VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH VYUŽÍVAJÍCÍ BOOLEOVSKÉ SÍTĚ

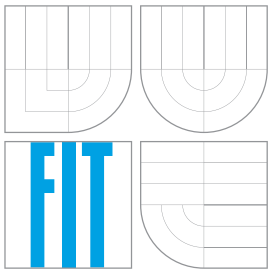
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL MRNUŠTÍK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH VYUŽÍVAJÍCÍ BOOLEOVSKÉ SÍTĚ

EVOLUTIONARY DESIGN USING RANDOM BOOLEAN NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL MRNUŠTÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2010

Abstrakt

Tato diplomová práce představuje možnosti využití booleovských sítí jako vývojového modelu v evolučním návrhu. Jsou zde popsány reprezentace booleovských sítí vhodné pro evoluční návrh včetně genetických operátorů. Booleovské sítě jsou použity jako vývojový model pro vývoj kombinačních obvodů a řadicích sítí. Dále je uvedena jedna z možných reprezentací aplikovatelná pro návrh obrazových filtrů. Navržené metody jsou experimentálně ověřeny a je navrženo jejich potenciální vylepšení a směr dalšího výzkumu.

Klíčová slova

Evoluční algoritmus, development, booleovská síť, kombinační obvod, řadicí síť, obrazový filtr.

Abstract

This master's thesis introduces the Random Boolean Networks as a developmental model in the evolutionary design. The representation of the Random Boolean Networks is described. This representation is combined with an evolutionary algorithm. The genetic operators are described too. The Random Boolean Networks are used as the developmental model for the evolutionary design of the combinational circuits and the sorting networks. Moreover a representation of the Random Boolean Networks for the design of image filters is introduced. The proposed methods are evaluated in different case-studies. The results of the experiments are discussed together with the potential improvements and topics of the next research.

Keywords

Evolutionary design, development, Random Boolean Network, combinational circuit, sorting network, image filter.

Citace

Michal Mrnuštík: Evoluční návrh využívající booleovské sítě, diplomová práce, Brno, FIT VUT v Brně, 2010

Evoluční návrh využívající booleovské sítě

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Mrnuštík
23. května 2010

Poděkování

Především bych chtěl poděkovat vedoucímu práce Ing. Michalu Bidlovi, Ph.D. za kvalitní vedení a podporu.

Obsah

Kapitola 1

Úvod

V každém oboru existují zažité postupy a metody. Ty jsou odzkoušeny léty praxe a fungují. Výsledky, kterých je dosaženo, jsou však do určité míry omezené. Evoluční návrh umožňuje v určitých případech nalezení lepších řešení, než jaké poskytují zažité postupy. Také nevyžaduje tolik znalostí v daném oboru jako konvenční návrhové metody a umožňuje částečnou automatizaci.

Cílem této diplomové práce je využití booleovské sítě jako vývojového modelu v evolučním návrhu. Konkrétně se zabývá aplikací booleovských sítí jako konstruktoru kombinačních obvodů a dále jejich využitím při filtrování obrazu.

Základem evolučního návrhu jsou evoluční algoritmy. Ty jsou založeny na simulaci přirozeného výběru, který probíhá v přírodě (viz [4]). Schopnost přežít je vyjádřena schopností řešit určitý problém. Evolučním algoritmům a jejich rozšířením o vývojové modely se věnuje kapitola 2. Problematika evolučních algoritmů je velmi rozsáhlá, proto jsou popsány jen základní principy a informace nutné k pochopení dalších kapitol.

V kapitole 3 jsou popsány booleovské sítě, jejich struktura a možnosti. Původní model sloužící jako genetická regulační síť byl v kapitole 4 upraven, aby mohl sloužit jako vývojový model při evolučním návrhu. Proto byla navržena reprezentace booleovské sítě a genetické operátory aplikovatelné na tuto reprezentaci.

Pomocí evolučního návrhu lze navrhovat také hardware a software a existuje mnoho metod jak toho dosáhnout. V kapitole 5 je uvedeno jak za použití booleovských sítí navrhovat hardware. Konkrétně se jedná o kombinační obvody na úrovni hradel. Uvedená metoda navazuje na postup popsaný v [3], kde se k návrhu kombinačních obvodů využívají celulární automaty. V této kapitole jsou také popsány výsledky experimentů, jež měly za cíl navrhnout různé typy kombinačních obvodů.

Kapitola 6 se zabývá návrhem řadicích sítí. Koncept z kapitoly 5 je upraven, aby místo hradel mohly být použity komparátory a také musel být přizpůsoben struktuře řadicí sítě. Kapitola 7 se zabývá návrhem filtru. Model booleovských sítí popsaný v kapitole 3 je upraven tak, aby odpovídal reprezentaci obrazu a operacím, které s ním má smysl provádět. Je navrženo několik typů filtrů, které jsou otestovány a výsledky jsou srovnány s mediánovým filtrem.

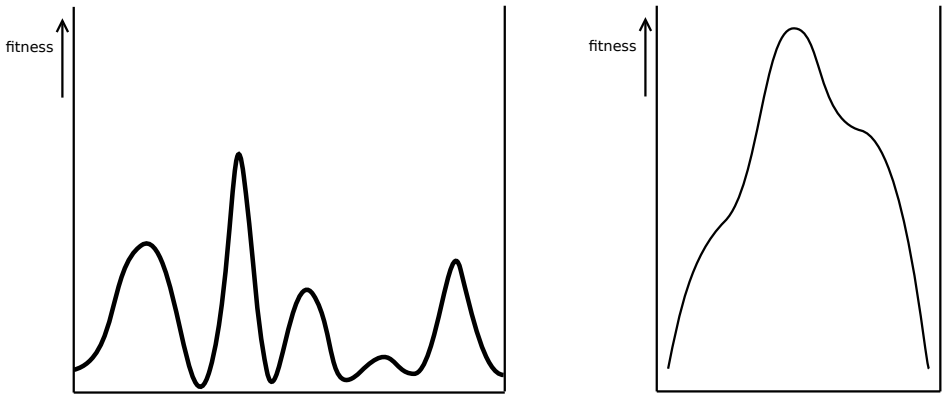
Na závěr v kapitole 8 je uvedeno shrnutí navržených metod, jejich celkové zhodnocení a návrh možností dalšího výzkumu. Diplomová práce navazuje na stejnojmenný semestrální projekt vypracovaný v zimním semestru akademického roku 2009/2010. Výsledky tohoto projektu jsou využity v kapitolách 2, 3, 4 a 5.

Kapitola 2

Evoluční algoritmy

Příroda a její zákony se ukázaly jako dobrá inspirace při vývoji výpočetních metod. Základy pro evoluční algoritmy položil Charles Darwin svou teorií přirozeného výběru. Ta je popsána v [4]. Druhy obývající naši planetu se již po miliony let vyvíjely z jednobuněčných organismů. Přežívají ty, které se dokázaly přizpůsobit okolním podmínkám lépe než ostatní.

Evoluční algoritmy se používají k řešení složitých optimalizačních problémů, kde neexistuje jiný postup nebo je časově příliš náročný. Pracují na podobném principu jako evoluce v přírodě. Místo jednotlivých organismů máme jedince představující řešení problému. Jedinec bývá definován svou vnitřní reprezentací (tzv. genomem) a hodnotou fitness, která představuje číselně vyjádřenou kvalitu řešení, které jedinec představuje. Genom mívá různou reprezentaci. Nejčastěji bývá genom reprezentován binárně nebo vektorem reálných či přirozených čísel, ale lze použít cokoli, co bude vhodné (znaky, stromy, atd.). Fitness funkce je klíčovým bodem při návrhu evolučního algoritmu. Měla by být výpočetně co nejméně náročná. Dále musí platit, že čím vyšší (nebo nižší, pokud hledáme minimum) je hodnota fitness, tím kvalitnější řešení jedinec poskytuje. Také je vhodné, aby prostor, který prohledáváme, neobsahoval mnoho lokálních maxim, ale postupně stoupal ke globálnímu maximu (viz obrázek 2.1).



Obrázek 2.1: Vlevo nevhodné a vpravo vhodné rozložení fitness v prohledávaném prostoru

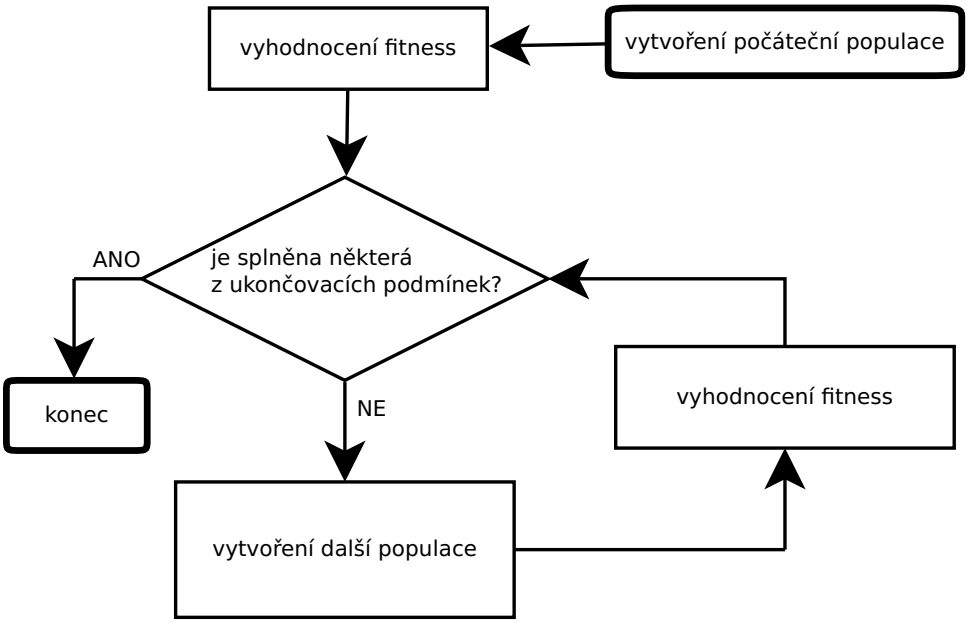
Podle [12] rozlišujeme evoluční algoritmy především na genetické algoritmy, evoluční strategie a genetické programování.

- Genetické algoritmy podle [8] mají v základní podobě pouze binárně kódované jedince.

Mutace i křížení se projevuje na úrovni jednotlivých bitů.

- Evoluční strategie (viz [16] a [17]) reprezentuje jedince jako vektor reálných čísel. Pro křížení a mutaci lze použít jak metod používaných pro genetické algoritmy, tak různých vektorových operací.
- Genetické programování (viz [11]) je vytváření algoritmů pomocí evolučního procesu.

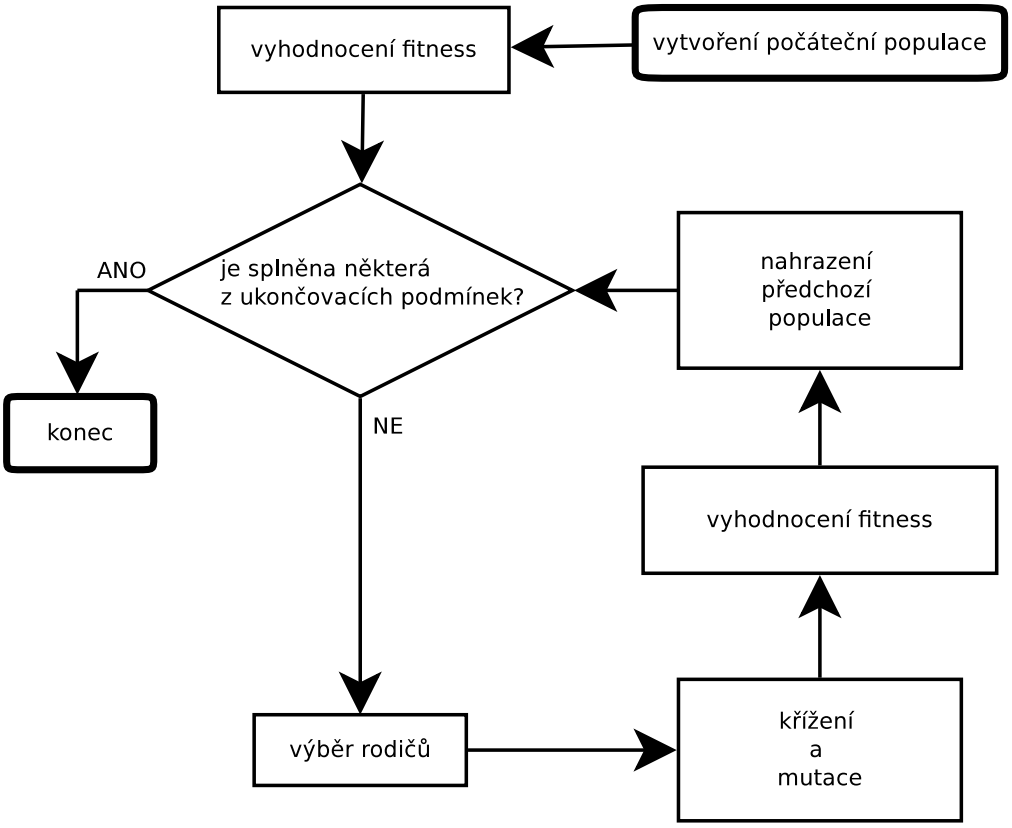
Na obrázku 2.2 je schéma obecného evolučního algoritmu. Jednotlivé varianty, které byly uvedeny výše se liší v tom, jakým způsobem získávají další populaci z té stávající. K návrhu metod v této práci byl vybrán genetický algoritmus, kterému se budeme věnovat dále.



Obrázek 2.2: Schéma obecného evolučního algoritmu podle [2]

2.1 Genetický algoritmus

Na obrázku 2.3 jsou jednotlivé fáze genetického algoritmu. Na začátku je náhodně vygenerována populace jedinců. Je vyhodnocena jejich fitness. Následně se kontroluje, zda nebyla splněna některá ukončovací podmínka (např. dosažení určité hodnoty fitness, maximálního počtu generací atd.). Pak jsou vybráni rodiče následující populace (viz 2.1.1). Křížením a mutací z nich získáme potomky a vyhodnotíme jejich fitness. Výběrem z rodičů a potomků nahradíme stávající populaci. Tak algoritmus pokračuje, dokud není splněna ukončovací podmínka.



Obrázek 2.3: Schéma genetického algoritmu

Ve zbytku této podkapitoly jsou příklady postupů pro jednotlivé kroky evolučního algoritmu. Kterýkoli z nich lze nahradit jiným (vhodnějším) postupem pro konkrétní problém.

2.1.1 Výběr rodičů

Existují různé postupy jak vybrat jedince, kteří se stanou rodiči následující populace. Výběr probíhá tak, aby jedinci s vyšší fitness měli větší šanci stát se rodiči, ale určitou roli hraje i náhoda. Výběr probíhá tolikrát, kolik potřebujeme rodičů (jedinec může být vybrán i víckrát). Jedním z algoritmů, které lze použít k výběru rodičů, je proporcionální výběr (tzv. vážená ruleta), který je popsán algoritmem 2.4. Další možností je turnajový výběr popsaný algoritmem 2.5. Na vybrané rodiče jsou aplikovány operátory křížení a mutace.

2.1.2 Křížení

Křížením se kombinují vlastnosti dvou jedinců (v některých případech lze použít jedinců i více, ale většinou k tomu není důvod). Cílem je vznik jedince lepšího než kterýkoli z rodičů. Postupy mohou být velmi rozmanité a lze je různě kombinovat. V některých případech není pro použití křížení důvod, a tak se vynechává.

Mezi nejpoužívanější patří jednobodové křížení, které je znázorněno na obrázku 2.6 a popsáno v algoritmu 2.7.

Existují i varianty, kde se generuje více bodů křížení. Například u dvou bodů vzniknou tři části A,B a C. Potomci vzniknou výměnou částí B.

- Spočítej součet hodnoty fitness pro všechny jedince v populaci S:

$$S = \sum_{j=1}^{P} fitness(i).$$

  P je velikost populace a j ukazuje na jednotlivé jedince v populaci.

- Generuj náhodné číslo $x \in \langle 0, 1 \rangle$.

- Opakuj postupně pro jedince j od 1 do P dokud $x > 0$:

$$x = x - \frac{fitness(j)}{S}$$

  Jedinec u kterého je dosaženo $x \leq 0$ je vybrán jako rodič.
  Funkce $fitness(j)$ značí hodnotu fitness funkce jedince j.

Algoritmus 2.4: Proporcionální výběr (vážená ruleta)

- Náhodně vyber x jedinců z populace.
- Z těchto x jedinců vyber toho s nejlepší fitness jako rodiče.

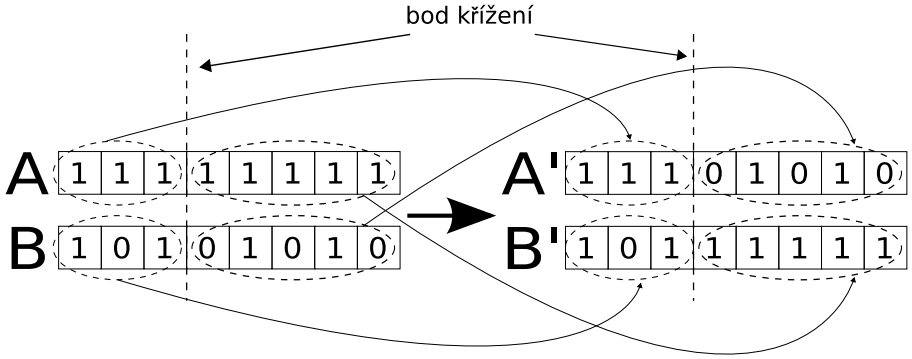
Algoritmus 2.5: Turnajový výběr o základu x

Křížení by mělo být navrženo tak, aby nerozbíjelo ty části genomu, které zapříčiňují vysokou fitness jedince, ale aby vhodně kombinovalo tyto části z obou rodičů. Pokud máme například binárně kódovaný genom, ve kterém je zahrnuto více částí (dejme tomu binárně reprezentovaný vektor přirozených čísel), bývá většinou vhodnější volit body křížení na hranici těchto částí. Potomek tak bude opravdu jen kombinací rodičů a nebudou se v něm objevovat prvky, které žádný z rodičů neobsahuje. Toto křížení lze použít i na genom reprezentovaný vektorem reálných čísel, jen místo bitů jsou složky vektoru. Další použitelnou metodou je křížení průměrem (viz [12]). Ze složek rodičovských vektorů r_1 a r_2 vznikne jediný potomek s odpovídající složkou $p = \frac{r_1+r_2}{2}$. Odlišný přístup je křížení lineární kombinací rodičů. Pro každou složku r_1 a r_2 rodičovských vektorů a číslo x v intervalu $\langle 0, 1 \rangle$ jsou vypočítány složky potomků:

$$p_1 = xr_1 + (1 - x)r_2 \quad (2.1)$$

$$p_2 = (x - 1)r_1 + xr_2 \quad (2.2)$$

Křížení nemusí probíhat vždy, ale může být dána pravděpodobnost s jakou ke křížení dojde.



Obrázek 2.6: Jednobodové křížení

- Vyber dva rodiče A, B.
- Náhodně zvol bod křížení.
- A' vznikne z té části A, která leží před bodem křížení a z části B ležící za bodem křížení.
- B' vznikne z té části B, která leží před bodem křížení a z části A ležící za bodem křížení.

Algoritmus 2.7: Jednobodové křížení

2.1.3 Mutace

V průběhu evolučního algoritmu je vhodné, když se v populaci objeví prvky, kterých nelze dosáhnout křížením. To zajišťuje právě mutace. Algoritmus 2.8 popisuje základní postup mutace pro binární genom. U reálných nebo celočíselných genomů se neprovádí negace, ale například součet s předem stanoveným nebo náhodně generovaným číslem v určitém rozsahu.

- Pro každý bit b v genomu:
  - Náhodně generuj $x \in \langle 0, 1 \rangle$.
  - Pokud $x \leq p_m$ neguj b.

Algoritmus 2.8: Mutace binárního genomu

U mutace je velmi důležitá velikost pravděpodobnosti p_m s jakou k mutaci dojde. Příliš nízká hodnota způsobí, že se mutace neprojeví a do evoluce nezasáhne. Přehnaně vysoká pravděpodobnost změní jedince tak, že by se od původního lišil stejně jako náhodně generovaný.

2.1.4 Výběr jedinců do následující populace

Poté, co máme k dispozici potomky vzniklé křížením a mutací, je třeba vybrat, kteří jedinci se budou účastnit dalšího průběhu evoluce. Používají se různé kombinace předchozí populace a potomků. Rodiče a potomky lze spojit do jedné populace a z ní potom vybrat potřebný počet jedinců pomocí postupů popsaných v podkapitole 2.1.1. Také lze vybrat ty nejlepší (tzv. elitismus), nebo celou předchozí populaci nahradit potomky.

2.2 Vývojové modely v evolučním návrhu

Evoluční návrh je použití evolučních algoritmů k činnosti, kterou bychom označili jako tvůrčí, pokud by ji prováděl člověk. Může se jednat o návrh strojních součástí, staveb, hardwaru, analogových obvodů atd. Po výsledku požadujeme, aby vyhovoval požadavkům lépe, nebo aby návrh trval kratší dobu, než kdyby ho prováděl odborník v dané oblasti. Výhodou je, že evoluční návrh může být značně inovativní, protože bere do úvahy i řešení, kterých nelze dosáhnout konvenčním postupem. Můžeme tak objevit věci, na které by člověk nikdy nepřišel (například [21]).

Pomocí evolučních algoritmů, tak jak jsou popsány v předchozí kapitole, a vhodného zakódování lze poměrně efektivně vyvíjet jednoduché struktury, jako například kombinační obvody (například [22]). Pokud bychom však chtěli vytvářet obvody složitější, tak se zvětší prostor, který musí evoluční algoritmus prohledávat. To je způsobeno tím, že větší a složitější obvod sestává z více hradel a tím se zvětší velikost genomu, který má tento obvod reprezentovat. Další komplikací je, že počet kombinací, které musíme testovat, roste exponenciálně s počtem vstupů obvodu. Hovoříme o tzv. problému škálovatelnosti.

Tento problém lze řešit pomocí vývojových modelů (tzv. development). Stejně jako evoluční algoritmy i vývojové modely našly svou inspiraci v přírodě. V tomto případě však ve vývoji vícebuněčných organismů. Vlastnosti a schopnosti organismů jsou dány molekulou DNA. Z jediné buňky vznikne mnohobuněčný organismus, například člověk. Jeho vývoj je dán právě DNA a prostředím. Nabízí se tedy možnost nehledat pomocí evolučního algoritmu přímo řešení, ale nějaký předpis (např. algoritmus), pomocí kterého řešení získáme. Vývojové modely v evolučním návrhu představují netriviální způsob mapování mezi genotypy a fenotypy.

Algoritmus 2.9 převzatý z [1] je ukázkou genetického algoritmu s developmentem. Oproti klasickému evolučnímu algoritmu obsahuje aplikaci vývojového modelu na jedince. Teprve z výsledku této operace se počítá fitness funkce, jejíž hodnota je následně přiřazena jedinci.

2.2.1 Mapování genotypu na fenotyp

Příroda vytvořila DNA jako reprezentaci genotypu. Postup, kterým DNA určuje vývoj organismu (fenotypu), ještě není úplně prozkoumán. Při použití vývojového modelu je tedy třeba navrhnout reprezentaci genotypu, genetické operátory (křížení a mutace, viz podkapitoly 2.1.2 a 2.1.3) a postup jakým se z genotypu stane fenotyp. Není nutné, aby byl tak složitý a univerzální jako ten přírodní. Stačí, když přinese výhody oproti přímému mapování. Jednou z takových výhod může být, že nevyvíjíme obvod určité velikosti, ale najdeme postup pro návrh libovolně velkých obvodů. To bylo provedeno například pro řadicí sítě (viz [18]). Byla použita malá řadicí síť a pomocí evolučního algoritmu byl nalezen předpis, který ji postupně upravoval až na požadovanou velikost.

1. Čas $t = 0$.
2. Náhodně generuj jedince v populaci $P(t)$.
3. Aplikuj development na každého jedince v $P(t)$.
4. Vypočti fitness takto získaných řešení.
5. Přiřaď fitness jedincům.
6. Dokud není splněna ukončovací podmínka opakuj:
   - Vyber rodiče z $P(t)$.
   - Křížením a mutací získej potomky $O(t)$.
   - Aplikuj development každého jedince v $O(t)$.
   - Vypočti fitness takto získaných řešení.
   - Přiřaď fitness jedincům.
   - Z $O(t)$ a $P(t)$ vyber populaci $P(t + 1)$.
   - Aktualizuj čas $t = t + 1$.

Algoritmus 2.9: Genetický algoritmus s developmentem

V [6] je přístup k vývojovým modelům rozdělen na explicitní a implicitní. Explicitní přístup je založen na použití embrya. Embryo je zárodek s určitou částí požadované funkčnosti. Evolučním algoritmem hledáme předpis k jeho modifikaci až do dosažení požadované funkčnosti. Tento přístup je využit v již zmiňovaném článku [18].

Implicitní přístup používá gramatiky a přepisovací pravidla podobně jak jsou používány ve formálních jazycích. Máme nějaký počáteční symbol a hledána jsou přepisovací pravidla. Do této skupiny patří i použití Lindenmayerových systémů (viz [13]). Ty se od běžných gramatik liší v tom, že je aplikováno více přepisovacích pravidel paralelně v jednom kroku. Použito je to například v [7].

2.3 Implementace genetického algoritmu

Implementace se drží schématu z obrázku 2.3. Náhodně se vygenerují booleovské sítě (viz kapitola 4), provede se jejich vývoj a výsledek je ohodnocen. To je popsáno v kapitolách zabývajících se konkrétními aplikacemi. Počet generovaných sítí závisí na zvolené velikosti počáteční populace. Jedince je třeba ohodnotit pouze jednou, protože podmínky vyhodnocení fitness funkce se v průběhu algoritmu nemění.

K výběru rodičů je použit turnaj (viz algoritmus 2.5). Náhodně jsou vybráni dva jedinci a ten lepší z nich se stane rodičem. Vždy dva rodiče se spolu kříží (viz podkapitola 4.2.2) a vzniknou dva potomci. Křížení probíhá vždy, ale když je jako bod křížení vybrán první nebo poslední prvek genomu, tak se křížení neprojeví. Na potomky je dále aplikován operátor mutace, který je popsán v podkapitole 4.2.1. Takto vytvoříme tolik potomků, jaká je velikost populace, provedeme jejich vývoj a vyhodnotíme fitness.

Potomci jsou umístěni do předchozí populace a ta je seřazena podle hodnoty fitness. Nejhorší jedinci, kteří přesahují velikost populace, jsou zahozeni. Takto evoluce pokračuje, dokud nenalezne jedince s nejlepší možnou fitness (například funkční kombinační obvod), nebo nedosáhne nastaveného omezení počtu generací.

Kapitola 3

Booleovské sítě

Booleovské sítě (Random Boolean Networks, dále RBN) byly navrženy Stuartem Kauffmannem (viz [9]) jako model pro genetické regulační sítě. Původním cílem byl tedy výzkum principů života, ale podle [5] našly uplatnění například v matematice, sociologii, neuronových sítích a robotice. Od Kauffmanova návrhu vzniklo několik modifikací, ale v této kapitole bude popsána původní verze.

3.1 Struktura booleovské sítě

Síť se skládá z N uzlů. Každý uzel může nabývat hodnoty 1 nebo 0 (odtud booleovské) a je připojen na K libovolných uzlů. Může být připojen víckrát k jednomu uzlu i sám k sobě. Propojení uzlů je hlavní rozdíl oproti binárnímu celulárnímu automatu, kde následující stav závisí na buňkách v sousedství. Booleovské sítě jsou tedy zobecněním binárního celulárního automatu. Na obrázku 3.3 v části (a) je zobrazen příklad schématu booleovské sítě s $N = 3$ a $K = 2$.

Každý uzel v sobě obsahuje logickou funkci f_i, která ze stavů $s_i(t)$ připojených uzlů určí další stav $s_i(t+1)$. Podle Kauffmanova modelu závisí stav uzlu v čase $t+1$ pouze na stavech připojených uzlů v čase t. Při výpočtu musíme mít uložen stávající stav, spočítat následující stav pro všechny uzly a pak aktualizovat stav všech uzlů. Logické funkce pro jednotlivé uzly jsou uloženy ve vyhledávacích tabulkách. Příklad takové tabulky je na obrázku 3.3 v části (c). Velikost tabulky je 2^K hodnot (všechny kombinace hodnot vstupních uzlů). Důležité je tedy i pořadí v jakém jsou uzly připojeny. Náhodné vytvoření sítě je popsáno v algoritmu 3.1 a vývoj, který v síti probíhá, je popsán v algoritmu 3.2.

- Náhodně generuj logickou funkci f_i s K vstupy pro každý uzel.
- Náhodně připoj ke každému uzlu K uzlů.
- Náhodně vygeneruj počáteční stav pro každý uzel (1 nebo 0).

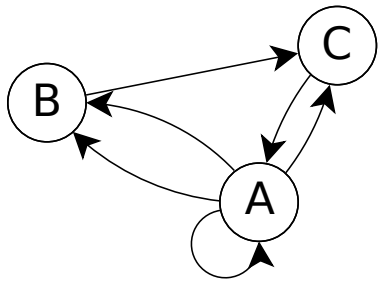
Algoritmus 3.1: Náhodné vytvoření booleovské sítě

Kvůli lepšímu pochopení si popíšeme obrázek 3.3. V části (a) je grafické znázornění sítě. Uzel je ovlivněn těmi uzly, z kterých do něj směřují šipky. Takže například uzel A

- V každém kroku t:
  - Pro každý uzel i $(0 < i \leq N)$ urči jeho následující hodnotu $s_i(t+1)$ z připojených uzlů pomocí funkce f_i.
  - Aktualizuj hodnotu všech uzlů.

Algoritmus 3.2: Vývoj booleovské sítě

je ovlivněn uzlem C a sám sebou a uzel B je ovlivněn pouze uzlem A. V části (c) vidíme tabulky logických funkcí f_i pro jednotlivé uzly. V části (b) je ukázán vývoj sítě z počátečního stavu, kdy $t = 0$. V počátečním stavu mají všechny uzly hodnotu 0. Prvním vstupem uzlu A je uzel A a druhým je uzel C. Hodnoty obou uzlů jsou v čase $t = 0$ rovny 0, použijeme tedy řádek 00 a vidíme, že v čase $t = 1$ bude mít uzel A hodnotu 1. Stejně budeme postupovat u dalších uzlů. Když známe hodnoty všech uzlů, můžeme nastavit spočítané stavy uzlů jako aktuální a zvýšit čas na $t = 1$. Stejným způsobem potom vývoj sítě pokračuje dál.



(a)

	t					
	0	1	2	3	4	5
$s_A(t)$	0	1	0	0	0	1
$s_B(t)$	0	1	0	1	1	1
$s_C(t)$	0	1	1	1	0	0

(b)

A			B			C		
$s_A(t)$	$s_C(t)$	$s_A(t+1)$	$s_A(t)$	$s_A(t)$	$s_B(t+1)$	$s_B(t)$	$s_A(t)$	$s_C(t+1)$
0	0	1	0	0	1	0	0	1
0	1	0	0	1	1	0	1	0
1	0	1	1	0	0	1	0	0
1	1	0	1	1	0	1	1	1

(c)

Obrázek 3.3: Příklad booleovské sítě a jejího vývoje:
(a) graf booleovské sítě, (b) znázornění vývoje, (c) logické funkce pro jednotlivé uzly

3.2 Možnosti booleovských sítí

V [20] je ukázáno, že pomocí dvourozměrného celulárního automatu lze simulovat výpočetně úplnou množinu hradel. Tento celulární automat však není binární. V [15] je dokázáno, že pomocí binárního celulárního automatu, který používá pravidlo 110, lze simulovat determi-

nistický Turingův stroj. Vzhledem k tomu, že booleovské sítě jsou zobecněním celulárních automatů, tak by mělo být možné pomocí vhodných booleovských sítí realizovat jakýkoli algoritmus. Klíčovým problémem je tedy pro daný úkol nalézt tu správnou síť. Jen všech možných logických funkcí pro jeden uzel je 2^{2^K} a počet všech možných sítí o velikosti N a počtu připojení K je uveden v [5] a lze jej vyjádřit rovnicí:

$$\text{počet všech možných sítí} = \left(\frac{2^{2^K} N!}{(N-K)!} \right)^N . \tag{3.1}$$

Toto nám pro výpočet, který chceme realizovat sítí s $K = 3$ a $N = 8$, dává přibližně $3 \cdot 10^{39}$ možných sítí. Tedy i pro poměrně malou síť dostaneme velký stavový prostor, který je třeba prohledat.

Kapitola 4

Booleovské sítě v evolučním návrhu

V předchozí kapitole jsme si popsali Kauffmanův model booleovských sítí. Ten je třeba nějakým způsobem reprezentovat, aby jej bylo možno implementovat a mohly být navrženy genetické operátory křížení a mutace. Tato reprezentace bude sloužit v konkrétních aplikacích, jenž jsou uvedeny v dalších kapitolách.

4.1 Návrh reprezentace booleovských sítí

Je třeba navrhnout, jakým způsobem budou realizována propojení a tabulky logických funkcí pro jednotlivé uzly. Propojení lze realizovat pomocí vektoru přirozených čísel. Každý uzel v síti dostane podle svého umístění pořadové číslo i $(0 < i \leq N)$. Ke každému uzlu i je připojeno K uzlů. Těmto připojením přiřadíme pořadové číslo j $(0 < j \leq K)$. Výslednou reprezentací všech propojení v síti je vektor

$$\vec{c} = (c_{11}, c_{12}, \ldots, c_{1K}, \ldots, c_{ij}, \ldots, c_{NK}), \tag{4.1}$$

kde c_{ij} je pořadové číslo uzlu, který je připojen k uzlu i na j-tou pozici.

Tabulky logických funkcí lze realizovat podobným způsobem. Tentokrát se však bude jednat o binární vektor. Nejprve si ukážeme tvar tabulky pro jeden uzel a získání hodnoty z ní. Tabulka pro uzel i je reprezentována binárním vektorem

$$\vec{f_i} = (f_{i0}, f_{i1}, \ldots, f_{ik}, \ldots, f_{i(2^K-1)}). \tag{4.2}$$

Index k, který potřebujeme pro získání hodnoty uzlu i v čase $t+1$, získáme převedením binárního vektoru $\vec{v_i}$ na číslo v desítkové soustavě. $\vec{v_i}$ má tvar

$$\vec{v_i} = (v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots v_{iK}). \tag{4.3}$$

Kde v_{ij} je hodnota uzlu c_{ij} v čase t, v_{i1} je nejvýznamnější bit a v_{iK} nejméně významný. Tabulky pro celou síť jsou reprezentovány zřetězením vektorů f_i pro jednotlivé uzly. Výsledný vektor má tvar

$$\vec{f} = (f_{10}, f_{11}, \ldots, f_{1(2^K-1)}, f_{20}, \ldots, f_{N(2^K-1)}). \tag{4.4}$$

4.2 Genetické operátory pro booleovské sítě

Vektor připojení $\vec{c}$ je tvořen celými čísly a vektor reprezentující tabulky logických funkcí $\vec{f}$ je binární. Oba tyto vektory dohromady tvoří jedince. Booleovskou síť B tedy budeme reprezentovat jako dvojici vektorů:

$$B = (\vec{c}, \vec{f}). \tag{4.5}$$

Operátory křížení a mutace pracují s každým z těchto dvou vektorů zvlášť.

4.2.1 Mutace

Pro vektor připojení $\vec{c}$ zvolíme pravděpodobnost mutace $p_{mc} \in (0, 1)$. Každá složka vektoru bude mutovat s pravděpodobností p_{mc}. Pokud zmutuje, tak se náhodně vybere jiný uzel, ke kterému připojení povede.

Pro vektor tabulek logických funkcí $\vec{f}$ zvolíme pravděpodobnost mutace $p_{mf} \in (0, 1)$. Každá složka vektoru bude s pravděpodobností p_{mf} negována.

4.2.2 Křížení

Booleovské sítě budeme křížit pomocí jednobodového křížení (viz 2.1.2). Pro každý vektor, kterým reprezentujeme booleovskou síť, zvolíme náhodně bod křížení.

Nejdříve náhodně zvolíme bod křížení B_c ($1 < B_c < |\vec{c}|$) a potomkům přiřadíme odpovídající části vektorů $\vec{c}$, které náležely rodičům. Poté náhodně zvolíme bod křížení B_f ($1 < B_c < |\vec{f}|$) a postupujeme stejně jako v předchozím případě. V následujících kapitolách budou booleovské sítě rozšířeny ještě o další vektory, jenž budou použity ke specifickým účelům. Křížení však bude probíhat stejně jako u předchozích dvou vektorů.

Kapitola 5

Návrh kombinačních obvodů pomocí booleovských sítí

V [3] je popsána metoda pro evoluční návrh kombinačních obvodů na úrovni hradel. Jako vývojový model je použit celulární automat. V této kapitole bude popsána upravená metoda, která místo celulárního automatu bude používat booleovskou síť. Použijeme původní Kauffmanův model tak, jak je popsán v kapitole 3 s reprezentací a genetickými operátory podle kapitoly 3. Ten rozšíříme o část sloužící k návrhu kombinačního obvodu.

5.1 Návrh obvodu booleovskou sítí

Obvod bude složen z logických hradel se dvěma vstupy. Použitá hradla jsou uvedena v tabulce 5.1. Počet hradel v jedné úrovni bude roven počtu vstupů obvodu (M). Bude umožněno jen připojení k předchozí úrovni. První úroveň bude připojena na vstupy a k poslední budou připojeny výstupy. Každé hradlo v jedné úrovni obvodu je generováno jedním uzlem booleovské sítě. Počet uzlů booleovské sítě musí být alespoň takový, jako je počet vstupů generovaného obvodu. Celkový počet uzlů v síti (N) může být i vyšší. Část uzlů generuje hradla a část slouží pouze pro zvýšení variability sítě. V následujícím popisu se uvažuje, že N je rovno počtu vstupů, ale experimenty byly prováděny i se sítěmi s vyšším N.

Číslo	Typ	Vstupy	Popis
0	AND	in_1, in_2	logický součin vstupů in_1 a in_2
1	OR	in_1, in_2	logický součet vstupů in_1 a in_2
2	XOR	in_1, in_2	exkluzivní logický součet vstupů in_1 a in_2
3	IDA	in_1,–	jednobitový buffer, výstup je stejný jako vstup in_1
4	IDB	–, in_2	jednobitový buffer, výstup je stejný jako vstup in_2

Tabulka 5.1: Použitá hradla

Logickou funkci f_i z algoritmu 3.2 upravíme tak, aby byl definován nejen další stav uzlu, ale také hradlo, které v tom kroku vznikne. Pokud použijeme síť s počtem připojených uzlů $K = 3$, máme pro každý uzel i definovánu přechodovou funkci jako množinu pravidel tvaru

$$v_{i1}v_{i2}v_{i3} \rightarrow s_i(t+1), \tag{5.1}$$

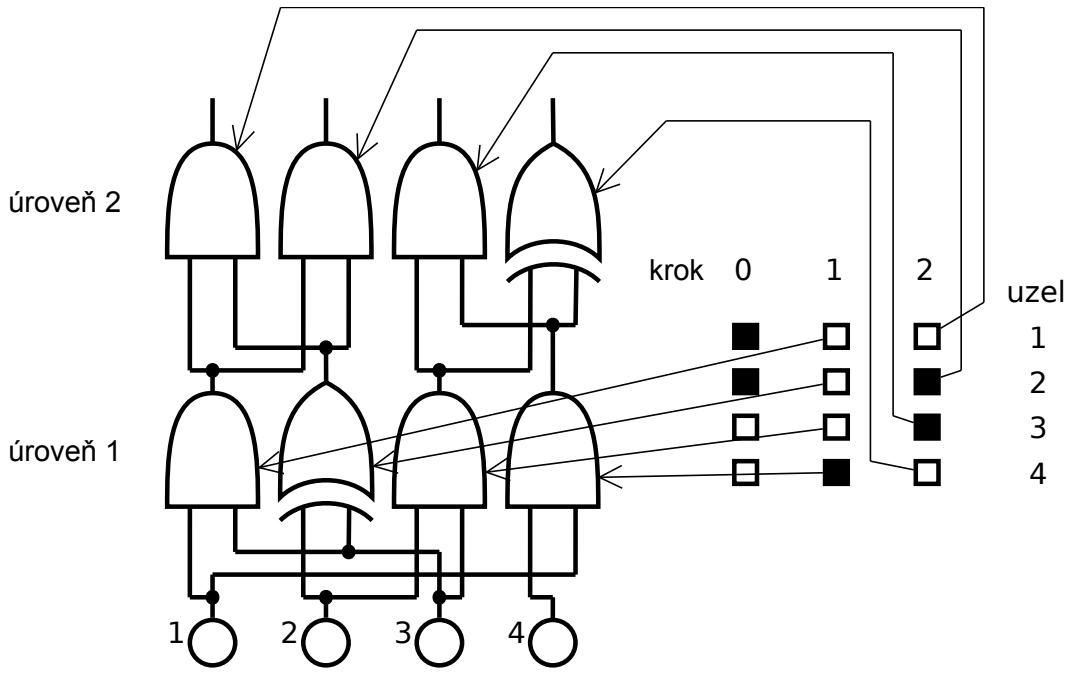
kde $s_i(t+1)$ je stav uzlu i v čase $t+1$ a $v_{i1}v_{i2}v_{i3}$ jsou hodnoty uzlů připojených k uzlu i na první, druhé a třetí pozici. Pokud chceme navrhovat kombinační obvody, musí pravidlo specifikovat hradlo, které má být vytvořeno, a tak pro návrh kombinačních obvodů použijeme množinu pravidel tvaru

$$v_{i1}v_{i2}v_{i3} \rightarrow s_{i(t+1)} : g\ in_1\ in_2, \tag{5.2}$$

kde g je hradlo z tabulky 5.1 generované uzlem i, in_1 a in_2 jsou indexy pinů, na které jsou vstupy hradla připojeny. V první úrovni jsou to vstupy obvodu a v dalších úrovních výstupy hradel z předchozí úrovně. Pokaždé, když se určuje následující stav některého uzlu, se také vytvoří nové hradlo. Jedna úroveň obvodu vznikne v jednom kroku booleovské sítě.

Příklad vytvoření obvodu je znázorněn na obrázku 5.1. Vlevo je obvod a vpravo je počáteční stav a první dva kroky booleovské sítě. Bílé čtverečky přestavují uzly ve stavu 1, černé ve stavu 0. V tabulce 5.2 je část pravidel pro uzel 2. K tomu je na pozici jedna připojen uzel 1, na pozici 2 také uzel 1 a na pozici 3 uzel 3. V počátečním stavu sítě vybereme z tabulky 5.2 řádek, kde jsou hodnoty připojených uzlů 0 0 1, následující stav bude 1 a vytvoříme v první úrovni hradlo na pozici 2 připojené na vstupy 2 a 3. To se provede i pro ostatní uzly (podle jejich vlastních tabulek, které tu nejsou uvedeny). V dalším kroku sítě vybereme z tabulky 5.2 řádek 1 1 1, který odpovídá hodnotám připojených uzlů. Vytvoří se hradlo AND na pozici dvě v druhé úrovni. To je připojeno k hradlům v první úrovni na pozicích 1 a 2. Pak se určí nové stavy dalších uzlů a vytvoří se odpovídající hradla. Tak se pokračuje, dokud není získán funkční obvod, nebo není dosaženo maximálního počtu kroků.

Pokud by byl počet uzlů N větší než počet vstupů M, tak M uzlů bude vytvářet hradla a bude pro ně platit tabulka podobná tabulce 5.2. Ostatní uzly nebudou v tabulce mít sloupce g,i_1 a i_2.



Obrázek 5.1: Vytváření obvodu booleovskou sítí

v_{21}	v_{22}	v_{23}	$s_{2(t+1)}$	g_2	i_1	i_2
0	0	0	1	OR	3	4
0	0	1	1	XOR	2	3
			...			
1	1	1	0	AND	1	2

Tabulka 5.2: Část tabulky pro uzel 2 z obrázku 5.1
Připojení jsou $c_{21} = 1, c_{22} = 1, c_{23} = 3$

5.2 Reprezentace a genetické operátory

V kapitole 4.1 byla popsána reprezentace booleovských sítí. Tuto reprezentaci použijeme a přidáme k ní další vektor $\vec{r}$. Ten bude obsahovat tabulky pravidel pro jednotlivé uzly. Tato pravidla budou spojena za sebou pro všechny uzly. Výsledný vektor bude mít tvar

$$\vec{r} = (g_{10}, in_{110}, in_{210}, g_{11}, in_{111}, in_{211}, \ldots, g_{1(2^K-1)}, in_{11(2^K-1)}, in_{21(2^K-1)}, \ldots \\ \ldots, g_{M(2^K-1)}, in_{1M(2^K-1)}, in_{2M(2^K-1)}). \tag{5.3}$$

Pokud budeme chtít získat z tohoto vektoru hodnoty pro určitý uzel, použijeme následující vztahy uvedené v tabulce 5.3. Pro úplnost jsou zde i vztahy pro vektory $\vec{c}$ a $\vec{f}$.

Hodnota	Vektor	Vzorec
Uzel připojený k i-tému uzlu na j-té pozici	$\vec{c}$	$(i-1) \cdot K + j$
Následující hodnota uzlu i	$\vec{f}$	$(i-1) \cdot 2^K + k + 1$
Hradlo vytvořené na i-té pozici	$\vec{r}$	$((i-1) \cdot 2^K + k + 1) \cdot 3$
Vstup in_1 hradla na i-té pozici	$\vec{r}$	$((i-1) \cdot 2^K + k + 1) \cdot 3 + 1$
Vstup in_2 hradla na i-té pozici	$\vec{r}$	$((i-1) \cdot 2^K + k + 1) \cdot 3 + 2$

Tabulka 5.3: Přístup k hodnotám ve vektorech $\vec{c}$,$\vec{f}$ a $\vec{r}$, index i se počítá od 1

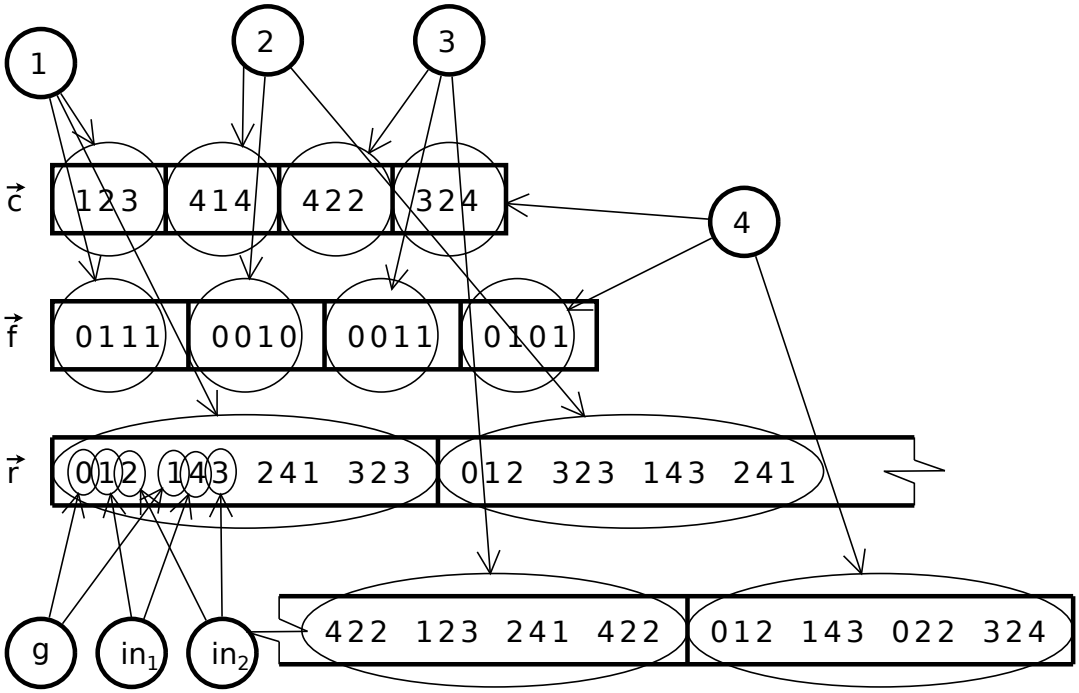
Jedince nám tedy budou tvořit vektory $\vec{c}$, $\vec{f}$ a $\vec{r}$. Na obrázku 5.2 je zobrazen příklad genomu. Je tam vidět, které části patří k uzlům 1–4. Také je u prvních dvou částí znázorněno, která část vektoru $\vec{r}$ odpovídá hradlu a vstupům.

Křížení a mutace odpovídá postupu popsaném v kapitole 4.2. Mutace proběhne stejně jako u vektoru $\vec{c}$. Pravděpodobnost mutace se pro každou položku vektoru vyhodnocuje zvlášť. Pokud mutuje položka odpovídající hradlu, je novou hodnotou náhodně zvolené hradlo. U vstupů je novou hodnotou náhodně vybraný vstup.

5.3 Fitness funkce

V [22] je fitness funkcí počet shodných bitů na výstupu evolucí navrženého obvodu s požadovaným výstupem pro všechny možné binární testovací vektory obvodu. Požadovaný výstup je dán pravdivostní tabulkou navrhovaného obvodu. Počet všech možných binárních testovacích vektorů pro obvod s V vstupy je 2^V.

V našem případě je fitness funkce počet bitů, které se od požadovaného výstupu liší. Evoluční algoritmus se tedy snaží fitness funkci minimalizovat a dosáhnout hodnoty 0, což je funkční obvod.



Obrázek 5.2: Příklad kompletního genomu pro $K = 2$, $M = 4$ a $N = 4$

5.4 Experimenty

Byl proveden návrh sčítačky, násobičky, řadicího obvodu a mediánového obvodu. U každého z těchto obvodů bylo experimentálně ověřeno, jak velký obvod lze nalézt, jak se evoluce chová pro různé hodnoty N a jestli mají uzly navíc (ty, které negenerují žádná hradla) nějaký vliv na úspěšnost evoluce, nebo vlastnosti vytvářených obvodů.

Předem nevíme kolik, kroků bude booleovská síť potřebovat ke tvorbě funkčního obvodu. Fitness funkce je proto vyhodnocována postupně pro každý počet úrovní až do stanoveného maxima. Výsledná fitness jedince odpovídá hodnotě fitness pro ten počet úrovní, který dosáhl nejlepších výsledků.

Pro každou variantu (velikost obvodu, různá N a K) bylo provedeno 100 běhů evoluce omezených na 1000 generací a v každé generaci bylo 1000 jedinců. Údaj, že evoluce byla úspěšná například v 50% případů znamená, že v padesáti běžích evoluce byl nalezen funkční obvod dříve, než bylo dosaženo maximálního počtu generací. Parametry evoluce byly zvoleny na základě krátkých experimentů během implementace. Pravděpodobnost mutace byla stanovena na 5%. Počet generací je omezen na 1 000. A jedinců v populaci je taktéž 1 000. Počáteční stav všech uzlů sítě je stanoven na 0.

5.4.1 Násobička

Největší násobička, kterou se podařilo nalézt, má velikost 2×2 bity. V tabulce 5.4 jsou uvedeny výsledky experimentů. Sloupce označené počet generací ukazují nejmenší, největší a průměrný počet generací, který byl potřeba k nalezení funkčního obvodu (počítají se pouze úspěšné běhy). Sloupec počet úrovní ukazuje, jaké zpoždění měly nalezené funkční obvody. Sloupec počet hradel uvádí počty hradel v nalezených funkčních obvodech (v tomto případě nepočítáme IDA a IDB jako hradla).

N	K	úspěšnost	počet generací			počet úrovní			počet hradel		
			min	průměr	max	min	průměr	max	min	průměr	max
4	2	86%	61	200.49	952	3	4.13	8	7	11.74	28
6	2	85%	78	227.42	940	3	4.15	8	7	11.71	25
8	2	85%	51	211.36	585	3	4.32	8	7	12.24	25
10	2	86%	63	279.26	865	3	4.43	8	7	12.80	31
12	2	87%	78	236.33	946	3	4.56	8	7	13.16	32
14	2	90%	87	330.99	977	3	4.27	8	7	11.89	24
16	2	92%	100	277.96	888	3	4.03	8	7	11.52	25
18	2	91%	83	292.53	987	3	4.25	8	7	11.89	28
20	2	89%	69	291.52	946	3	4.16	8	7	11.67	28
28	2	93%	80	288.52	882	3	4.01	8	7	11.42	30
4	3	90%	45	243.92	976	3	4.17	8	7	11.32	20
5	3	84%	78	267.15	945	3	4.23	8	7	11.87	28
6	3	82%	94	237.24	803	3	4.10	8	7	11.28	25
8	3	85%	85	270.82	863	3	3.79	8	7	10.31	24
10	3	89%	90	271.51	774	3	3.54	8	7	10.16	32
12	3	93%	110	311.06	896	3	3.72	8	7	10.78	26
14	3	88%	97	338.50	914	3	3.65	8	7	10.26	29
16	3	87%	46	302.24	854	3	3.72	8	7	10.49	25
4	4	82%	50	204.20	843	3	4.02	8	7	11.27	23
6	4	84%	110	279.07	961	3	3.77	8	7	10.20	25
8	4	89%	102	316.74	913	3	3.37	8	7	9.54	22
10	4	91%	140	327.97	963	3	3.38	8	7	9.70	28
12	4	92%	132	318.42	756	3	3.28	6	7	9.33	18
14	4	89%	110	335.87	987	3	3.26	8	7	9.15	18
16	4	93%	106	361.98	967	3	3.24	6	7	8.99	14

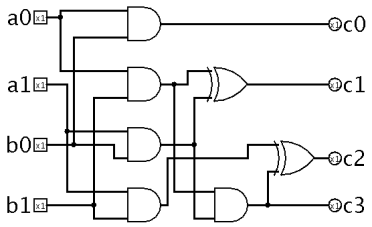
Tabulka 5.4: Výsledky experimentů pro násobičku 2×2 bity, omezení počtu úrovní je 8.

Jak je patrné z tabulky 5.4, počet uzlů sítě N ani počet propojení K nemají výraznější vliv na úspěšnost evolučního procesu. Podle počtu generací lze usoudit, že nejrychleji se dařilo hledat sítě s $N = 2$. Se vzrůstajícím N a K se u potřebného počtu generací vyskytují vyšší hodnoty. Při všech testovaných kombinacích N a K se podařilo najít obvody s nejmenším zpožděním 3 a počtem hradel 7, což jsou nejlepší nalezené hodnoty. Při vyšších N a K mají nalezené násobičky průměrně menší počet úrovní i hradel.

Nyní se budeme věnovat booleovské síti, která vytvořila jednu z nejlepších násobiček. V tabulce 5.5 jsou uvedena pravidla, která evoluce našla pro jednotlivé uzly. Uzel označuje index uzlu počítaný od 0. Pravidla jsou uvedena ve tvaru popsaném v rovnici 5.2.

Na obrázku 5.3 v části (a) je schéma vytvořené násobičky. Vlevo jsou vstupy obvodu a vpravo výstupy. Skládá se pouze z hradel AND a XOR. V prvním kroku všechny uzly vytvořily hradlo AND, v dalších krocích se již vygenerovaná hradla liší. Na obrázku 5.3 v části (b) je tabulka s postupem vytvoření obvodu. Připojení uvádí indexy uzlů, ke kterým je daný uzel připojen. V dalších řádcích jsou uvedeny stavy jednotlivých uzlů a hradla, která byla vytvořena při přechodu do následujícího stavu.

Na obrázcích 5.4 a 5.5 jsou další příklady nalezených násobiček. Ta na obrázku 5.4 je velmi podobná té z obrázku 5.3, ale uzel s indexem 2 se ustálil již v druhém kroku a hradlo XOR muselo být vygenerováno jiným uzlem. Síť poté potřebovala ještě jeden krok, aby dovedla výstup hradla XOR na správnou pozici. Na obrázku 5.5 je násobička, kde se vůbec nevyskytují buffery IDA a IDB. Místo nich jsou použita hradla OR se vstupy připojenými na výstup stejného hradla.



(a)

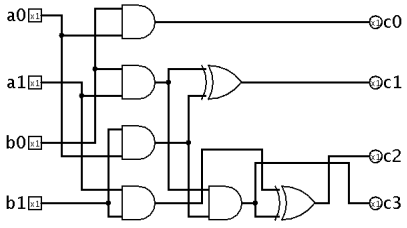
uzel	0	1	2	3
připojení	0 1	2 0	2 0	2 3
stav	0	0	0	0
hradlo	AND 0 2	AND 3 0	AND 2 1	AND 1 3
stav	1	0	0	1
hradlo	IDB 2 0	XOR 2 1	IDA 3 3	AND 2 1
stav	1	0	1	0
hradlo	IDB 2 0	IDA 1 0	XOR 2 3	IDA 3 0
stav	1	0	1	1

(b)

Obrázek 5.3: Jedna z nejlepších nalezených násobiček ($N = 4$, $K = 2$) a postup jejího vytvoření.

uzel	0	1	2	3
pravidla	$00 \to 1 : AND\ 0\ 2$	$00 \to 0 : AND\ 3\ 0$	$00 \to 0 : AND\ 2\ 1$	$00 \to 1 : AND\ 1\ 3$
	$01 \to 1 : IDB\ 2\ 0$	$01 \to 1 : OR\ 0\ 1$	$01 \to 0 : AND\ 2\ 1$	$01 \to 1 : IDA\ 3\ 0$
	$10 \to 1 : AND\ 3\ 2$	$10 \to 0 : XOR\ 2\ 1$	$10 \to 1 : IDA\ 3\ 3$	$10 \to 0 : AND\ 2\ 1$
	$11 \to 1 : IDB\ 3\ 2$	$11 \to 0 : IDA\ 1\ 0$	$11 \to 1 : XOR\ 2\ 3$	$11 \to 1 : IDB\ 0\ 2$

Tabulka 5.5: Ukázka nalezených pravidel.



(a)

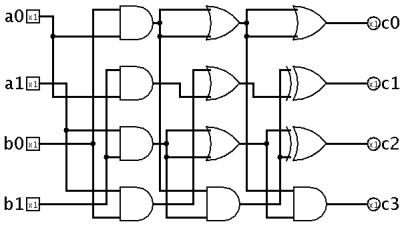
úroveň			
1	2	3	4
AND 2 0	IDA 0 2	IDA 0 0	IDA 0 0
AND 2 1	XOR 1 2	IDB 0 1	IDB 0 1
AND 3 0	IDA 3 1	IDA 3 1	IDA 3 1
AND 1 3	AND 1 2	XOR 3 2	IDB 0 2

(b)

Obrázek 5.4: Jedna z násobiček nalezených s $N = 5$ a $K = 3$.

5.4.2 Sčítačka

Na rozdíl od násobičky se podařilo nalézt sčítačku $3 + 3$ bity. To je způsobeno také tím, že násobička je pro evoluční návrh složitější (viz [19]). Výsledky experimentů jsou shrnuty v tabulce 5.6. Sloupce mají stejný význam jako u násobičky. U experimentů pro $K = 2$ byl



(a)

úroveň		
1	2	3
AND 2 0	OR 0 0	OR 0 0
AND 3 0	OR 3 1	XOR 3 1
AND 3 1	OR 2 2	XOR 2 3
AND 1 2	AND 0 2	AND 0 2

(b)

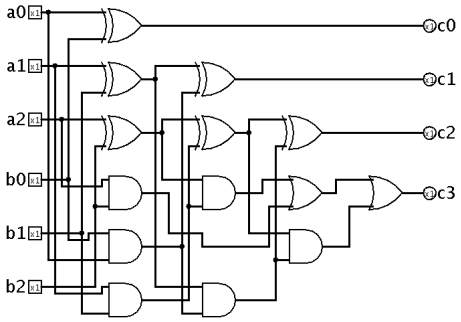
Obrázek 5.5: Jedna z násobiček nalezených s $N = 6$ a $K = 3$.

maximální počet úrovní nastaven na 15. Z výsledků je vidět, že tato velikost je o mnoho větší než nejmenší nalezená. Proto byla horní hranice pro $K = 3$ a $K = 4$ stanovena na 10.

V tabulce 5.6 jsou uvedeny výsledky experimentů pro sčítačku $3 + 3$ bity. Experimenty s nastaveným $K = 4$ se vyznačují nižší úspěšností, ale na druhou stranu také menším průměrným zpožděním nalezených sčítaček. Nejlepší sčítačka byla nalezena při nastavení $N = 8$ a $K = 3$. Na obrázku 5.6 v části (a) je její schéma a v části (b) jsou hradla vytvořená pro jednotlivé úrovně. Je vidět, že jsou použity všechny typy hradel.

N	K	úspěšnost	počet generací			počet úrovní			počet hradel		
			min	průměr	max	min	průměr	max	min	průměr	max
6	2	12%	242	688.83	990	4	5.00	8	15	19.42	24
8	2	13%	336	627.46	921	4	5.92	10	14	24.38	38
10	2	28%	108	618.71	979	4	5.71	9	16	24.75	39
12	2	15%	309	628.87	965	4	6.40	15	15	27.67	65
14	2	16%	268	704.69	997	4	5.19	8	17	21.75	35
16	2	23%	384	714.96	971	4	5.35	10	17	22.00	35
18	2	17%	276	678.65	951	4	5.41	9	17	22.71	35
20	2	29%	337	713.38	986	4	5.62	11	16	23.52	43
30	2	25%	283	650.48	945	4	5.80	12	16	25.16	53
6	3	14%	178	586.14	988	4	5.36	9	17	23.57	38
8	3	22%	256	616.91	962	4	5.55	10	17	24.27	43
10	3	23%	308	694.35	995	4	5.09	8	17	23.17	41
12	3	18%	307	646.78	882	4	5.39	10	16	22.06	36
14	3	9%	488	683.78	984	4	5.56	10	17	22.22	35
16	3	10%	503	774.70	954	4	4.50	6	16	19.50	27
6	4	12%	404	699.17	999	4	5.42	9	16	23.75	41
8	4	9%	263	678.89	994	4	4.67	6	15	20.78	31
10	4	8%	532	729.75	905	4	4.38	5	16	18.50	20
12	4	6%	424	742.33	914	4	5.33	8	16	24.00	35
14	4	7%	509	754.14	963	4	4.86	6	15	22.57	30
16	4	7%	529	821.00	982	4	4.71	7	16	20.29	30

Tabulka 5.6: Výsledky experimentů pro sčítačku $3 + 3$ bity, omezení počtu úrovní je 15 pro $K = 2$ a 10 pro $K = 3$ a $K = 4$.



(a)

úroveň			
1	2	3	4
XOR 3 0	IDB 3 0	IDB 3 0	IDB 3 0
XOR 1 4	XOR 1 4	IDA 1 1	IDA 1 1
XOR 5 2	XOR 5 2	XOR 5 2	IDB 0 2
AND 2 5	AND 2 5	OR 3 4	OR 3 4
AND 3 0	IDA 3 0	AND 2 5	IDB 5 4
AND 1 4	AND 1 4	IDB 3 3	IDB 3 3

(b)

Obrázek 5.6: Nejlepší sčítačka ($N = 8$, $K = 2$) a vytvořená hradla.

N	K	úspěšnost	počet generací			počet úrovní			počet hradel		
			min	průměr	max	min	průměr	max	min	průměr	max
6	2	2%	490	546.00	602	8	10.00	12	44	52.00	60
8	2	8%	512	660.00	933	5	9.00	14	30	49.75	72
10	2	7%	473	636.86	945	6	9.00	14	36	49.43	74
12	2	10%	397	685.00	986	5	9.40	14	30	52.90	78
14	2	10%	475	700.80	969	6	10.00	15	34	54.60	90
16	2	9%	305	601.56	923	5	8.22	12	30	47.44	70
6	3	1%	530	530.00	530	12	12.00	12	60	60.00	60
8	3	3%	813	864.00	910	6	8.00	12	36	41.00	51
10	3	3%	435	536.00	589	8	9.00	11	44	52.67	66
12	3	5%	606	794.60	926	7	8.80	12	42	50.40	72
14	3	4%	436	731.00	905	7	9.00	11	42	53.50	66
16	3	4%	727	843.50	963	8	8.75	9	38	48.50	54
6	4	2%	544	597.50	651	5	5.50	6	30	33.00	36
8	4	3%	472	716.67	915	7	8.33	10	41	44.33	48
10	4	2%	740	859.50	979	7	8.00	9	40	47.00	54
12	4	4%	534	774.75	998	6	8.00	12	33	45.75	66
14	4	1%	662	662.00	662	10	10.00	10	60	60.00	60

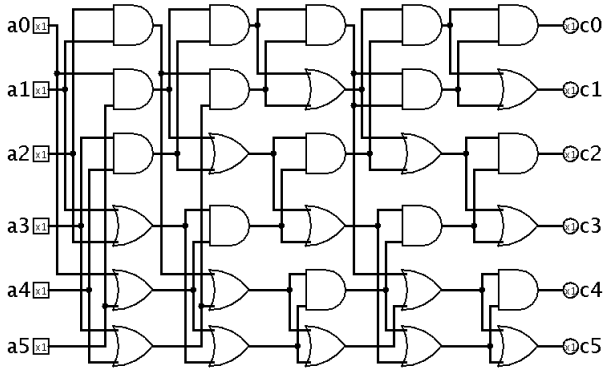
Tabulka 5.7: Výsledky experimentů pro řadicí obvod se šesti vstupy.

5.4.3 Řadicí obvod

Pro experimenty s řadicími obvody nebylo evoluci umožněno použít hradlo XOR, protože pokud jsou při návrhu použita jen hradla AND a OR, lze je v obvodu nahradit prvky maximum a minimum a použít i pro jiné než binární hodnoty. Podrobněji je tato problematika popsána v kapitole 6.1. Největší řadicí obvod, který se podařilo navrhnout, má šest vstupů.

V tabulce 5.7 jsou výsledky experimentů pro šestibitový řadicí obvod. Úspěšnost evoluce je nižší než u předchozích experimentů. Pro $K = 2$ byla úspěšnost nejlepší a pak postupně klesala. Při nastavení $N = 16$ a $K = 4$ se nepodařilo nalézt žádný řadicí obvod. Pro vyvozování nějakých závěrů o vlivu nastavení N a K na počet úrovní a hradel bylo nalezeno příliš málo funkčních řadicích obvodů.

Na obrázku 5.7 v části (a) je schéma jednoho z nejlepších nalezených řadicích obvodů a v části (b) jsou uvedena hradla, která byla vygenerována booleovskou sítí v jednotlivých krocích. Třetí a pátá úroveň sítě jsou shodné a čtvrtá se liší od druhé jen v hradlu AND, které bylo použito pro přenos hodnoty svého vstupu do další úrovně. Také lze vidět, že ve všech úrovních kromě čtvrté se vyskytují vždy dvojice hradel AND a OR, které mají stejné vstupy. To odpovídá struktuře komparátoru, která je popsána v kapitole 6.1.



(a)

úroveň				
1	2	3	4	5
AND 2 1	AND 2 1	AND 1 0	AND 2 1	AND 1 0
AND 5 0	AND 5 0	OR 1 0	AND 0 0	OR 1 0
AND 3 4	OR 2 1	AND 2 3	OR 2 1	AND 2 3
OR 2 1	AND 3 4	OR 2 3	AND 3 4	OR 2 3
OR 0 5	OR 5 0	AND 4 5	OR 5 0	AND 4 5
OR 3 4	OR 3 4	OR 5 4	OR 3 4	OR 5 4

(b)

Obrázek 5.7: Jeden z nejlepších šestibitových řadicích obvodů ($N = 8$, $K = 2$) a vytvořená hradla.

5.4.4 Mediánový obvod

Z důvodu uvedeného u předchozího experimentu bylo i u mediánového obvodu vypuštěno hradlo XOR. Nejdříve byly provedeny experimenty pro mediánový obvod o sedmi vstupech, jejich výsledky jsou v tabulce 5.8. Na rozdíl od násobiček a sčítaček nevznikají pro vyšší K obvody s menším počtem úrovní a hradel, spíše naopak. S rostoucím K se také snižuje úspěšnost evoluce a zvyšuje počet generací potřebných k nalezení funkčního řešení.

Nejlepší nalezený mediánový obvod se sedmi vstupy má sedm úrovní a ve tvaru, kterém byl navržen evolucí má 36 hradel. Na obrázku 5.8 v části (b) jsou hradla vygenerována pro jednotlivé úrovně. Při vytvoření čtvrté úrovně se booleovská síť ustálila a vytvořená hradla se pak opakují až do poslední úrovně. V části (a) je schéma nalezeného obvodu po odstranění zbytečných hradel, čímž se počet hradel snížil na 25. Stejně jako u řadicího obvodu z obrázku 5.7 se u navrženého mediánového obvodu vyskytují dvojice hradel AND a OR se stejnými vstupy. Není to však tak pravidelné jako u řadicího obvodu, protože

u mediánového obvodu nám stačí hodnota uprostřed seřazené posloupnosti, kdežto řadicí obvod musí tuto posloupnost poskytnout celou.

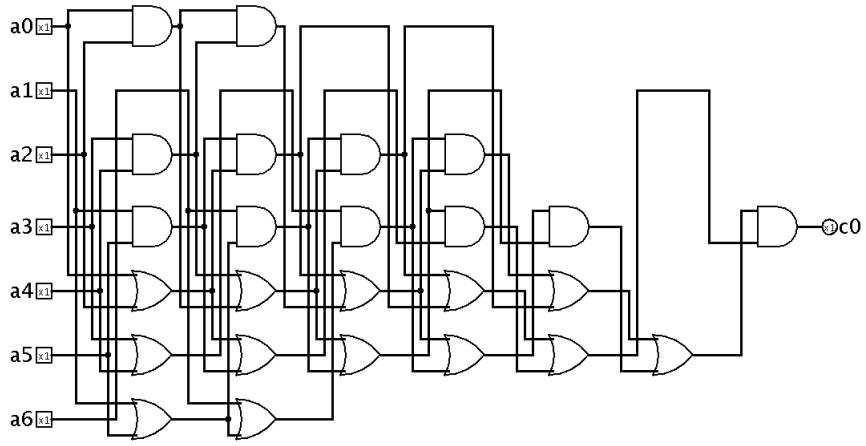
Největší mediánový obvod, který se podařilo navrhnout, má 9 vstupů. Byl proveden pouze jeden experiment o 100 evolučních bězích a na základě výsledků z tabulky 5.8 byly vybrány pro tento experiment parametry, při kterých bylo dosaženo nejvyšší úspěšnosti($N = 14$ a K) a zároveň byl nalezen nejlepší šestibitový mediánový obvod. Výsledky tohoto experimentu jsou v tabulce 5.9. Nejlepší zpoždění odpovídá počtu vstupů stejně jako u mediánového obvodu se sedmi vstupy.

N	K	úspěšnost	počet generací			počet úrovní			počet hradel		
			min	průměr	max	min	průměr	max	min	průměr	max
7	2	15%	199	448.87	885	8	10.40	14	42	62.80	84
8	2	34%	211	472.41	974	7	10.62	15	40	62.71	91
10	2	37%	197	370.81	866	7	10.92	15	40	65.51	91
12	2	40%	127	414.05	959	7	10.38	15	38	61.00	86
14	2	47%	123	387.09	866	7	10.57	14	36	61.57	90
16	2	34%	167	514.71	951	7	10.74	15	39	64.35	105
7	3	10%	355	587.90	876	7	10.30	13	42	61.80	79
8	3	22%	213	634.68	994	10	11.36	15	50	68.36	82
10	3	34%	212	478.26	952	7	11.06	15	42	65.09	98
12	3	30%	215	606.73	998	7	11.00	15	38	65.50	90
14	3	37%	207	584.30	1000	8	10.78	15	42	63.54	90
16	3	29%	387	668.38	989	7	10.72	15	42	62.69	92
7	4	3%	422	540.00	643	10	12.00	15	60	80.33	105
8	4	14%	290	571.07	991	8	11.50	15	48	69.36	97
10	4	29%	218	636.31	989	8	11.14	15	42	65.14	84
12	4	23%	290	638.78	950	7	11.00	15	41	63.70	96
14	4	27%	380	712.67	990	8	11.37	15	50	69.33	105
16	4	29%	424	769.52	1000	7	10.52	15	39	61.79	87

Tabulka 5.8: Výsledky experimentů pro sedmibitový mediánový obvod.

N	K	úspěšnost	počet generací			počet úrovní			počet hradel		
			min	průměr	max	min	průměr	max	min	průměr	max
14	2	15%	156	658.67	878	11	13.47	15	66	106.80	135

Tabulka 5.9: Výsledky experimentů pro devítibitový mediánový obvod.



(a)

úroveň						
1	2	3	4	5	6	7
AND 2 0	AND 2 0	IDA 2 4	IDA 2 4	IDA 2 4	IDA 2 4	IDA 2 4
IDB 0 6	IDB 0 5	IDB 0 5	IDB 0 5	IDB 0 5	IDB 0 5	IDB 0 5
AND 4 3	AND 4 3	AND 4 3	AND 4 3	AND 4 3	AND 4 3	AND 4 3
AND 5 1	AND 1 6	AND 1 6	AND 5 1	AND 5 1	AND 5 1	AND 5 1
OR 2 0	OR 2 0	OR 2 0	OR 2 0	OR 2 0	OR 2 0	OR 2 0
OR 4 3	OR 4 3	OR 4 3	OR 4 3	OR 4 3	OR 4 3	OR 4 3
OR 1 5	OR 1 6	IDB 0 2	AND 2 1	AND 2 1	AND 2 1	AND 2 1

(b)

Obrázek 5.8: Nejlepší šestibitový mediánový obvod ($N = 14$, $K = 2$) a vytvořená hradla.

Kapitola 6

Návrh řadicích sítí

Jako další využití booleovské sítě coby vývojového modelu představíme metodu pro návrh řadicích sítí. Navrhované řadicí sítě mají pevně určený počet vstupů. Na rozdíl od předchozí kapitoly nebude návrh probíhat na úrovni jednotlivých hradel, ale budou použity komparátory.

6.1 Řadicí sítě a komparátory

Komparátor je základním prvkem řadicí sítě. Má dva vstupy a dva výstupy. Prvky na vstupech porovná a předá na výstup tak, že pro výstupy a a b vždy platí, že $a \leq b$. Na obrázku 6.1 v části (a) je znázorněna realizace komparátoru pomocí maxima a minima. Ten lze použít pro řazení jakýchkoli prvků, které je možno porovnat. Pokud máme porovnávat pouze jednotlivé bity (a seřadit je do neklesající posloupnosti), je možno použít místo prvků maxima a minima hradla `AND` a `OR` (na obrázku 6.1 v části (b)). Část (c) znázorňuje značku komparátoru, která se používá při grafickém znázornění řadicích sítí.

Úkolem řadicí sítě je seřadit libovolnou posloupnost, která je přivedena na její vstup. Je realizována pomocí vhodně uspořádaných komparátorů. Tento koncept byl představen roku 1954 a jeho historie je popsána v [10]. Na rozdíl od klasických řadicích algoritmů je počet porovnání (a tím i složitost sítě) pevně dán a nezávisí na vstupních hodnotách. To je vhodné pro paralelní zpracování a hardwarovou implementaci. Pro síť o V vstupech jsou důležité dva parametry. Počet komparátorů, ze kterých se síť skládá, a zpoždění sítě. To udává počet skupin komparátorů, které musí být vykonány sekvenčně.

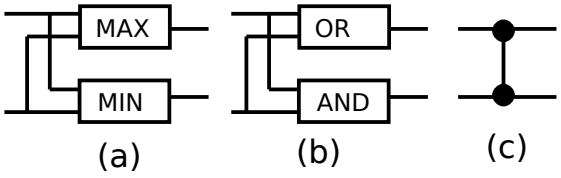
Pokud máme zjistit, zda síť seřadí správně všechny možné vstupy, je třeba vyzkoušet $V!$ vstupních vektorů. V [10] je uveden tzv. $1 - 0$ princip (zero-one principle). Ten říká, že pokud řadicí síť umí seřadit do neklesající posloupnosti všechny možné vstupní kombinace binárních hodnot, umí (pokud použijeme komparátor složený z prvků maximum a minimum) seřadit posloupnost libovolných hodnot. Díky tomu stačí vyzkoušet 2^V vstupních kombinací a navíc lze pracovat jen s binárními hodnotami.

6.2 Návrh řadicí sítě booleovskou sítí

Vytváření řadicí sítě je obdobné jako vytváření kombinačního obvodu v kapitole 5. Řadicí sítě jsou však složeny pouze z komparátorů, a tak bude množina pravidel pro vytváření komparátorů vypadat takto:

$$v_{i1}v_{i2}v_{i3} \rightarrow s_{i(t+1)} : in_1 \; in_2. \tag{6.1}$$

V tomto pravidle in_1 a in_2 jsou indexy připojených vstupů a zároveň výstupů komparátoru. V jednom kroku sítě je vytvořeno nejvýše $\frac{V}{2}$ komparátorů. Na rozdíl od kombinačních obvodů může být v jedné úrovni vstup použit pouze jedním komparátorem. Pokud k této situaci dojde, tak komparátor porušující tuto podmínku není vytvořen. Všechny ostatní detaily včetně fitness funkce a operátorů křížení a mutace jsou stejné jako v kapitole 5.



Obrázek 6.1: Komparátor znázorněn pomocí maxima a minima (a), hradel (b) a schématické značky (c)

6.3 Experimenty

V tabulce 6.1, která je převzata z [18], jsou uvedeny parametry nejlepších známých řadicích sítí. Cílem experimentů je zjistit, jaké řadicí sítě s pomocí booleovských sítí podaří navrhnout a porovnat jejich parametry s těmi známými.

Počet vstupů	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Zpoždění	0	1	3	3	5	5	6	6	7	8	8	9	10	10	10	10
Počet komparátorů	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60

Tabulka 6.1: Zpoždění a počet komparátorů u nejlepších známých řadicích sítí.

Nejdříve bylo provedeno několik experimentů s různým nastavením, aby bylo možno odhadnout vhodné nastavení pro finální sadu experimentů. Pro každou kombinaci hodnot bylo provedeno 100 běhů evoluce. Každý běh byl omezen na 1 000 generací a populace měla 1 000 jedinců. Pravděpodobnost mutace byla nastavena na 5%.

Výsledky jsou shrnuty v tabulce 6.2. Sloupce počet úrovní uvádějí zpoždění řadicí sítě a tím i počet kroků výpočtu booleovské sítě, které je třeba provést, abychom danou řadicí síť získali. Sloupec omezení je nejvyšší počet kroků booleovské sítě, pro který se testuje správnost funkce vytvořené řadicí sítě.

Největší řadicí síť, kterou se podařilo nalézt, má 13 vstupů a obsahuje 66 komparátorů. Podařilo se najít větší řadicí sítě než pomocí kombinačních obvodů, což je způsobeno použitím komparátorů místo hradel. Vidíme, že v žádném experimentu nebylo dosaženo zpoždění, které by odpovídalo údajům z tabulky 6.1. Z tohoto důvodu je u následujících experimentů, kde se snažíme dosáhnout nejlepších hodnot, nastaven počet kroků booleovské sítě na nejlepší známou hodnotu pro daný počet vstupů z tabulky 6.1. Vzhledem k tomu, že nalézt funkční sítě s menším zpožděním je pro evoluční algoritmus těžší, než když je zpoždění větší (viz tabulka 6.2 a výsledky pro 10 vstupů), tak byl zvýšen maximální počet generací na 5 000. Pro každý počet vstupů sítě bylo provedeno 100 běhů evoluce, velikost populace byla opět 1 000.

V tabulce 6.3 jsou uvedeny výsledky těchto experimentů. Největší nalezená řadicí síť, jejíž zpoždění odpovídá tabulce 6.1, má 10 vstupů. Největší síť, u které odpovídá tabulce

6.1 i počet komparátorů, má 8 vstupů. Úspěšnost evoluce se s počtem vstupů postupně snižovala, a síť o jedenácti vstupech se zpožděním 8 se již objevit nepodařilo.

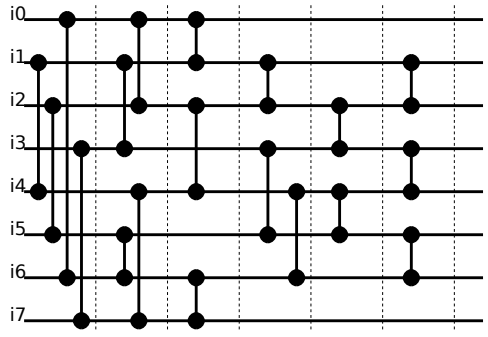
Na obrázku 6.2 v části (a) je schéma sítě s osmi vstupy a v části (b) jsou uvedeny komparátory, z kterých se skládá (čísla označují, které dva vstupy komparátor porovnává). Symbol × označuje, že byl vygenerován komparátor, který měl použít již použité vstupy, a tak byl odstraněn.

počet vstupů	N	K	úspěšnost	počet generací			počet úrovní			
				min	průměr	max	omezení	min	průměr	max
10	10	2	100%	64	225.78	386	20	15	18.74	20
10	10	2	3%	480	732.67	958	9	9	9.00	9
12	12	2	64%	267	613.33	935	16	15	15.95	16
12	12	2	87%	233	520.44	912	20	18	19.63	20
12	14	2	52%	329	600.73	946	16	15	15.94	16
13	13	2	1%	984	984.00	984	13	13	13.00	13

Tabulka 6.2: Výsledky experimentů s různým počtem vstupů, počtem uzlů booleovské sítě N a omezením počtu úrovní (kroků booleovské sítě).

počet vstupů	N	K	počet úrovní	úspěšnost	počet generací			počet komparátorů		
					min	průměr	max	min	průměr	max
5	5	2	5	100%	4	31.80	801	9	9.87	10
6	6	3	5	98%	29	175.65	3743	12	13.31	15
7	7	3	6	71%	105	1070.23	4943	16	17.65	18
8	8	3	6	39%	279	1790.56	4991	19	21.41	23
9	9	3	7	1%	3384	3384.00	3384	27	27.00	27
10	10	4	8	4%	2000	3589.00	4668	35	37.00	40

Tabulka 6.3: Výsledky experimentů, kde byly nalezeny sítě se zpožděním odpovídajícím tabulce 6.1.



(a)

úroveň					
1	2	3	4	5	6
1-4	5-6	2-4	3-5	2-3	5-6
2-5	1-3	×	4-6	×	×
0-6	4-7	6-7	×	4-5	3-4
3-7	0-2	0-1	1-2	×	1-2

(b)

Obrázek 6.2: Řadicí síť s osmi vstupy (a) a vytvořené komparátory (b).

Kapitola 7

Návrh filtru pro odstraňování šumu

Další použití booleovských sítí se zabývá návrhem filtru pro odstraňování impulsního šumu z obrazu. Původní Kauffmanův model je upraven, aby lépe odpovídal reprezentaci obrazu. Také se z určitého úhlu pohledu nejedná o vývojový model, protože navrhovaná síť není použita k navržení filtru, ale přímo k jeho realizaci.

7.1 Impulsní šum a jeho odstraňování

Šum je poškození obrazu, které vzniká různými způsoby v průběhu získávání a přenosu obrazu. Podle [23] je jedním z nejzmiňovanějších právě impulsní šum. Jedním z jeho typů je šum sůl a pepř (salt-and-pepper noise), jemuž se budeme v této práci věnovat. Ten se vyznačuje tím, že poškozené pixely nabývají maxima, nebo minima barevného rozsahu (pro obraz v odstínech šedi vznikají černé, nebo bílé pixely). Obraz může obsahovat různé množství šumu. To bývá vyjádřeno v procentech. Například intenzita šumu 5% znamená, že v obrazu by mělo být 5% poškozených pixelů.

Existují dva typy filtrů. Lineární filtr projde postupně všechny pixely obrazu a vypočte jejich novou hodnotu jako lineární kombinaci okolních pixelů (např. Gaussův filtr, viz [24]). Nevýhodou tohoto přístupu je rozmazání obrazu a tím pádem i ztráta detailů. Druhým typem jsou nelineární filtry. Ty s okolními pixely provádějí jiné operace než lineární kombinaci. V [23] je jako nejoblíbenější uveden mediánový filtr, který je sice vhodný k hardwarové implementaci, ale pro vyšší intenzitu šumu již nedává příliš dobré výsledky.

Počet okolních pixelů, které jsou brány v úvahu při výpočtu nové hodnoty pixelu je jedním ze základních parametrů filtru. Většinou se jedná o čtverec a jeho strana má lichý počet pixelů. Středem je pixel, jehož hodnota je filtrována. Toto „okno“ se postupně posouvá pixel po pixelu obrazem. Okrajové body obrazu jsou ty, jejichž okolí dané velikostí filtru by bylo mimo obraz. V této práci jsou zanedbány.

7.2 Nelineární filtr realizovaný booleovskou sítí

Cílem je navrhnout nelineární obrazový filtr, který bude odstraňovat impulsní šum typu sůl a pepř. Vstupní obraz bude v odstínech šedi s barevnou hloubkou 8 bitů. Pokud bychom chtěli použít Kauffmanův model, tak bychom potřebovali síť o velikosti alespoň:

$$N = \mathsf{velikost\ filtru} \cdot \mathsf{barevná\ hloubka} \tag{7.1}$$

a pro filtr o velikosti pouhých 3×3 pixely nám minimální počet uzlů vychází:

$$N = 3 \cdot 3 \cdot 8 = 72. \tag{7.2}$$

Při předchozích experimentech s kombinačními obvody a řadicími sítěmi se nám nepodařilo nalézt takto velkou booleovskou síť, která by plnila svou funkci, a tak je třeba Kauffmanův model mírně upravit.

Pixely obrazu jsou reprezentovány osmibitovým číslem, a tak budeme uzly sítě reprezentovat stejně. Pro takové uzly již není vhodné použít tabulku, která může obsahovat libovolnou logickou funkci, jak tomu bylo u kombinačních obvodů. Taková tabulka by byla příliš velká.

V [19] je popsána metoda pro návrh obrazového filtru pomocí virtuálního rekonfigurovatelného obvodu. Jsou zde také uvedeny operátory vhodné pro návrh nelineárního filtru a z nich byly vybrány takové, které jsou vhodné pro použití v booleovské síti. Ty jsou uvedeny v tabulce 7.1. Vstupem operátoru jsou hodnoty připojených uzlů. Pokud má operátor pouze jeden vstup, použije se první připojený uzel. V tabulce 7.1 jsou jako příklad uvedeny jen dva vstupy, ale skutečný počet vstupů závisí na počtu připojených uzlů. Výstup je použit jako následující stav uzlu. Vzhledem k této změně je třeba vektor $\vec{f}$ z rovnice 4.4 nahradit vektorem

$$\vec{o} = (o_1, o_2, \ldots, o_N), \tag{7.3}$$

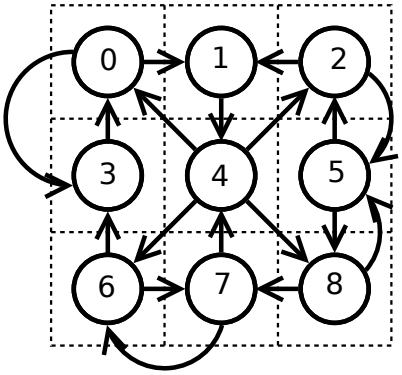
kde $o_1 \ldots o_N$ jsou operátory z tabulky 7.1 přiřazené jednotlivým uzlům.

operátor	význam
255	konstanta
x	identita
$255 - x$	inverze
$x \vee y$	bitový OR
$\bar{x} \vee y$	bitový $\bar{x}$ OR y
$x \wedge y$	bitový AND
$\overline{x \wedge y}$	bitový NAND
$x \oplus y$	bitový XOR
$x \gg 1$	dělení dvěma
$x \gg 2$	dělení čtyřmi
$(x + y)\ mod\ 255$	součet modulo 255
$(x + y) \gg 1$	průměr
$max(x, y)$	maximum
$min(x, y)$	minimum

Tabulka 7.1: Funkce použité při návrhu obrazového filtru.

Evoluční algoritmus hledá ke každému uzlu sítě operátor a připojené uzly. Použito je jednobodové křížení, kterým se kříží vektory $\vec{c}$ a $\vec{o}$. Mutovaný prvek vektoru $\vec{o}$ je nahrazen náhodně zvoleným operátorem.

Počáteční stav uzlů sítě je určen hodnotou odpovídajících pixelů vstupního obrazu. Uzly jsou seskupeny do čtverce, jak je vidět na obrázku 7.1. Čárkovaně jsou pixely, plnou čarou jsou znázorněny uzly a jejich propojení. Z počátečního stavu se provede určitý počet kroků. Pak se odečte hodnota prostředního uzlu a uloží se jako hodnota odpovídajícího pixelu do výstupního obrazu.



Obrázek 7.1: Příklad filtru o velikosti 3×3 pixely ($N = 9$ a $K = 2$).

7.2.1 Fitness funkce

Úkolem fitness funkce je zhodnotit, jak dobře se podařilo odstranit šum z poškozeného obrazu. K tomu potřebujeme původní (nepoškozený) obraz a poškozený obraz (ten obsahuje šum). Na poškozený obraz aplikujeme testovaný filtr a výsledek porovnáme s nepoškozeným obrazem. Fitness funkci spočteme jako součet druhých mocnin rozdílu hodnot pixelů originálu a výsledku:

$$fit = \sum_{i=1}^{size} (O_i - R_i)^2, \tag{7.4}$$

kde *size* je velikost obrazu v pixelech, O_i jsou pixely originálního obrazu a R_i jsou pixely obrazu po odstranění šumu. Fitness funkci se snažíme při použití tohoto kritéria minimalizovat.

7.2.2 Filtr s různým počtem kroků

Počet kroků booleovské sítě provádějící filtrování obrazu nemusí být předem pevně dán. V této práci si klademe za cíl navrhnout filtr, který v každém provedeném kroku zlepšil vlastnosti obrazu. Při použití takového filtru je možno rozhodnout, zda nám stačí méně kvalitní obraz, který bude zpracován rychleji (méně kroků booleovské sítě), nebo požadujeme kvalitnější obraz a kroků provedeme více. Za tímto účelem byla navržena fitness funkce, která tento požadavek zohledňuje:

$$fitness = \sum_{k=1}^{steps} fit(k) + \sum_{k=2}^{steps} (fit(k) - fit(k-1)), \tag{7.5}$$

kde *steps* je maximální počet kroků booleovské sítě a $fit(k)$ je hodnota fitness podle rovnice 7.4 v kroku k. První část výpočtu je součet hodnot $fit(k)$ ze všech kroků. V druhé části se tato hodnota sníží, pokud je fitness v dalším kroku nižší než v předchozím. Postupně se snižující hodnota $fit(k)$ je tedy zvýhodněna.

7.2.3 Trénovací data

Jak již bylo zmíněno v popisu fitness funkce, k návrhu filtru je potřeba nepoškozený obraz a obraz obsahující šum (na ten budeme aplikovat filtr). Na rozdíl od kombinačních

obvodů není možné testovat všechny možné kombinace hodnot na vstupech filtru. V [14] bylo ukázáno, že vhodný obraz o rozměru 128×128 pixelů obsahuje dostatek informací pro ohodnocení filtru.

Pokud jsou filtry ohodnocovány pouze na obrazech obsahujících šum, budou sice odstraňovat šum, ale mohou také poškodit šumem nepoškozené části obrazu. V [19] je doporučeno nechat část trénovacího obrazu nepoškozenou. Poškozený obraz získáme umělým přidáním impulsního šumu typu sůl a pepř do originálního obrazu. Část obrazu je při generování šumu vynechána, aby zůstala nepoškozená. Na obrázku 7.2 je ukázka trénovacích obrazů s různou intenzitou šumu. Pravý dolní roh zůstává nepoškozen.

(a) originál (b) 5% šum (b) 10% šum (b) 30% šum

Obrázek 7.2: Ukázka trénovacích obrazů

7.3 Experimenty

Cílem experimentů je pomocí evoluce navrhnout filtry a porovnat je s mediánovým filtrem. Jako testovací data sloužila sada 24 obrazů z `http://www.fit.vutbr.cz/~vasicek/imagedb/`. Pro porovnání filtrů je stejně jako v [19] použita průměrná hodnota $PSNR$ (peak signal-to-noise ratio, špičková hodnota poměru signálu ku šumu):

$$PSNR = 10log_{10}\frac{255^2}{\frac{1}{PR}\sum_{i,j}(O(i,j) - R(i,j))^2}, \tag{7.6}$$

kde $P \times R$ je velikost obrazu (počet pixelů), O je původní (nepoškozený) obraz a R je výsledek filtrace. Pro porovnáni s navrženými filtry byly použity mediánové filtry s velikostí filtrovacího okna 3×3 pixely a 5×5 pixelů.

7.3.1 Filtr s pevně daným počtem kroků

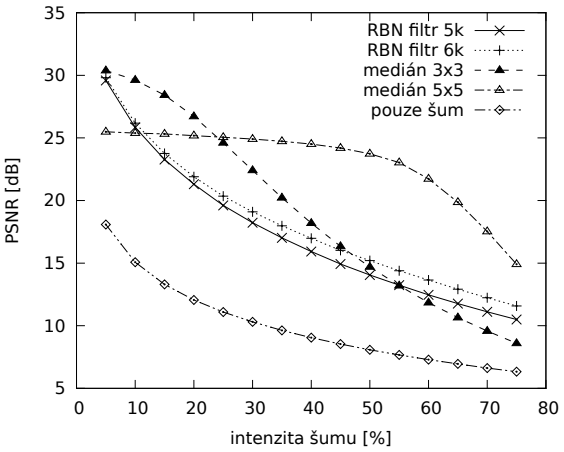
Nejdříve byl proveden experiment s pevně daným počtem kroků. Evoluční algoritmus vybíral ze všech funkcí v tabulce 7.1. Parametry evoluce a booleovské sítě jsou uvedeny v tabulce 7.2. Vzhledem k časové náročnosti experimentu byl proveden jen jeden běh.

Na obrázcích 7.3 je vidět porovnání s mediánovým filtrem. V části (a) je závislost na intenzitě šumu pro jednotlivé typy filtrů (`RBN filtr` je označení pro navržený filtr). Pro srovnání je uvedena také hodnota $PSNR$ pro poškozené obrazy.

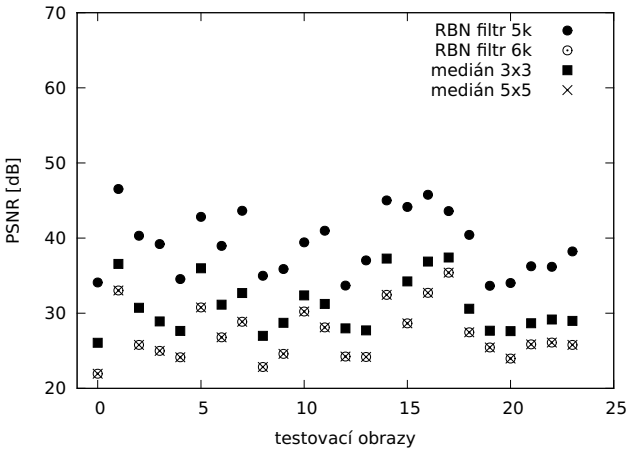
Filtr byl vyvíjen tak, aby pracoval v pěti krocích booleovské sítě (viz 7.2). Pokud se však provede šest kroků, tak je filtr pořád funkční. Na obrázcích 7.3 jsou výsledky filtru po pěti krocích značeny jako `RBN filtr 5k` a po šesti krocích jako `RBN filtr 6k`. Pro nízkou intenzitu šumu výsledky téměř dosahují kvality mediánového filtru 3×3, ale mnohem rychleji se zhoršují. Při intenzitě šumu okolo 45% (pro `RBN filtr 5k`) a 55% (pro `RBN filtr 6k`) jsou výsledky zase srovnatelné a naopak výsledky mediánového filtru klesají rychleji.

Pravděpodobnost mutace	5%
Počet generací	3 000
Velikost populace	1 000
Počet kroků sítě	5
Velikost filtru	3×3
Intenzita šumu	5%
Počet uzlů sítě N	9
Počet propojení K	2

Tabulka 7.2: Parametry evoluce pro `RBN filtr`



(a) Výsledky na poškozených obrazech



(b) Výsledky na originálních obrazech

Obrázek 7.3: Porovnání výsledků filtrů `RBN filtr 5k` a `RBN filtr 6k` s mediánovým filtrem.

	uzel								
	0	1	2	3	4	5	6	7	8
propojení s uzly	7,6	0,6	6,5	2,4	1,7	3,8	4,4	0,6	5,8
funkce v uzlu	255	$(x+y)$	min	max	min	max	$x \wedge y$	$(x+y)$	$x \gg 1$

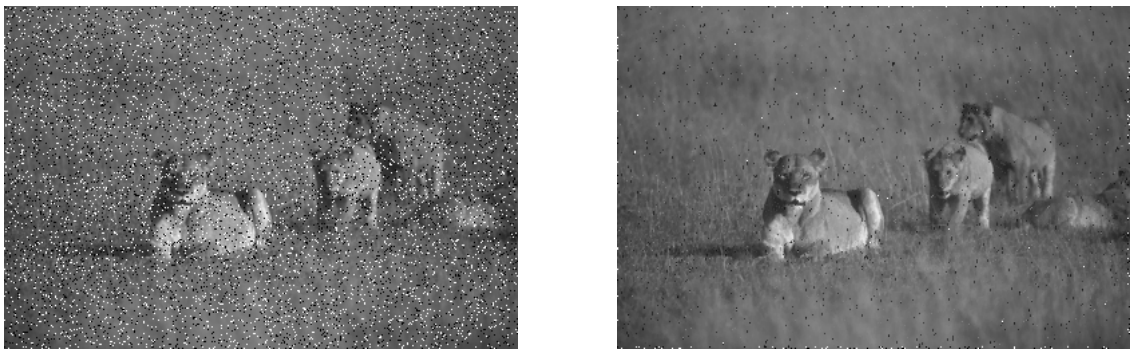
Tabulka 7.3: Propojení a operátory použité ve filtru `RBN filtr`

Mimo odstraňování šumu jsme zkoumali i vliv filtru na nepoškozený obraz. Výsledky pro jednotlivé obrazy z testovací sady jsou vidět na obrázku 7.3 v části (b). Zde je vidět, že `RBN filtr 5k` má ve všech případech nejmenší vliv na obraz bez šumu. Po něm následuje mediánový filtr 3×3. Výsledky pro `RBN filtr 6k` jsou naprosto shodné s mediánovým filtrem 5×5. Aby bylo možno posoudit vizuální kvalitu výstupu, je na obrázku 7.4 ukázka poškozeného obrazu a vedle obraz po aplikaci filtru `RBN filtr 5k`.

V tabulce 7.3 jsou uvedeny funkce a propojení uzlů nalezeného filtru. Po provedení experimentů byla zjištěna chyba v implementaci, která zapříčinila, že výsledek filtru nebyl brán z uzlu s indexem 4 (viz obrázek 7.1), ale z uzlu s indexem 5. Vzhledem k tomu, že se evoluce přizpůsobila, tak experimenty nebyly provedeny znovu. Ukázalo se tím, že

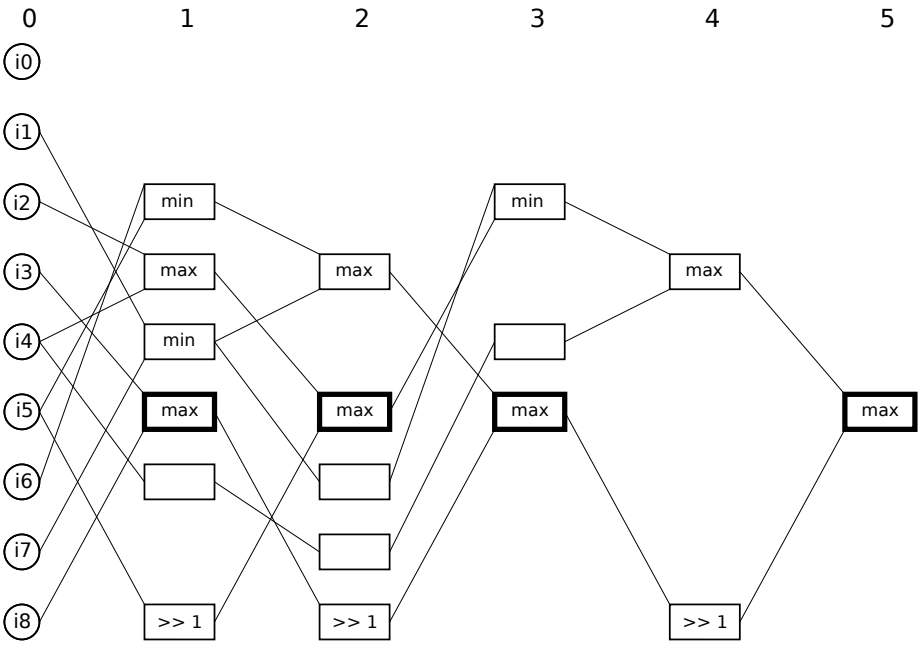
nezáleží na tom, z kterého uzlu budeme výsledek brát, ale musí to být ten, ze kterého byl výsledek brán při vyhodnocování fitness funkce. Evoluce booleovskou síť přizpůsobí. Tato chyba ovlivnila i filtry `RBN filtr L` a `RBN filtr I`, a tak všechny filtry ve zpracovaných experimentech dávají svůj výstup na uzlu s indexem 5.

Na obrázcích 7.5 a 7.6 jsou schémata filtrů, které vznikly zapojením funkcí z tabulky 7.3 za sebe podle připojení uzlů. Funkce, které neměly vliv na výsledek byly odstraněny, a funkce, které pouze kopírovaly svůj vstup, byly nahrazeny prázdným obdélníkem. Tlustou čarou jsou zvýrazněny bloky na indexu, ze kterého je brán výstup filtru. Je vidět, že filtr `RBN filtr 5k` vůbec nevyužívá levý horní pixel ze svého okolí, `RBN filtr 6k` již využívá všechny pixely.

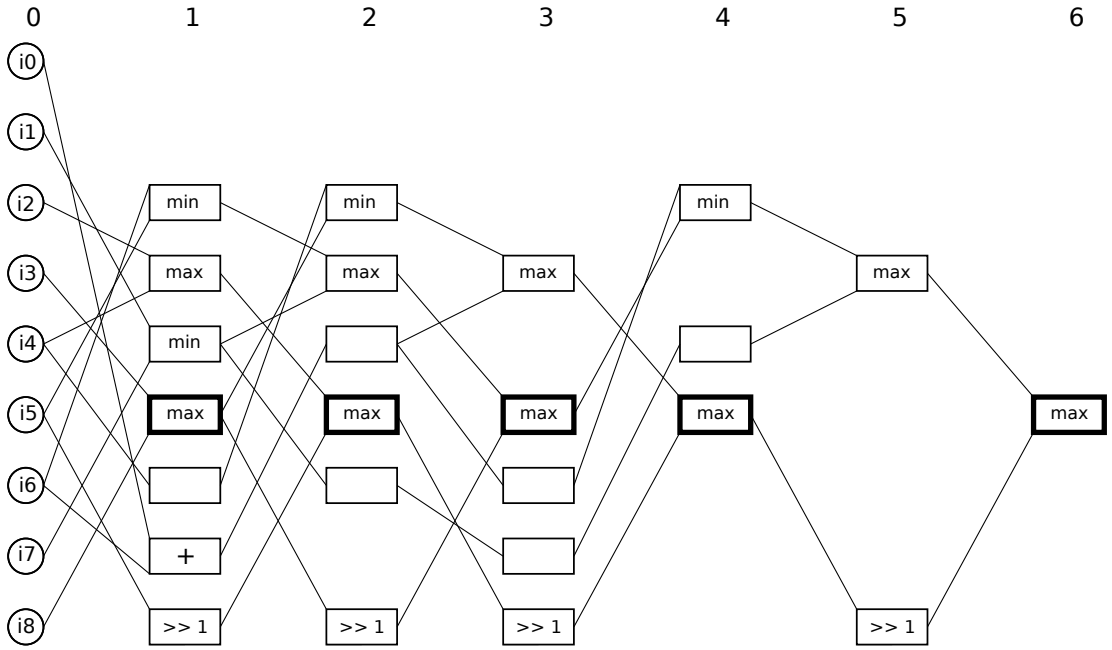


(a) 10% šumu (b) Výstup filtru `RBN filtr 5k`

Obrázek 7.4: Ukázka obrazu před a po aplikaci filtru



Obrázek 7.5: Schéma filtru `RBN filtr 5k`



Obrázek 7.6: Schéma filtru `RBN filtr 6k`

7.3.2 Filtr s pevně daným počtem kroků používající pouze logické funkce

Cílem druhého experimentu bylo navrhnou filtr, který by využíval pouze logické funkce. Proto byly některé funkce z tabulky 7.1 ubrány a některé přidány. Použité funkce jsou v části (a) tabulky 7.5. Parametry evoluce jsou uvedeny v tabulce 7.4.

Pravděpodobnost mutace	5%
Počet generací	3 000
Velikost populace	1 000
Počet kroků sítě	8
Velikost filtru	3×3
Intenzita šumu	10%
Počet uzlů sítě N	9
Počet propojení K	3

Tabulka 7.4: Parametry evoluce pro `RBN filtr L`

V části (b) tabulky 7.5 jsou uvedeny operátory a propojení, které evoluce našla. Z nabízených jedenácti jsou použity pouze čtyři. Tento filtr byl navržen, aby pracoval v osmi krocích booleovské sítě. Na obrázku 7.7 je srovnání s mediánovým filtrem. Je vidět, že filtr `RBN filtr L` značně za mediánovým filtrem zaostává, a to jak v odstraňování šumu, tak i ve vlivu, který má na nepoškozený obraz. I na ukázce výstupu na obrázku 7.8 je zřetelně vidět množství poškozených pixelů. Vzhledem k výsledkům experimentů je pravděpodobné, že pouze logické funkce z části (a) tabulky 7.5 na návrh kvalitního filtru nestačí. Na obrázku 7.8 je vidět srovnání výstupu filtru s poškozeným obrazem.

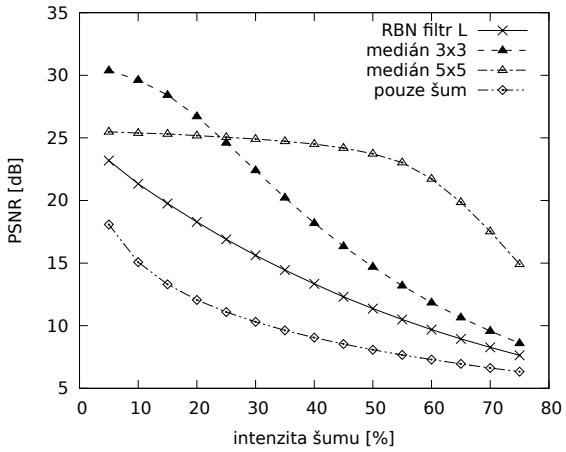
operátor	význam
x	identita
$x \vee y$	bitový OR
$\overline{x} \vee y$	bitový $\overline{x}$ OR y
$\overline{x \vee y}$	bitový NOR
$x \wedge y$	bitový AND
$\overline{x} \wedge y$	bitový $\overline{x}$ AND y
$\overline{x \wedge y}$	bitový NAND
$x \oplus y$	bitový XOR
$\overline{x} \oplus y$	bitový $\overline{x}$ XOR y
$\overline{x \oplus y}$	bitový NXOR
$\overline{x}$	negace

(a) Operátory použité při návrhu

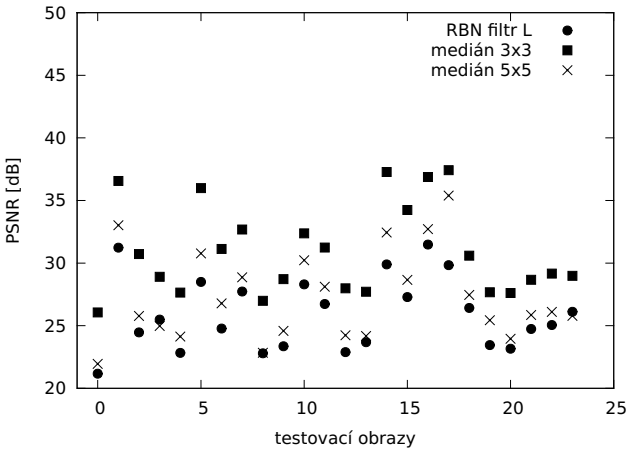
uzly	propojení	funkce
0	6,5,3	$\overline{x \wedge y \wedge z}$
1	2,6,8	$\overline{x \wedge y \wedge z}$
2	8,8,6	$\overline{x \vee y \vee z}$
3	3,2,3	$\overline{x} \oplus y \oplus z$
4	7,1,7	$\overline{x \vee y \vee z}$
5	0,2,5	$\overline{x \wedge y \wedge z}$
6	1,1,7	$\overline{x \wedge y \wedge z}$
7	4,7,1	$\overline{x}$
8	0,2,2	$\overline{x \vee y \vee z}$

(b) Operátory a propojení použité ve výsledném filtru `RBN filtr L`

Tabulka 7.5: Návrh filtru `RBN filtr L`.



(a) Výsledky na poškozených obrazech



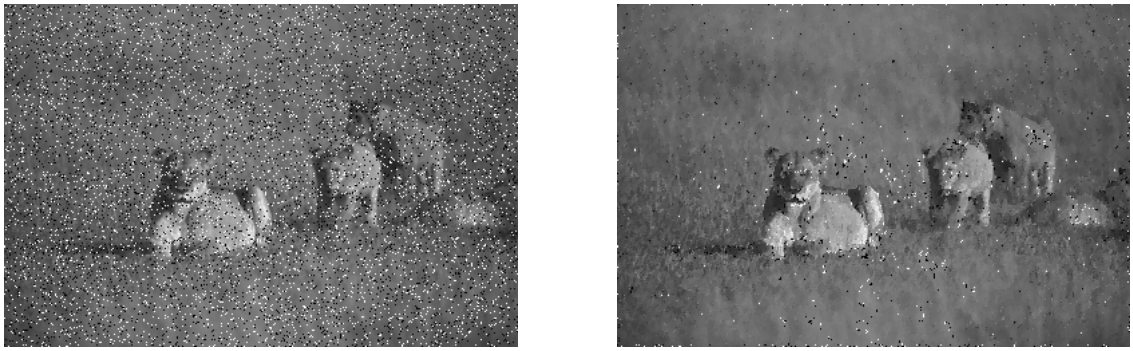
(b) Výsledky na originálních obrazech

Obrázek 7.7: Porovnání výsledků filtru používajícího pouze logické (`RBN filtr L`) funkce s mediánovým filtrem.

7.3.3 Filtr s nastavitelným počtem kroků

Tento experiment ukáže výsledky filtru, který s rostoucím počtem kroků postupně zlepšuje svůj výsledek. Fitness funkce, která to zařídila, je popsána v podkapitole 7.2.2. V tabulce 7.6 opět vidíme parametry evoluce. Jednotlivé kandidátní filtry byly v průběhu evoluce ohodnocovány v každém kroku od 1 po 9 a pak byla spočtena fitness funkce dle rovnice 7.5 (parametr $steps = 9$).

V tabulce 7.7 jsou uvedeny průměrné hodnoty PSNR pro jednotlivé kroky navrženého filtru a různou intenzitu šumu. Z tabulky je vidět, že kvalita filtrovaného obrazu se postupně zvyšuje. Největší skok je mezi prvním a druhým krokem, další výrazné zlepšení je mezi třetím a čtvrtým krokem, pak už se výsledky zlepšují jen mírně a mezi osmým a devátým krokem již není žádný rozdíl.



(a) 10% šumu (b) Výstup filtru `RBN filtr L`

Obrázek 7.8: Ukázka obrazu před a po aplikaci filtru

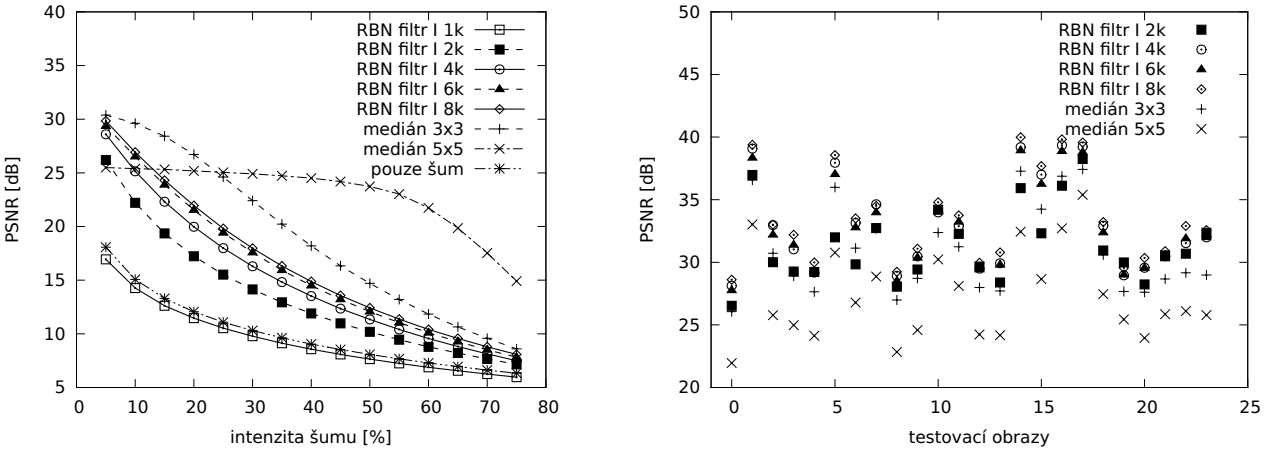
Pravděpodobnost mutace	5%
Počet generací	3 000
Velikost populace	1 000
Počet kroků sítě	1 – 9
Velikost filtru	3×3
Intenzita šumu	10%
Počet uzlů sítě N	9
Počet propojení K	2

Tabulka 7.6: Parametry evoluce pro filtr `RBN filtr I`

intenzita	krok								
šumu	1	2	3	4	5	6	7	8	9
5%	16.94	26.20	26.38	28.60	29.04	29.36	29.84	29.84	29.84
10%	14.27	22.21	22.75	25.14	26.28	26.52	26.92	26.92	26.92
15%	12.62	19.36	20.03	22.31	23.68	23.89	24.30	24.31	24.31
20%	11.46	17.23	17.90	19.97	21.33	21.54	21.94	21.95	21.95
25%	10.53	15.52	16.16	17.99	19.21	19.42	19.81	19.82	19.82
30%	9.78	14.14	14.71	16.30	17.38	17.58	17.97	17.98	17.98
35%	9.13	12.93	13.44	14.82	15.72	15.93	16.31	16.33	16.33
40%	8.57	11.90	12.35	13.52	14.28	14.50	14.86	14.87	14.87
45%	8.08	10.98	11.37	12.35	13.00	13.21	13.56	13.57	13.57
50%	7.64	10.19	10.50	11.34	11.87	12.07	12.40	12.42	12.42
55%	7.25	9.45	9.72	10.41	10.85	11.05	11.36	11.37	11.37
60%	6.88	8.79	9.00	9.56	9.90	10.10	10.40	10.41	10.41
65%	6.56	8.21	8.37	8.82	9.09	9.28	9.56	9.57	9.57
70%	6.25	7.65	7.77	8.13	8.33	8.51	8.78	8.79	8.79
75%	5.96	7.16	7.23	7.50	7.64	7.82	8.07	8.08	8.08

Tabulka 7.7: Průměrná hodnota PSNR pro jednotlivé kroky filtru a různou intenzitu šumu u filtru `RBN filtr I`

V grafech na obrázku 7.9 je opět srovnání s mediánovým filtrem. Kvůli přehlednosti jsou zde vyneseny jen hodnoty pro vybrané kroky. Z části (a) je vidět, že pro šum o intenzitě 5% se výsledky `RBN filtr I 8k` blíží mediánovému filtru 3×3, ale s rostoucí intenzitou klesají rychleji. Stejně jako v tabulce 7.3 i zde je zřetelné, jak se výsledky filtru postupně zlepšují. Výsledky po prvním kroku jsou horší než pro obraz, na který nebyl aplikován žádný filtr (první krok kvalitu obrazu sníží), ale od druhého kroku se kvalita obrazu postupně zvyšuje. Výsledky na nepoškozených obrazech však počtu kroků již příliš neodpovídají. `RBN filtr I 8k` sice ve většině případů poškodí obraz nejméně, ale `RBN filtr I 4k` má v mnoha případech lepší výsledek než `RBN filtr I 6k`. `RBN filtr I 8k`, `RBN filtr I 6k` i `RBN filtr I 4k` poškodí obraz méně než mediánový filtr.



(a) Výsledky na poškozených obrazech

(b) Výsledky na originálních obrazech

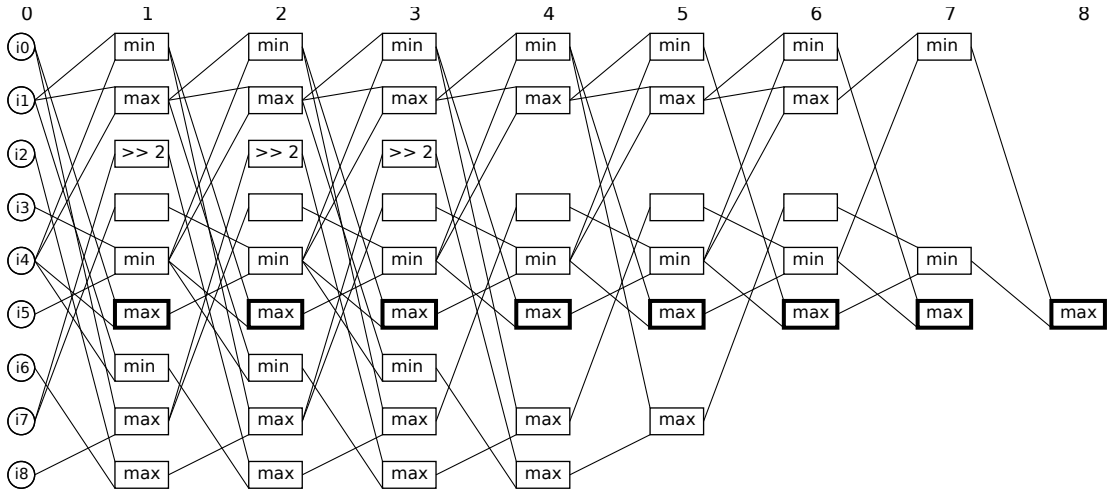
Obrázek 7.9: Porovnání výsledků postupně se zlepšujícího filtru a mediánového filtru.

V tabulce 7.3 jsou uvedeny operátory, které jsou ve filtru použity a jejich propojení. Evoluce použila jen 4 operátory a to minimum, maximum, dělení čtyřmi a identitu prvního vstupu. Na obrázku 7.10 je uvedeno schéma tohoto filtru, tlustou čarou jsou znázorněny bloky, ze kterých je v jednotlivých krocích brán výstup filtru. Opět jsou odstraněny bloky, které nemají na výstup filtru žádný vliv.

	uzly								
	0	1	2	3	4	5	6	7	8
propojení	4,1	4,1	7,0	7,8	3,5	0,4	1,4	0,8	6,2
funkce	min	max	$x \gg 2$	x	min	max	min	max	max

Tabulka 7.8: Operátory a propojení použité ve výsledném filtru `RBN filtr I`

Na obrázku 7.12 jsou výsledky filtru po různém počtu kroků na jednom z obrazů testovací sady. Je vidět to co bylo popsáno výše. Výsledek po prvním kroku vypadá subjektivně hůře než zašuměný obraz a dále se výsledek postupně zlepšuje. Mezi částí (e) a (f) již není žádný zřetelnější rozdíl.

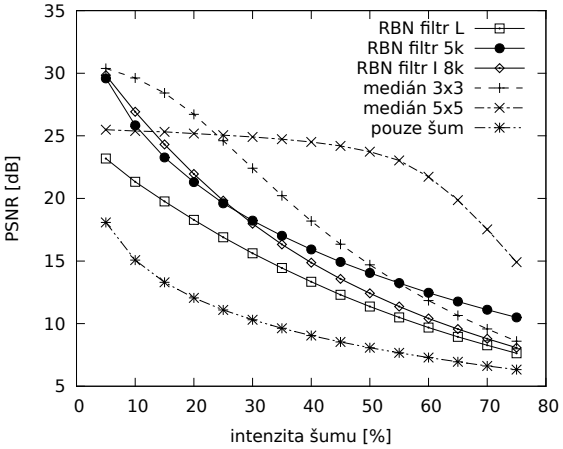


Obrázek 7.10: Schéma filtru `RBN filtr I`

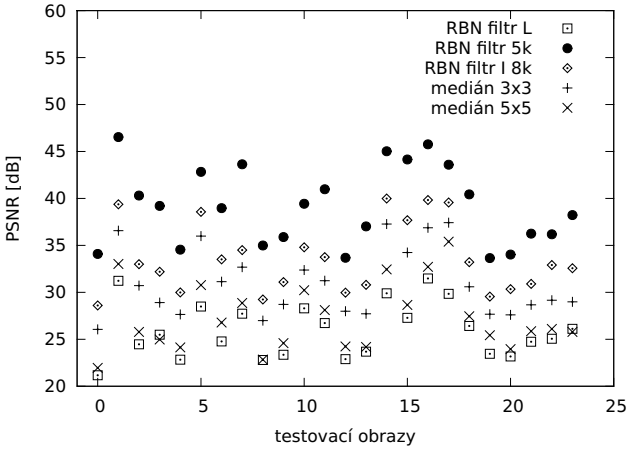
7.3.4 Celkové srovnání

Na závěr srovnáme všechny námi nalezené typy filtrů. Na obrázku 7.11 je srovnání našich třech filtrů a mediánového filtru. `RBN filtr I 8k` (filtr z podkapitoly 7.3.3 po provedení osmi kroků) je až do intenzity šumu 25% mírně lepší než obyčejný `RBN filtr`, ale pak se zhoršuje rychleji a blíží se k filtru `RBN filtr I 8k`, který používá jen logické funkce.

Využití navržených filtru je možné především u intenzity šumu okolo 5%, kde kvalita výstupu odpovídá mediánovému filtru. Vzhledem k tomu, že všechny úrovně filtru používají stejné bloky i jejich propojení k předchozí úrovni (pokud nejsou odstraněny jako nevyužité), tak lze lze filtr implementovat i pouze pomocí jedné úrovně těchto bloků a po určitém počtu kroků odebrat výstup. Nejsme však omezeni na jednu, nebo plný počet úrovní, ale je možno implementovat tolik úrovní, kolik odpovídá dostupnému místu na čipu, a tím zvolit nejvhodnější poměr cena/výkon.



(a) Výsledky na poškozených obrazech



(b) Výsledky na originálních obrazech

Obrázek 7.11: Porovnání všech typů nalezených filtrů a mediánového filtru.

(a) 10% šumu

(b) 1 krok

(c) 2 kroky

(d) 4 kroky

(e) 6 kroků

(f) 8 kroků

Obrázek 7.12: Ukázka obrazu před a po aplikaci různého počtu kroků filtru `RBN filtr I`

Kapitola 8

Závěr

Tato práce se zabývá využitím booleovských sítí v evolučním návrhu. Byly zde uvedeny možnosti jak booleovské sítě reprezentovat a byly navrženy genetické operátory, jež lze na tuto reprezentaci použít. Navržená reprezentace byla nejdříve experimentálně ověřena při návrhu kombinačních obvodů, poté byla mírně upravena pro návrh řadicích sítí. Nakonec byla provedena výraznější úprava a ta byla použita pro návrh obrazového filtru.

Metoda pro návrh kombinačních obvodů z kapitoly 5 se ukázala funkční. Bohužel se podařilo navrhnout pouze obvody s nízkým počtem vstupů. To by bylo možné napravit podrobnějším průzkumem a úpravou evolučního algoritmu. Neúspěch na větších obvodech může být způsoben také vlastnostmi booleovských sítí.

V [5] je uvedeno, že sítě s $K \leq 2$ mají tendenci konvergovat do jednoho ustáleného stavu, nebo opakujícího se cyklu. K návrhu menších obvodů stačí menší počet kroků booleovské sítě, a tak je obvod navržen dříve, než se síť stihne ustálit na jedné hodnotě, ale u větších obvodů by to mohl být problém. Se vzrůstajícím K se chování sítě stává více chaotické, bohužel pro větší K existuje i větší počet všech možných booleovských sítí a tím se zvětší prohledávaný prostor (viz rovnice 3.1). To platí i pro řadicí sítě na úrovni komparátorů. Zde však konvergence sítě neměla pravděpodobně takový vliv, protože řadicí sítě mohou mít poměrně pravidelnou strukturu.

Řešením by mohlo být použití většího K a nějakým způsobem omezit vzniklé logické funkce. V průběhu implementace navrhovaných metod proběhlo několik pokusů s uniformními sítěmi (všechny uzly obsahují stejnou logickou funkci), ale ty nebyly příliš úspěšné. Pokud by se však uzly rozdělily do několika skupin, kde uzly v jedné skupině sdílejí jedinou logickou funkci, mohlo by dojít k omezení prohledávaného prostoru a zároveň zachování dostatečné variability. Možností je dostatek a tím také prostor pro další výzkum.

Navržené obrazové filtry mají různé zajímavé vlastnosti. Pomocí vhodného návrhu fitness funkce lze objevit filtr s požadovanými vlastnostmi, jako bylo například postupné zlepšování kvality výstupu z kapitoly 7. Navržené filtry však nejsou příliš kvalitní a bylo by třeba provést více experimentů a případně upravit evoluční algoritmus, aby se našel kvalitnější filtr.

V [19] je zmíněn princip tzv. banky filtrů, kdy je evolučně navrženo více filtrů a z jejich výsledků je vybrán konečný výsledek například pomocí mediánu. Při návrhu filtrů se ukázalo, že nezáleží na tom, který z uzlů si zvolíme jako výstup. Nabízí se řešení vyvinout banku filtrů jako jediný obvod tak, že budeme brát výstup např. ze tří uzlů a pak pomocí mediánu vybrat konečný výsledek. Dále by se dalo zaměřit na co nejjednodušeji hardwarově realizovatelné funkce a z nich se pokusit navrhnout funkční filtr. Je ovšem potřeba použít i jiné než jen logické funkce, protože ty se v experimentech v kapitole 7 příliš neosvědčily.

Literatura

[1] Bidlo, M.: *Evoluční návrh řadicího algoritmu*. Diplomová práce, FIT VUT v Brně, 2004.

[2] Bidlo, M.: *Evolutionary Design of Generic Structures Using Instruction-Based Development*. Dizertační práce, 2009.

[3] Bidlo, M.; Vašíček, Z.: Gate-Level Evolutionary Development Using Cellular Automata. In *AHS '08: Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, Washington, DC, USA: IEEE Computer Society, 2008, ISBN 978-0-7695-3166-3, s. 11–18.

[4] Darwin, C.: *On the Origin of Species By Means of Natural Selection*. Dover Publications, 2006.

[5] Gershenson, C.: Introduction to Random Boolean Networks. In *Workshop and Tutorial Proceedings, ALife IX*, 2004, s. 160–173.

[6] Gordon, T. G. W.; Bentley, P. J.: Towards Development in Evolvable Hardware. In *EH '02: Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)*, Washington, DC, USA: IEEE Computer Society, 2002, ISBN 0-7695-1718-8, str. 241.

[7] Haddow, P. C.; Tufte, G.; Remortel, P. v.: Shrinking the Genotype: L-systems for EHW? In *ICES '01: Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, London, UK: Springer-Verlag, 2001, ISBN 3-540-42671-X, s. 128–139.

[8] Holland, J. H.: *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0262581116.

[9] Kauffman, S. A.: Metabolic stability and epigenesis in randomly constructed genetic nets. In *Journal of Theoretical Biology, 22*, 1969, s. 437–467.

[10] Knuth, D. E.: *The Art of Computer Programming. Volume 3: Sorting and Searching*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998, ISBN 0201485419.

[11] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992, ISBN 0262111705.

[12] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. STU Bratislava, 2000, ISBN 8022713775.

[13] Lindenmayer, A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, ročník 18, č. 3, March 1968: s. 300–315, doi:10.1016/0022-5193(68)90080-5.

[14] Martínek, T.; Sekanina, L.: An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA. In *Evolvable Systems: From Biology to Hardware*, LNCS 3637, Springer Verlag, 2005, ISBN 978-3-540-28736-0, s. 76–85.

[15] Neary, T.; Woods, D.: P-completeness of cellular automaton Rule 110. In *In International Colloquium on Automata Languages and Programming (ICALP), volume 4051 of LNCS*, Springer, 2006, s. 132–143.

[16] Rechenberg, I.: *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[17] Schwefel, H.-P.: *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. Diplomarbeit, Technische Universität Berlin, Hermann Föttinger–Institut für Strömungstechnik, März 1965.

[18] Sekanina, L.; Bidlo, M.: Evolutionary Design of Arbitrarily Large Sorting Networks Using Development. *Genetic Programming and Evolvable Machines*, ročník 6, č. 3, 2005: s. 319–347, ISSN 1389-2576.

[19] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.

[20] Sipper, M.: *Evolution of Parallel Cellular Machines, The Cellular Programming Approach*, *Lecture Notes in Computer Science*, ročník 1194. Springer, 1997, ISBN 3-540-62613-1.

[21] Thompson, A.: *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*. Distinguished dissertation series, Springer-Verlag, 1998.

[22] Vašíček, Z.; Sekanina, L.: Evoluční návrh kombinačních obvodů. *Elektrorevue - www.elektrorevue.cz*, ročník 2004, č. 43, 2004: s. 1–6, ISSN 1213-1539.

[23] Vašíček, Z.; Sekanina, L.: Novel Hardware Implementation of Adaptive Median Filters. In *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, IEEE Computer Society, 2008, ISBN 978-1-4244-2276-0, s. 110–115.

[24] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2005, ISBN 80-251-0454-0.