obrázek zašuměný 40 % šumem.

(b) Výsledek přesného filtrování.

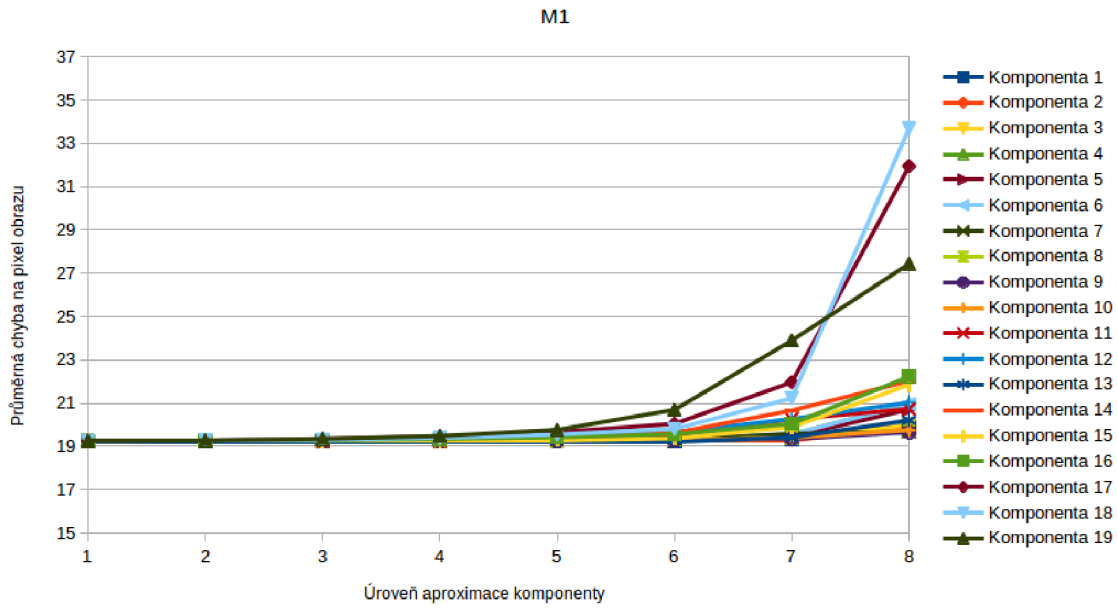
(c) Výsledek s aproximací úrovně 5. Aproximována komponenta č. 1.

(d) Výsledek s aproximací úrovně 5. Aproximována komponenta č. 16.

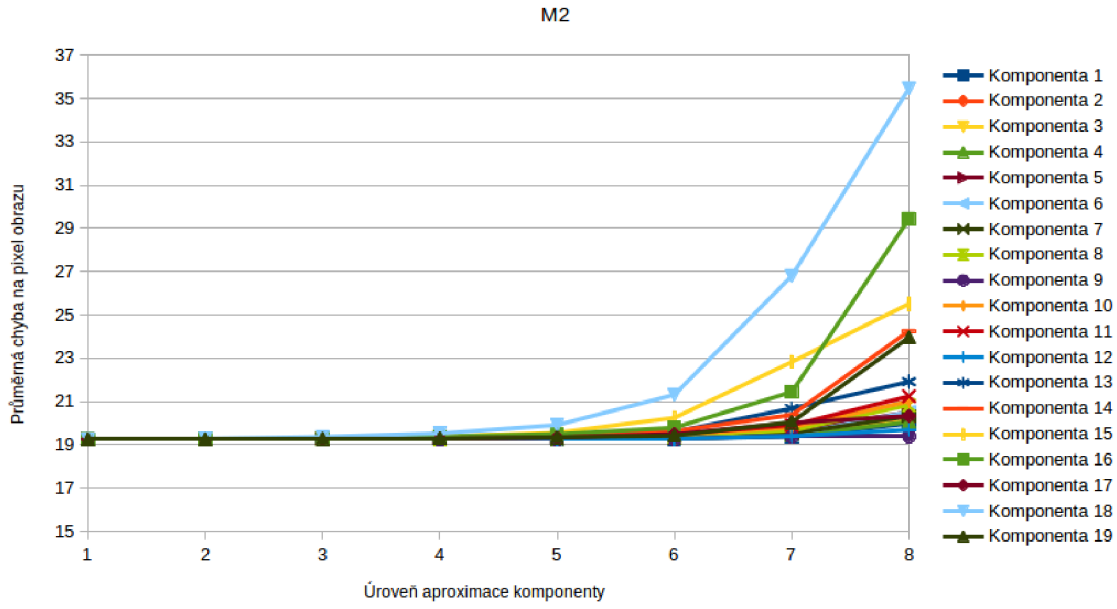
(e) Výsledek s aproximací úrovně 8. Aproximována komponenta č. 1.

(f) Výsledek s aproximací úrovně 8. Aproximována komponenta č. 16.

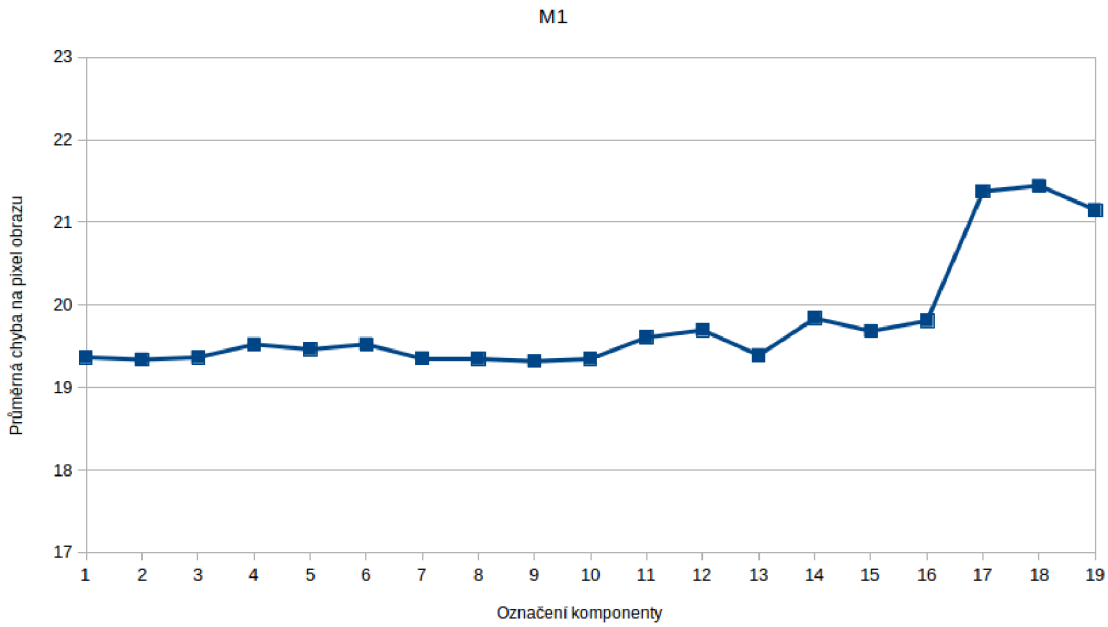
Obrázek 5.1: Porovnání výsledků filtrování při aproximaci pouze jedné komponenty (M1). Vyšší míra šumu zvolena z důvodu dobré vizualizace rozdílné kvality filtrace při aproximaci různých pozic ve filtru.



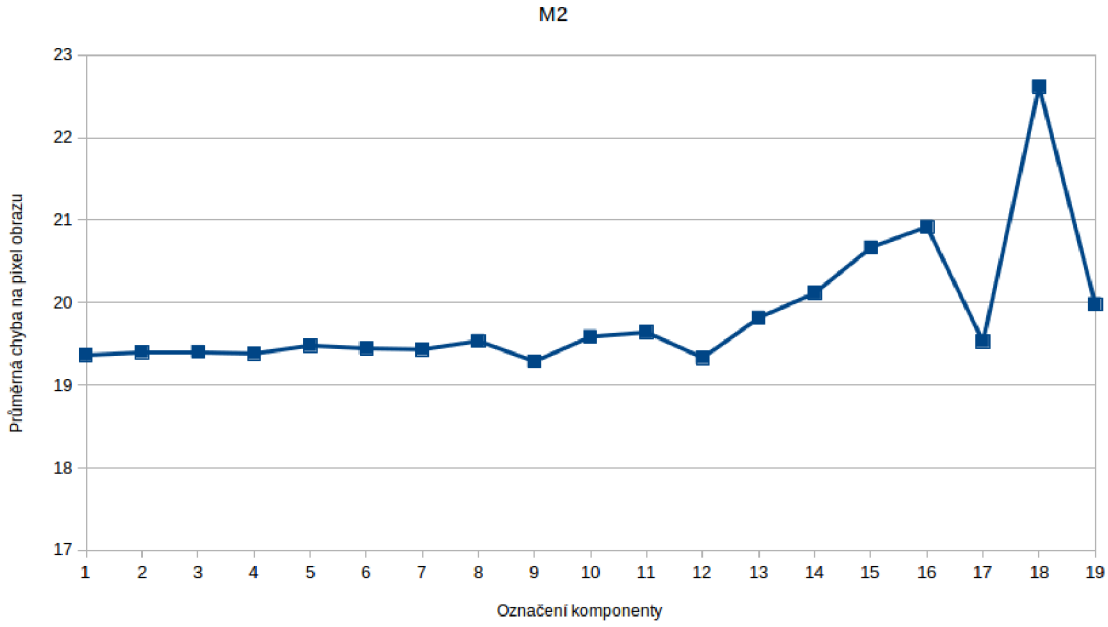
Obrázek 5.2: Citlivost pozic komponent na aproximaci. Průměrná chyba na pixel obrazu v závislosti na úrovni aproximace dané komponenty po profiltrování celého testovacího datasetu. Jedná se o první zvolenou implementaci mediánového filtru (M1).



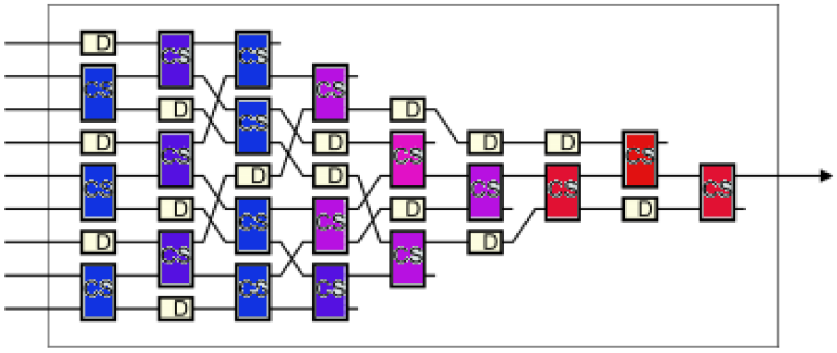
Obrázek 5.3: Citlivost jednotlivých pozic komponent na aproximaci. Průměrná chyba na pixel obrazu v závislosti na úrovni aproximace dané komponenty po profiltrování celého testovacího datasetu. Jedná se o druhou zvolenou implementaci mediánového filtru (M2).



Obrázek 5.4: Citlivost pozic komponent na aproximaci. Průměrná chyba na pixel obrazu v závislosti na pozici dané komponenty v řadicí síti po profiltrování celého testovacího datasetu. Jedná se o první zvolenou implementaci mediánového filtru (M1).



Obrázek 5.5: Citlivost pozic komponent na aproximaci. Průměrná chyba na pixel obrazu v závislosti na pozici dané komponenty v řadicí síti po profiltrování celého testovacího datasetu. Jedná se o druhou zvolenou implementaci mediánového filtru (M2).



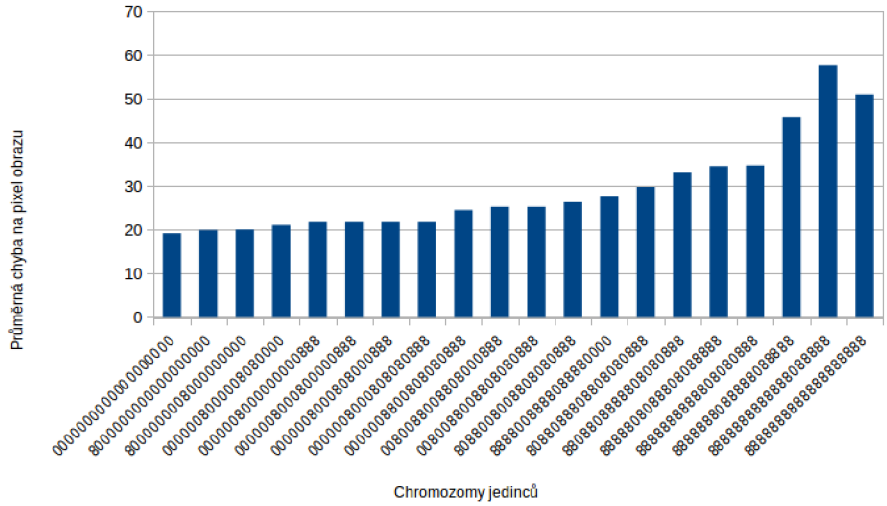
Obrázek 5.6: Heatmapa citlivosti na aproximaci u filtru zvoleného pro využití v dalších experimentech. Modrá – malá citlovost na aproximaci, červená – vysoká citlivost na aproximaci.

5.2 Experiment 2

V rámci druhého experimentu byl prohledán celý stavový prostor řešení, kdy se jednotlivé komponenty odpojují (tedy aproximují úrovní aproximace 8). Na obrázku 5.7 můžeme vidět postupné zhoršování kvality filtrace s nabývajícími požadavky na menší cenu, kterou má filtr zabírat. Vyjímkou je předposlední filtr, který má horší chybovost než filtr, který neprovádí vůbec nic. V podstatě se tedy dá říci, že šum v obraze ještě zhoršuje, než aby ho eliminoval. Na obrázku 5.7 je možné dále vidět, že v návrhu uvedeným způsobem vybraná řešení musí zákonitě obsahovat i neaproximovaný (přesný) filtr a také absolutně odpojený, tedy žádný filtr. Tyto filtry jsou však ve vyhodnocování ponechány pro možnost porovnání s nalezenými filtry.

Dalším zajímavým fenoménem, který však ze zde uvedeného grafu nelze dobře vyčíst, je, že u několika implementací filtrů se naprosto shodovala jejich schopnost filtrovat testovací dataset. Jednalo se o následující filtry[1] `0000008000000000888`, `0000008000800000888`, `0000008000808000888`, `0000008000808080888`, které měly shodně chybovost 21,8140 na jeden pixel obrazu a pak také o `0080088008808000888` a `0080088008808080888`, které mely chybovost 25,3347 na pixel obrazu. Tento fenomén lze však snadno vysvětlit. Podívejme se na obrázek 3.1. Například filtr `0000008000000000888` odpojuje komponenty číslo 7, 17, 18 a 19. Filtr se shodnou chybovostí `0000008000800000888` pak odpojuje komponenty číslo 7, 11, 17, 18 a 19. Odpojení komponenty 11 u tohoto způsobu aproximace tedy nehraje žádnou roli, protože neovlivňuje výstupní hodnotu. Výstupem z řadicí sítě je v podstatě horní výstup komponenty 16, která například komponentu 11 v předchozích úrovních řadicí sítě vůbec nevyužívá. Stejnou úvahu lze použít i ostatních filtrů, kde se hodnoty chybovosti shodují.

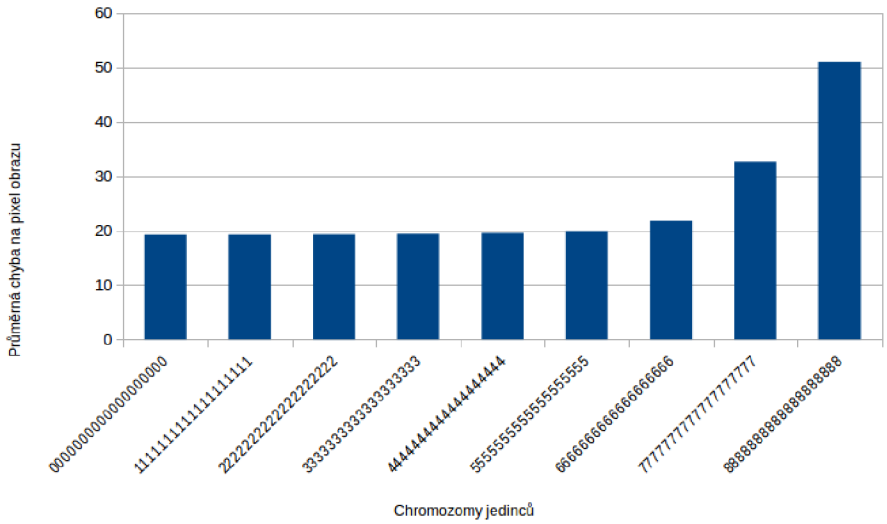
[1] V tomto kódování `0000008000000000888` znamená, že jsou aproximovány komponenty 7, 17, 18 a 19 úrovní aproximace 8. Ostatní komponenty nejsou aproximovány.



Obrázek 5.7: Hodnocení nejlepších filtrů nalezených hrubou silou pomocí odpojování komponent. Průměrná chyba po profiltrování celého testovacího datasetu na pixel obrazu v závisloti na chromozomu jedince. V grafu vidíme 20 různých povolených úrovní ceny.

5.3 Experiment 3

Ve výsledcích experimentu 3 můžeme vidět, že malá aproximace (provedená stejným způsobem) komponent zároveň nevnáší do filtrace příliš výraznou chybu. Tento experiment byl jednodušší, bude však zajímavé jeho výsledky porovnat s řešeními, které nalezne evoluční algoritmus v experimentu 5.



Obrázek 5.8: Hodnocení nejlepších filtrů nalezených hrubou silou pomocí aproximace všech komponent tak, že všechny mají stejnou úroveň aproximace. Průměrná chyba po profiltrování celého testovacího datasetu na pixel obrazu v závisloti na chromozomu jedince. V grafu vidíme 9 různých úrovní ceny.

5.4 Experiment 4

Při provádění 4. experimentu byly získávány průměrné hodnoty fitness po padesáti bězích v jednom nastavení parametrů evolučního algoritmu. Byl sledován vliv nastavení velikosti populace a pravděpodobnosti mutace. Následně jsem tyto získané hodnoty vynesl do tabulky, kterou jsem barevně označil podle získaných hodnot fitness. Nejnižší získanou hodnotou byla hodnota 16 706 844,98, která byla označena červenou barvou. K této hodnotě algoritmus došel v případě nastavení parametrů na 30 jedinců v populaci 40 % pravděpodobnosti mutace. Vyšší hodnoty fitness byly pak dále označeny oranžovou, žlutou až po zelenou. Výsledek můžeme vidět v tabulce 5.1. Barvy přirozeně přecházejí a někdy vytváří „shluky“. Nejvyšší hodnotu jsem zaznamenal u nastavení, kdy byla velikost populace stanovena na 70 jedinců a pravděpodobnost mutace na 50 %. Celkově se z tabulky zdá být nastavení 70 jedinců v populaci jako nejlepší zkoumané. Proto v závěrečném pátém experimentu bude ono nalezené nastavení využito.

P. mutace \ Velikost populace	30	40	50	60	70	80
40	-50.87	-42.46	-41.79	-10.06	11.93	-8.65
50	-14.25	-43.08	-13.88	-4.39	13.35	-11.96
60	7.23	-3.05	-20.06	-14.77	7.64	4.19
70	7.19	-1.56	6.86	12.76	3.2	12.19
80	1.39	3.15	10.29	0.01	1.97	-0.01
90	-4.04	-6.58	2.95	5.76	2.91	-5.25
100	2.85	8.22	-8.62	1.33	-6.53	-19.39

Tabulka 5.1: Výsledky průměrné fitness hodnoty po 50 bězích s nastavením 10 000 evaluací fitness na běh. Vyjádřeno relativně v parts per million (ppm) vůči hodnotě mediánu těchto hodnot.

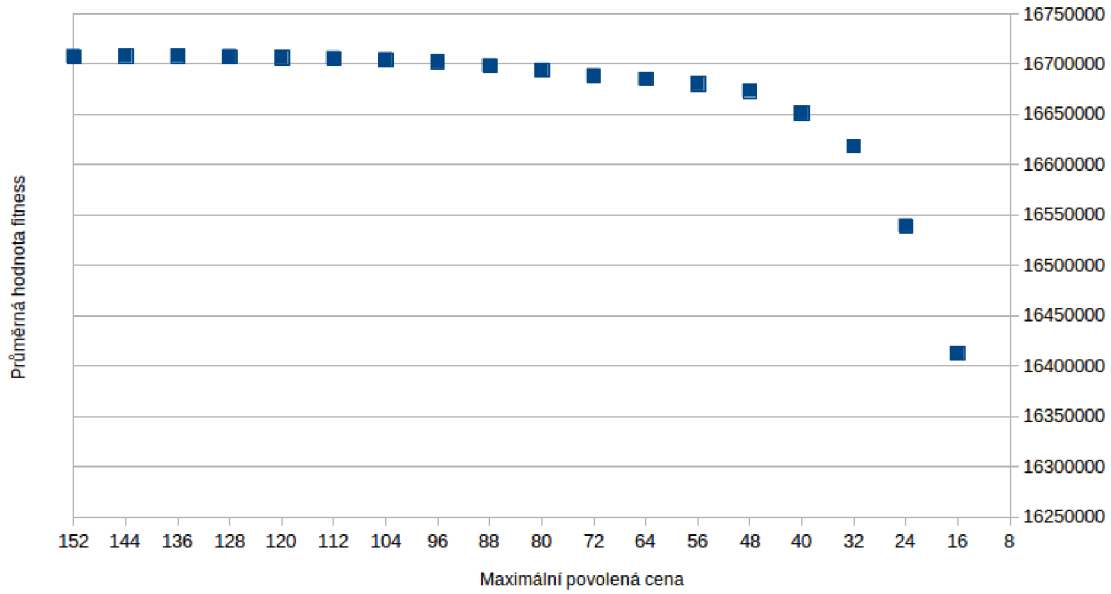
5.5 Experiment 5

Parametry evolučního algoritmu byly nastaveny na hodnoty získané předchozím experimentem – 70 jedinců v populaci a 50 % pravděpodobnost mutace. Dale jsem postupoval podle návrhu experimentu z kapitoly 3.8.5. Z vyhodnocování výsledků jsem vypustil všechny výstupy, které nesplňovaly požadavek na maximální povolenou cenu (menší cena však byla akceptována). Jednalo se konkrétně o nastavení, kdy se evoluční algoritmus snažil najít implementace, které mají maximální cenu 0 a 8. V těchto případech se ani jednou nepodařilo nalézt dostatečně „malé“ řešení. Z vyhodnocování jsem také vypustil 2 výsledky z nastavení maximální ceny na 16, kdy opět došlo k překročení daného limitu.

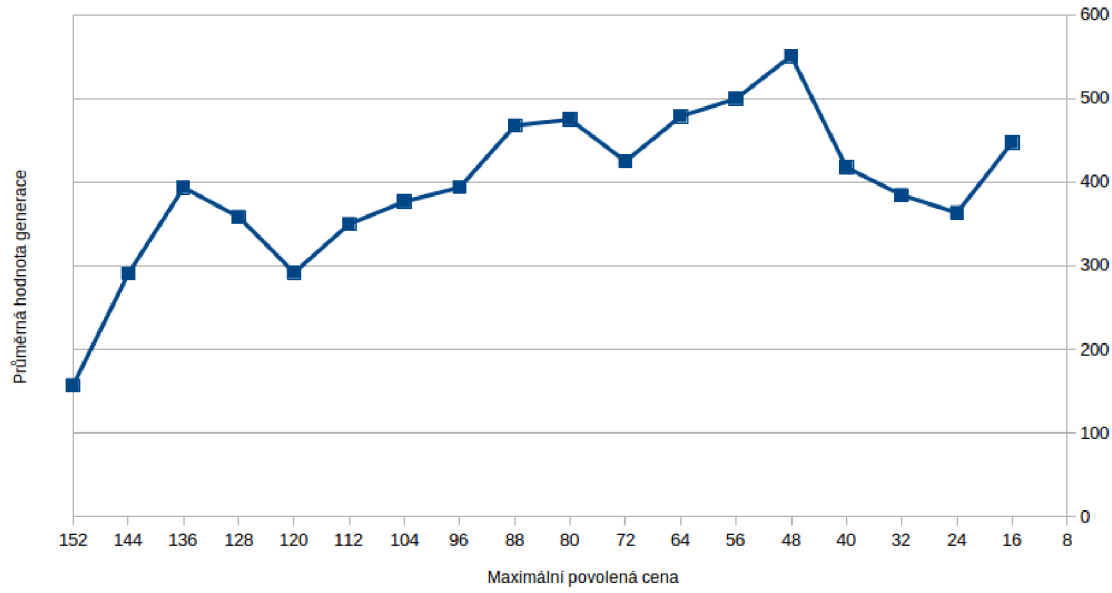
Na obrázku 5.9 vidíme, že s menší vyžadovanou cenou postupně klesá průměrná fitness hodnota nalezených filtrů. Tento fakt je předpokládatelný, zajímavé ale je, že u největších možných cen evoluce v zásadě nikdy nevyužila kompletně celou využitelnou cenu. Toto zjištění je zásadní a hlouběji se jím zabývám v kapitole 5.6. Při pohledu na graf na obrázku 5.10 pak můžeme vidět, že při menších omezeních na cenu evoluční algoritmus velmi rychle nalezl kvalitní řešení. Je třeba připomenout, že hodnota parametru počtu generací byla u evolučního algoritmu nastavena na 714, tak aby se počet evaluací při populaci 70 jedinců blížil 50 000.

Dalším zjištěním je, že průměrná kvalita filtrace se u filtrů nalezených při menších nárocích na cenu výrazněji nelišila od kvality filtrace nejlepšího nalezeného filtru. Při vyšším

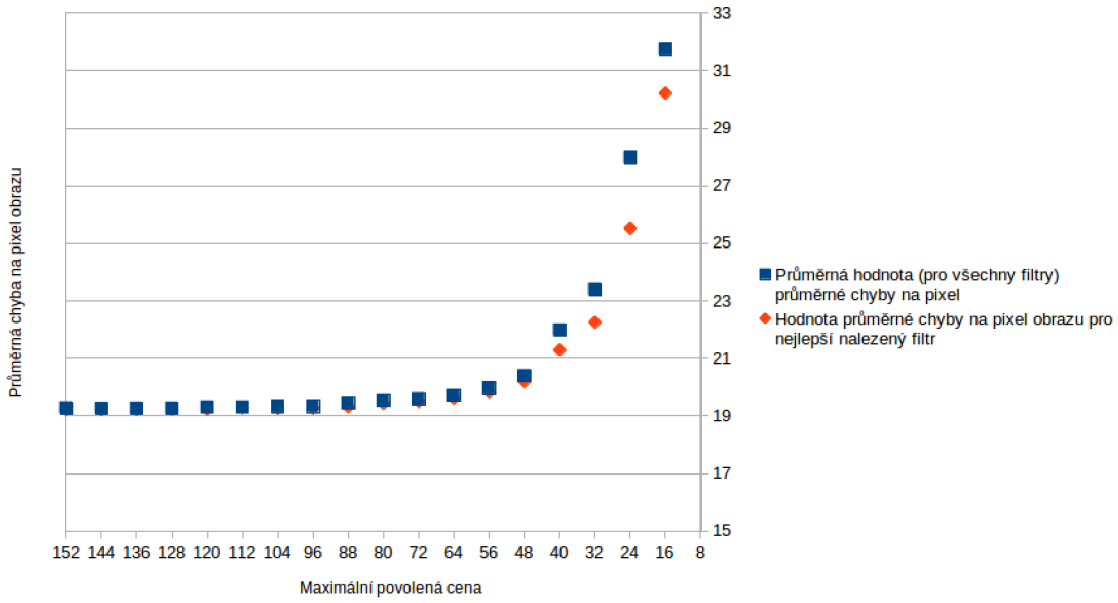
omezení ceny se však rozdíl začal projevovat. I to ilustrují grafy na obrázcích 5.9 a 5.10. Dále v tabulce 5.2 můžeme vidět postupné zhoršování kvality u jednotlivých nalezených filtrů v závislosti na vyžadované maximální ploše.



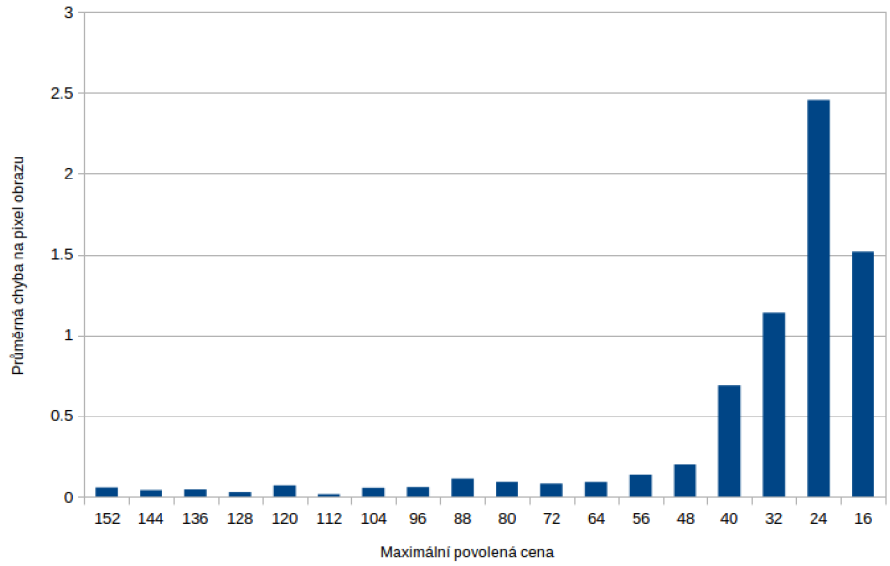
Obrázek 5.9: Průměrné fitness hodnoty všech získaných filtrů v závislosti na maximální povolené ceně implementace.



Obrázek 5.10: Průměrné hodnoty generací, ve kterých bylo nalezeno nejlepší řešení, v závislosti na maximální povolené ceně implementace.



Obrázek 5.11: Porovnání průměrné hodnoty chyby na pixel obrazu v závislosti na maximální ceně implementace. Průměr pro všechny filtry v porovnání s nejlepším nalezeným filtrem po provedení filtrovaní celého testovacího datasetu.



Obrázek 5.12: Rozdíly průměrné hodnoty chyby na pixel obrazu (všechny filtry vs. nejlepší filtr) v závislosti na maximální ceně implementace po provedení filtrovaní celého testovacího datasetu.

Maximální povolená cena	Běh algoritmu									
	1	2	3	4	5	6	7	8	9	10
152	19.38	19.22	19.27	19.27	19.27	19.27	19.27	19.27	19.27	19.27
144	19.27	19.22	19.27	19.27	19.22	19.27	19.27	19.27	19.27	19.27
136	19.27	19.27	19.27	19.22	19.27	19.27	19.27	19.27	19.27	19.27
128	19.27	19.27	19.24	19.27	19.27	19.27	19.27	19.27	19.27	19.27
120	19.28	19.59	19.28	19.28	19.28	19.28	19.28	19.29	19.28	19.24
112	19.3	19.3	19.29	19.3	19.3	19.3	19.29	19.3	19.3	19.39
104	19.33	19.33	19.4	19.32	19.35	19.31	19.34	19.27	19.32	19.32
96	19.28	19.38	19.32	19.31	19.35	19.36	19.32	19.37	19.31	19.37
88	19.32	19.36	19.47	19.55	19.35	19.5	19.36	19.47	19.41	19.56
80	19.45	19.49	19.53	19.44	19.54	19.57	19.47	19.55	19.55	19.72
72	19.58	19.56	19.52	19.78	19.54	19.56	19.64	19.71	19.51	19.51
64	19.75	19.68	19.75	19.62	19.71	19.62	19.75	19.7	19.75	19.76
56	20.23	19.88	19.83	20.03	19.95	19.92	20.07	19.96	19.98	19.83
48	20.22	20.2	20.25	20.42	20.3	20.2	20.42	20.23	20.35	21.36
40	23.12	21.38	22.19	22.18	21.94	21.72	21.85	21.82	21.29	22.29
32	25.13	23.35	23.16	23.21	24.76	22.92	22.25	23.04	22.92	23.19
24	25.88	26.52	27.04	27.66	32.52	26.81	27.42	25.52	30.65	29.74
16	33.04	31.35	31.26	31.08	34.07	30.22	31.51	31.37		

Tabulka 5.2: Tabulka chybovosti jednotlivých nalezených řešení. Je uvedena průměrná chyba na pixel obrazu po profiltrování celého testovacího datasetu.

5.6 Porovnání nalezených filtrů

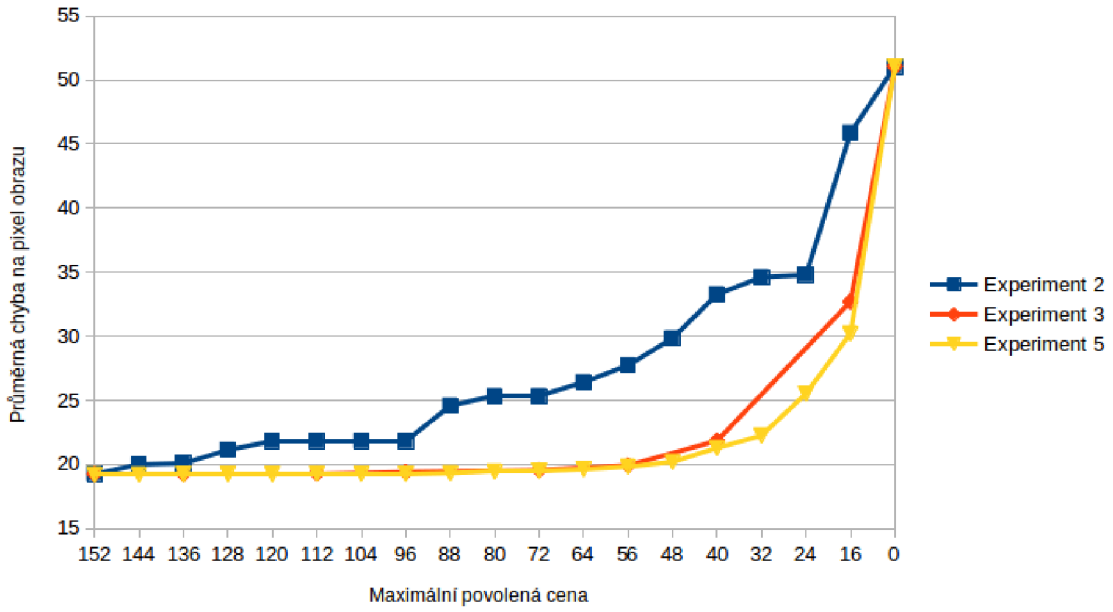
Ze srovnání všech experimentů, vidíme, že objektivně nejlepších výsledků při filtrování celého testovacího datasetu dosahuje evoluční algoritmus. Tento fakt ukazuje tabulka 5.3. Dobrých výsledků v tomto případě dosahuje i způsob aproximace z experimentu 3. Evoluce však ve všech případech byla schopná nalézt lepší řešení (není brána v potaz situace, kdy byla vyžadována cena 8 a 0). Na obrázku 5.13 pak můžeme vidět srovnání ve formě získaných Pareto front.

V rámci experimentu 3 se mi podařilo najít zajímavé implementace filtrů. Tento úsudek vychází z toho, že implementovat filtr pomocí součástky jednoho typu je jednodušší, než využívat různé implementace jednotlivých součástek. Nejzajímavější výsledky jsem však nalezl v experimentu 5. Evoluční algoritmus nalezl Pareto množinu, ze které vyčnívá několik implementací aproximovaných mediánových filtrů, které mají lepší schopnost filtrace (testováno na celém datasetu) než neaproximovaný mediánový filtr. Zároveň ale vyžadují menší implementační cenu než neaproximovaná verze filtru (která zabíra největší možnou cenu).

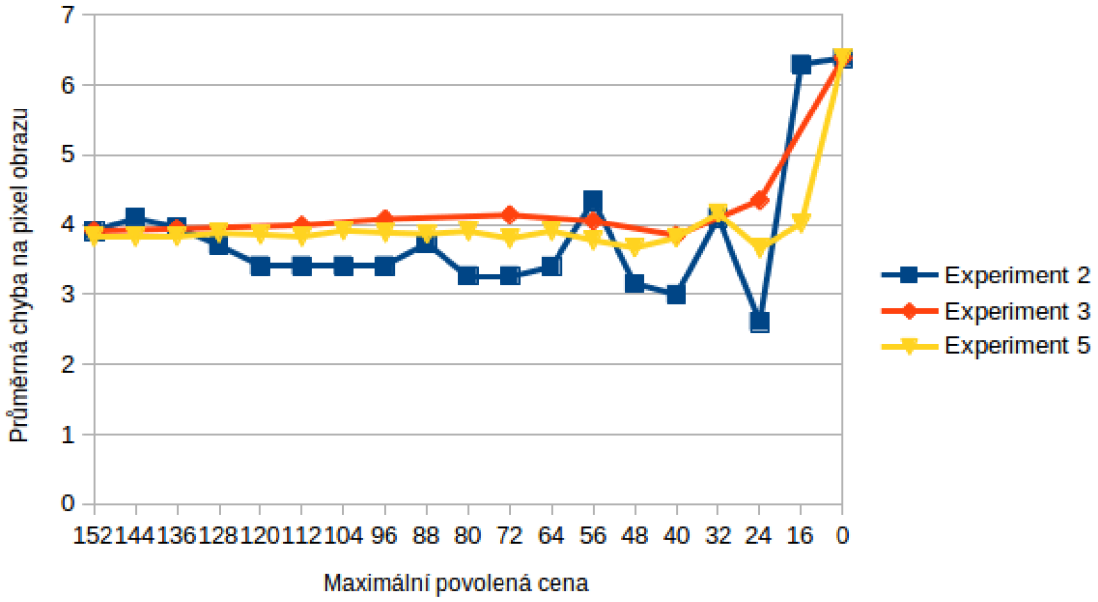
Dále jsem zjišťoval, jaké kvality filtrace dosahují nalezené filtry při filtrování pouze jednotlivých úrovní šumu. Na obrázku 5.13 vidíme porovnání implementací pro šum 5 % úrovně. V této situaci je vidět, že postup z experimentu 2 dosahuje častěji dobrých výsledků. Avšak při pohledu na graf na obrázku 5.14, kdy byl filtrován 10 %, se rozdíl začíná zmenšovat a na grafu 5.16 již evoluční algoritmus dominuje. Zdá se tedy, že evolučně nalezené filtry jsou použitelnější při filtraci vyšší úrovně šumu. Tonto však není jediné možné využití evolučně nalezených filtrů. Na obrázcích 5.17, 5.18 a 5.19 jsem srovnal některé výsledky filtrování jak konvečního mediánového filtru, tak filtrů nalezených v experimentech. Z experimentálně nalezených filtrů jsem vybral takové filtry, které zabírají zhruba polovinu nejvyšší možné ceny a také nejnižší vyhodnocovanou cenu. Jako v předchozím porovnání jsem zvolil 5 %, 10 % a 50 % úrovně šumu. Můžeme vidět, že evolučně nalezené filtry, narozdíl od filtrů nalezených hrubou silou, dokáží dobře pracovat i s malými nároky na cenu implementace. Jak u velkých nároků na cenu, tak i u případů, kdy byla vyžadována polovina maximální implementační ceny, dokáží z vizuálního hlediska kvalitně odfiltrovat šum.

Cena	Experiment 2	Experiment 3	Experiment 5
152	19.25	19.25	19.22
144	20		19.22
136	20.08	19.27	19.22
128	21.13		19.24
120	21.81		19.24
112	21.81	19.31	19.29
104	21.81		19.27
96	21.81	19.41	19.28
88	24.58		19.32
80	25.33		19.44
72	25.33	19.56	19.51
64	26.41		19.62
56	27.74	19.92	19.83
48	29.84		20.2
40	33.28	21.85	21.29
32	34.61		22.25
24	34.78		25.52
16	45.87	32.7	30.22
0	51.05	51.05	51.05

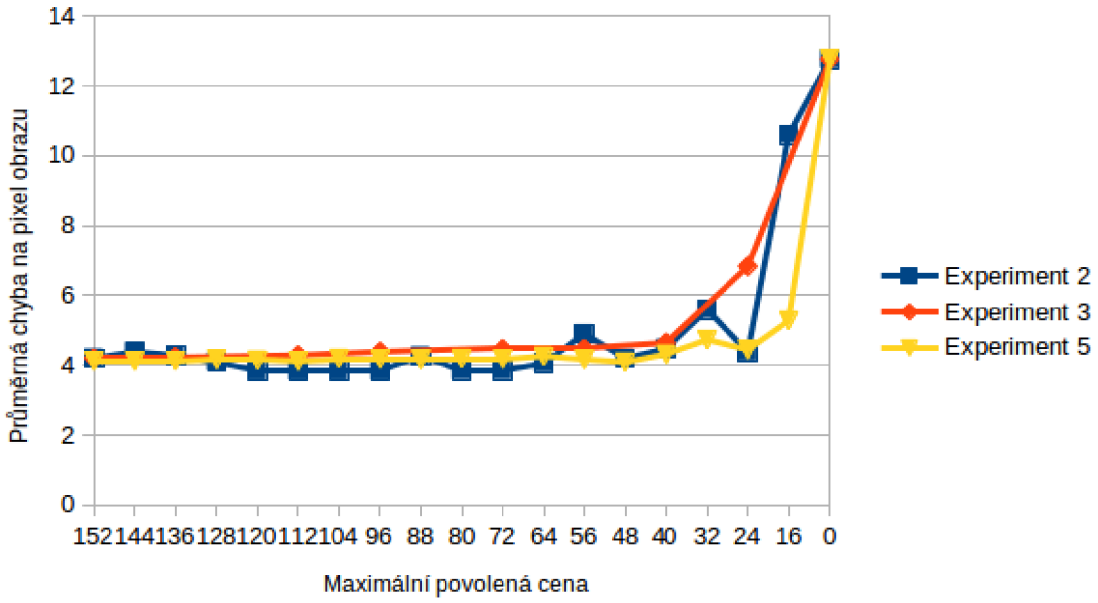
Tabulka 5.3: Tabulka chybovosti nejlepších implementací nalezených v jednotlivých experimentech. Jednotkou je průměrná chyba na pixel obrazu po provedení filtrování všech obrázků v testovacím datasetu. Modrá místa značí nemožnost získání hodnoty plynoucí z logiky konstrukce experimentu. Evoluční algoritmus nalézá ve všech případech nejlepší řešení.



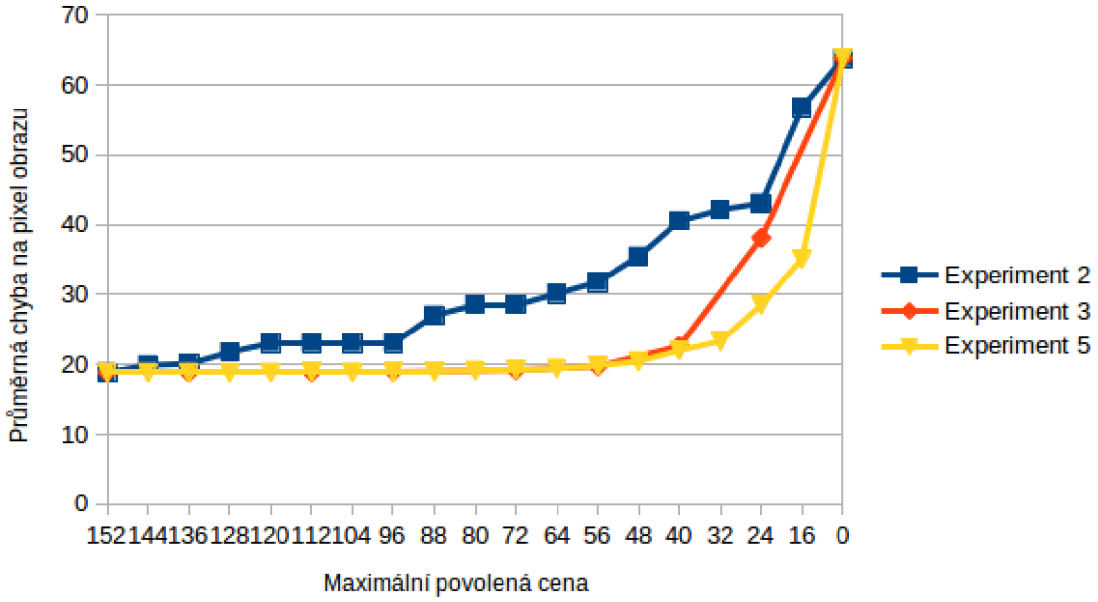
Obrázek 5.13: Chybovost nejlepších implementací nalezených v jednotlivých experimentech po profiltrování celého testovacího datasetu. Vidíme, že výsledky evolučního algoritmu v tomto případě vždy dominují ostatní nalezená řešení.



Obrázek 5.14: Chybovost nejlepších implementací nalezených v jednotlivých experimentech po profiltrování datasetu s 5 % šumem.



Obrázek 5.15: Chybovost nejlepších implementací nalezených v jednotlivých experimentech po profiltrování datasetu s 10 % šumem.



Obrázek 5.16: Chybovost nejlepších implementací nalezených v jednotlivých experimentech po profiltrování datasetu s 50 % šumem.

(a) Obrázek zašuměný 5 % šumem.

(b) Výsledek přesného filtrování.

(c) Hrubá síla z exp. 2 s maximální cenou 80.

(d) Evoluční algoritmus z exp. 5 s maximální cenou 80.

(e) Hrubá síla z exp. 2 s maximální cenou 16.

(f) Evoluční algoritmus z exp. 5 s maximální cenou 16.

Obrázek 5.17: Porovnání výsledků filtrování konvečního mediánového filtru a filtrů nalezených v jednotlivých experimentech na obraze zašuměném 5 % šumem sůl a pepř.

(a) Obrázek zašuměný 10 % šumem.

(b) Výsledek přesného filtrování.

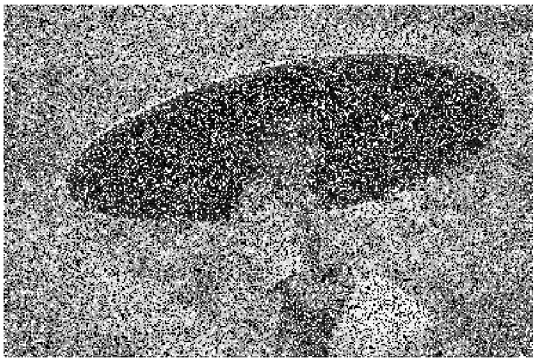
(c) Hrubá síla z exp. 2 s maximální cenou 80.

(d) Evoluční algoritmus z exp. 5 s maximální cenou 80.

(e) Hrubá síla z exp. 2 s maximální cenou 16.

(f) Evoluční algoritmus z exp. 5 s maximální cenou 16.

Obrázek 5.18: Porovnání výsledků filtrování konvečního mediánového filtru a filtrů nalezených v jednotlivých experimentech na obraze zašuměném 10 % šumem sůl a pepř.



(a) Obrázek zašuměný 50 % šumem.

(b) Výsledek přesného filtrování.

(c) Hrubá síla z exp. 2 s maximální cenou 80.

(d) Evoluční algoritmus z exp. 5 s maximální cenou 80.

(e) Hrubá síla z exp. 2 s maximální cenou 16.

(f) Evoluční algoritmus z exp. 5 s maximální cenou 16.

Obrázek 5.19: Porovnání výsledků filtrování konvečního mediánového filtru a filtrů nalezených v jednotlivých experimentech na obraze zašuměném 50 % šumem sůl a pepř.

Kapitola 6

Závěr

V rámci této diplomové práce byla shrnuta problematika aproximativních výpočtů s důrazem na funkcionální aproximaci. Dále byla nastíněna oblast filtrování obrazu, byly popsány některé druhy obrazových filtrů a typy šumu v obraze. Dalším okruhem byla problematika evolučních algoritmů. Byly popsány některé konkrétní algoritmy – genetické programování a kartezské genetické programování. U jednotlivých teoretických okruhů byly vysvětleny používané pojmy.

Dále se tato diplomová práce zabývala návrhem řešení problému aproximace mediánového filtru 3 x 3. Problém jsem formuloval, stanovil jeho vstupy, výstupy a kroky nutné k jeho řešení. Identifikoval jsem některé překážky, které s problémem diplomové práce vystávají, a navrhl jejich řešení. Analyzoval jsem možnosti prohledávání stavového prostoru řešení. Navrhl jsem kritické části evolučního algoritmu. Určil potřebné technologie a navrhl sadu experimentů. Dále jsem vytvořil implementace potřebných programů, které jsem v této práci popsal. Popsal jsem výsledky jednotlivých experimentů, poté shrnul celkové výsledky. Porovnal jsem filtry nalezené různými způsoby. Schopnost redukce šumu v obraze jsem u filtrů zhodnotil jak ze statistického, tak i z vizuálního hlediska. Výstupem této práce jsou nové implementace obrazových filtrů.

Tato práce mi pomohla hlouběji proniknout do problematiky aproximativního počítání, filtrování obrazu a evolučních algoritmů. Pokračováním této práce by mohla být evoluční aproximace dalších obrazových filtrů například mediánového filtru 5 x 5, nebo evolučně navržených filtrů ze zde citovaných článků. Dalším logickým krokem by také mohlo být navržení konkrétních aproximovaných komponent v hardwaru a ověření kvality zde nalezených filtrů v praxi a ne pouze simulací.

Literatura

[1] Bidlo, M.: Aplikované evoluční algoritmy, přednášky. 2018, [Online; navštíveno 3.1.2019].

[2] Gajda, Z.; Sekanina, L.: Reducing the Number of Transistors in Digital Circuits Using Gate-level Evolutionary Design. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-697-4, s. 245–252, doi:10.1145/1276958.1277010.
URL http://doi.acm.org/10.1145/1276958.1277010

[3] Gimel'farb, G.: Part 3: Image Processing, Moving Window Transform. [Online; navštíveno 2.1.2019].
URL https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-04.pdf

[4] Li, P.; Lilja, D. J.; Qian, W.; aj.: Computation on Stochastic Bit Streams Digital Image Processing Case Studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, ročník 22, č. 3, March 2014: s. 449–462, ISSN 1063-8210, doi:10.1109/TVLSI.2013.2247429.

[5] Mather, P. M.: *Computer Processing of Remotely-Sensed Images: An Introduction*. USA: John Wiley & Sons, Inc., 2004, ISBN 0470849185.

[6] Mrázek, V.: *Metodologie pro automatický návrh nízkopříkonových aproximativních obvodů*. Dizertační práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/PD.php?id=841

[7] Sekanina, L.: Introduction to Approximate Computing: Embedded Tutorial. In *19th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Institute of Electrical and Electronics Engineers, 2016, ISBN 978-1-5090-2467-4, s. 90–95, doi:10.1109/DDECS.2016.7482460.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11075

[8] Sekanina, L.; Vašíček, Z.; Mrázek, V.: Approximate Circuits in Low-Power Image and Video Processing: The Approximate Median Filter. *Radioengineering*, ročník 26, 09 2017: s. 623–632, doi:10.13164/re.2017.0623.

[9] Sekanina, L.: Biologií inspirované počítače, přednášky. 2018, [Online; navštíveno 3.1.2019].

[10] Sekanina, L and Harding, S. L. and Banzhaf, W. and Kowaliw, T.: *Image Processing and CGP*. Springer Berlin Heidelberg, 2011, ISBN 978-3-642-17310-3, s. 181–215,

doi:10.1007/978-3-642-17310-3_6.
URL https://doi.org/10.1007/978-3-642-17310-3_6

[11] Sekanina, L. and Vašíček, Z.: Approximate Computing and Approximate Circuits. AHS Tutorial. 2018, [Online; navštíveno 3.1.2019].
URL
http://www.fit.vutbr.cz/~sekanina/publ/ahs18/ahs18_tut_approx_comp_1.pdf

[12] Vašíček, Z.; Bidlo, M.; Sekanina, L.: Evolution of efficient real-time non-linear image filters for FPGAs. *Soft Computing*, ročník 17, 11 2013: s. 2163–2180, doi:10.1007/s00500-013-1040-8.

[13] Zemčík, P. and Beran, V. and Španěl, M.: Zpracování obrazu, přednášky. 2018, [Online; navštíveno 12.7.2019].

Příloha A

Nalezené filtry

Implementace filtrů nalezené v jednotlivých experimentech. Tabulky jsou uvedeny ve formátu chromozom filtru, cena implementace a kvalita filtrace v průměrné chybě na pixel obrazu.

Experiment 2

0000000000000000000	152	19.2491833722
8000000000000000000	144	19.9951616923
8000000008000000000	136	20.0847495468
0000008000008080000	128	21.1258479366
0000008000000000888	120	21.8140428049
0000008000800000888	112	21.8140428049
0000008000808000888	104	21.8140428049
0000008000808080888	96	21.8140428049
0000008800808080888	88	24.5772708752
0080088008808000888	80	25.3347060592
0080088008808080888	72	25.3347060592
8088008008808080888	64	26.4095249035
8888008888088880000	56	27.7360746596
8088088808808080888	48	29.8429078844
8808808888808080888	40	33.2759197094
8888808088808088888	32	34.6095314026
8888888888808080888	24	34.7767602143
8888888088888088888	16	45.8675572431
8888888888888888888	0	51.0507679409

Experiment 3

0000000000000000000	152	19.2491833722
1111111111111111111	133	19.2682813009
2222222222222222222	114	19.3074632433
3333333333333333333	95	19.4084948222
4444444444444444444	76	19.55683379
5555555555555555555	57	19.9184242531
6666666666666666666	38	21.8479610584
7777777777777777777	19	32.7011590746
8888888888888888888	0	51.0507679409

Experiment 5

0001105000105000000	152	19.2186026397
0001105000105000000	144	19.2186026397
0011105100105010000	136	19.2208247432
0112335031104000000	128	19.2383546335
1142445131105000000	120	19.2384683962
3424426144105000000	112	19.2906612538
4444445344205010000	104	19.2713576705
5424524255244404000	96	19.2773074115
6534525366145414000	88	19.3244843942
5555546244455525010	80	19.4400240015
6546644455244414444	72	19.5091207999
6556645446455515444	64	19.6184375622
6766745467445544444	56	19.8293889646
7766766577455545444	48	20.1961901629
7776767577555555655	40	21.2898411786
7776777677666657655	32	22.2548574094
7867887787757565776	24	25.5230491356
8878777788778767766	16	30.2213880751

Příloha B

Obsah přiloženého CD

- *projekt.pdf* – tato zpráva
- *tex/* – složka obsahující zdrojový tvar této zprávy
- *README* – soubor s popisem CD
- *noise_datasets/* – složka s testovacím datasetem poškozených obrázků
- *original/* – složka s nepoškozenými obrázky
- *experimentx/* – složka se soubory využitými v experimentu x a výstupními soubory experimentu x, může obsahovat soubry:
  - **.csv* – výstupní soubor s daty získanými během experimentu
  - *makefile*
  - **.cpp* – soubor spouštěný při provádění experimentu
  - **.h* – knihovna funkcí a parametrů pro soubor spouštěný při provádění experimentů
  - **.py* – skript provádějící experiment, popřípadě skript pro zpracování výsledků experimentu
  - **.png* – obrázky využívané v experimentu
  - *popisExperimentu.txt* – popis spuštění experimentu a dat získaných z experimentu
  - případné další potřebné spustitelné soubory