VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

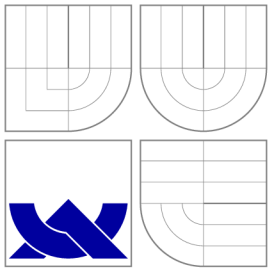
CELULÁRNÍ AUTOMAT V EVOLUČNÍM PROCESU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

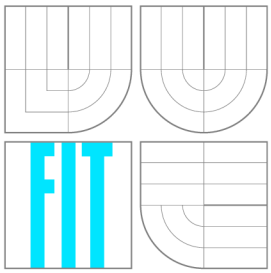
AUTOR PRÁCE
AUTHOR

Bc. MICHAL HEJČ

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

CELULÁRNÍ AUTOMAT V EVOLUČNÍM PROCESU

CELLULAR AUTOMATON IN EVOLUTIONARY PROCESS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL HEJČ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL BIDLO
SUPERVISOR

BRNO 2007

Abstrakt

Tato diplomová práce pojednává o využití evolučních algoritmů společně s technikou developmentu v celulárních automatech. Popisuje základní principy jednotlivých nástrojů a následně se zaměřuje na jednu specifickou oblast - návrh kombinačních logických obvodů. Pomocí genetického algoritmu je hledán neuniformní celulární automat, který slouží jako generátor výsledného obvodu. Jsou provedeny experimenty se základními typy kombinačních logických obvodů a se speciální třídou nazývanou polymorfní obvody. Na závěr jsou představeny dosažené výsledky a provedeno porovnání s uniformními celulárními automaty.

Klíčová slova

Genetický algoritmus, celulární automat, development, evoluční návrh, kombinační logický obvod, polymorfní obvod

Abstract

The aim of this master's theses it to focuse on the usage of genetic algorithms in combination with a technique of biologically inspired development in cellular automata. The principles of the proposed method is described. The main part of this work deals with the design of combinational logic circuits. The genetic algorithm is utilized to design a nonuniform one-dimensional cellular automaton (in particular, the local transition functions) which serves as a circuit generator. Experiments have been conducted to design of basic types of combinational circuits and polymorphic circuits. Finally, the results are presented and compared with the results obtained in the previous work in which a uniform cellular automaton was applied.

Keywords

Genetic algorithm, cellular automaton, development, evolutionary design, combinational logic circuit, polymorph circuit

Citace

Michal Hejč: Celulární automat v evolučním procesu, diplomová práce, Brno, FIT VUT v Brně, 2007

Celulární automat v evolučním procesu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla.

.....
Michal Hejč
22. května 2007

Obsah

1 Úvod **3**

2 Evoluční algoritmy - přehled technik **4**
2.1 Genetický algoritmus 5
2.2 Genetické programování 7
2.2.1 Symbolická regrese 10
2.2.2 Gramatická evoluce 11
2.3 Evoluční strategie 11
2.4 Evoluční programování 13

3 Celulární automaty **14**
3.1 Binární celulární automaty 15
3.2 Hybridní - neuniformní celulární automaty 16

4 Development **18**

5 Aplikace celulárních automatů jako vývojových modelů **20**
5.1 Návrh kombinačních logických obvodů 20
5.2 Příklad konstrukce kobminačního logického obvodu 21
5.3 Polymorfní logické obvody 24

6 Evoluční algoritmus **26**
6.1 Genetický algoritmus 26

7 Počáteční nastavení parametrů algoritmu **31**
7.1 Velikost populace 31
7.2 Pravděpodobnost mutace přechodových funkcí 32
7.3 Počet jedinců v turnajové selekci 33
7.4 Pravděpodobnost mutace počátečního stavu 34
7.5 Počet přepisovacích pravidel lokálních přechodových funkcí 34

8 Experimentální výsledky **39**
8.1 Sčítačky 40
8.2 Násobičky 41
8.3 Boolovská symetrie 42
8.4 Řadicí sítě 43
8.5 Mediánové obvody 44
8.6 Paritní obvody 45
8.7 Polymorfní obvody 46

9 Diskuze **55**

10 Závěr **56**

A Projektová dokumentace **59**
A.1 Popis implementovaných tříd 59
A.2 Překlad programu 61

B Programová dokumentace **63**

Kapitola 1

Úvod

I v dobách značného pokroku v oblasti počítačového inženýrství nejsme schopni v řadě domén lidského bádání nalézt optimální nebo alespoň uspokojující řešení. Jedná se zejména o výpočetní problémy, kde stavový prostor nabývá značných rozměrů. Výpočetní čas potřebný k prozkoumání celého takového prostoru je vzhledem k lidským potřebám neúměrně velký. V těchto případech jsou používány alternativní přístupy výpočtu, které svojí náročností dosahují řádově nižších hodnot a dokáží podat uspokojující řešení. Jednou z těchto technik jsou biologií inspirované algoritmy.

V rámci tohoto projektu se seznámíme s vybranými technikami evolučních algoritmů a celulárních automatů. Jako společný prvek popíšeme využití developmentu založeného na celulárních automatech ve spojení s evolučními algoritmy pro návrh kombinačních logických obvodů.

Text je členěn do následujících částí: Kapitola 2 popisuje problematiku evolučních algoritmů, jejich principy a použití. Dále jsou vyjmenovány nejpoužívanější techniky z tohoto oboru a jejich základní vlastnosti. Kapitola 3 se zabývá základními principy matematického modelu celulárních automatů. Kapitola 4 je zaměřena na vysvětlení techniky developmentu a jejího využití v evolučních algoritmech. Kapitola 5 uvádí oblasti vhodné k aplikaci celulárních automatů jako vývojových modelů a příklad konstrukce výsledného objektu pomocí techniky developmentu. V kapitole 6 je zvolen vhodný evoluční algoritmus pro řešení vybraných oblastí. Dále jsou popsány základní parametry zvoleného algoritmu. Kapitola 7 popisuje prvotní testování algoritmu a stanovení doporučených hodnot pro jeho jednotlivé parametry. V kapitole 8 jsou prezentovány experimentální výsledky. Kapitola 9 je diskuzí nad výsledky experimentů a použitého řešení. Závěrečná kapitola 10 je shrnutím.

Kapitola 2

Evoluční algoritmy - přehled technik

Podobně jako v řadě jiných oborů, začínají se v oblasti IT uplatňovat techniky, které jsou inspirovány přirozenými principy fungování a vývoje tohoto světa. Do této kategorie lze mimo jiné zařadit evoluční algoritmy [5, 3], které vycházejí ze základních pravidel vývoje (evoluce) biologických organismů (dnes již můžeme najít i evoluční algoritmy, které nejsou založeny na těchto principech a hledají inspiraci ve fyzice, např. simulované žíhání [3]).

Jádrem evolučních algoritmů se stala Darwinova evoluční teorie, která je založena na principu přirozeného výběru, podle kterého přežijí v dané populaci jen nejlépe přizpůsobení jedinci. Reprodukcí takových jedinců vzniká poměrně velká šance na vznik nového jedince s lepšími vlastnostmi, než kterými disponovali jeho rodiče.

Základním principem většiny evolučních algoritmů je populace tvořená jedinci (chromozomy), kteří obsahují genetické informace. Jedinec představuje jedno z mnoha řešení, kterému je přiřazeno určité kvalitativní ohodnocení (tzv. fitness hodnota), schopnost řešit danou úlohu. U jedinců s lepším ohodnocením je pak větší pravděpodobnost, že vstoupí do procesu reprodukce s jiným jedincem. Vznikne tak potomek, u kterého lze předpokládat, že bude daná kritéria splňovat lépe než jeho rodiče. Pro výpočet fitness hodnot je využíváno tzv. fitness funkce [3], která je jádrem daného evolučního algoritmu a pro každou úlohu je specifikována individuálně. Obvykle je hodnota fitness kladné reálné číslo, kde nula udává ideální řešení. Definujme některé pojmy:

Účelová funkce: Popisuje vztah mezi genotypem a fenotypem. Udává míru úspěšnosti chromozomu.

Fitness: Relativní schopnost přežití a reprodukce genotypu v daném prostředí.

Přirozený výběr: Proces, ve kterém jedinci s větší fitness vstupují do reprodukčního procesu s větší pravděpodobností než jedinci s menší fitness.

Náhodný genetický drift: Náhodné události zasahující do vývoje jedinců a ovlivňující vývoj populace. Jedná se např. o mutaci genetické informace nebo náhodné úmrtí jedince s velkým fitness ještě před tím, než měl možnost vstoupit do reprodukčního procesu.

Reprodukční proces: Reprodukce je proces, při kterém se z vybraných jedinců (rodičů) vytváří potomci.

Evoluční algoritmy patří v současné době mezi základní optimalizační algoritmy moderní informatiky. Jejich aplikace však můžeme zaznamenat i v jiných oblastech. Uplatnění nacházejí zejména tam, kde je nemyslitelné prohledávání celého stavového prostoru. Svoji obecností jsou použitelné např. téměř ve všech případech, kdy můžeme dané řešení ohodnotit pomocí jednoduchého rychlého výpočtu.

Je nutno poznamenat, že se jedná o stochastické optimalizační algoritmy, avšak od klasického slepého stochastického algoritmu se významně odlišují. Slepý stochastický algoritmus prohledává stavový prostor zcela náhodně. Nové řešení nemá žádnou návaznost na optimalizovanou funkci a v případech, kdy stavový prostor nabývá velkých rozměrů, je nalezení přijatelného řešení časově velmi náročné. Naproti tomu u evolučních algoritmů jsou ke stochastickým prvkům přiřazeny techniky, které dokáží usměrnit náhodné generování bodů k hodnotám, které mají přijatelnou vzdálenost od optimálního řešení. Dalším významným faktorem evolučních algoritmů je skutečnost, že v jednom kroku nepracují v rámci stavového prostoru pouze s jediným řešením, nýbrž s celou množinou (populací) řešení.

Mezi základní evoluční algoritmy patří genetický algoritmus, genetické programování, evoluční strategie a evoluční programování.

2.1 Genetický algoritmus

Autorem Genetického algoritmu se stal v polovině 70-tých let minulého století John Holland [9]. V současné době patří genetický algoritmus mezi nejpoužívanější a snad nejznámější evoluční algoritmus.

Genetický algoritmus pracuje se dvěma prostory: vyhledávací prostor a prostor řešení. Vyhledávací prostor je prostor binárně zakódovaných řešení dané problematiky (genotypů), které jsou mapovány do prostoru řešení. Ten obsahuje jednotlivé entity dané problémové domény (fenotypy).

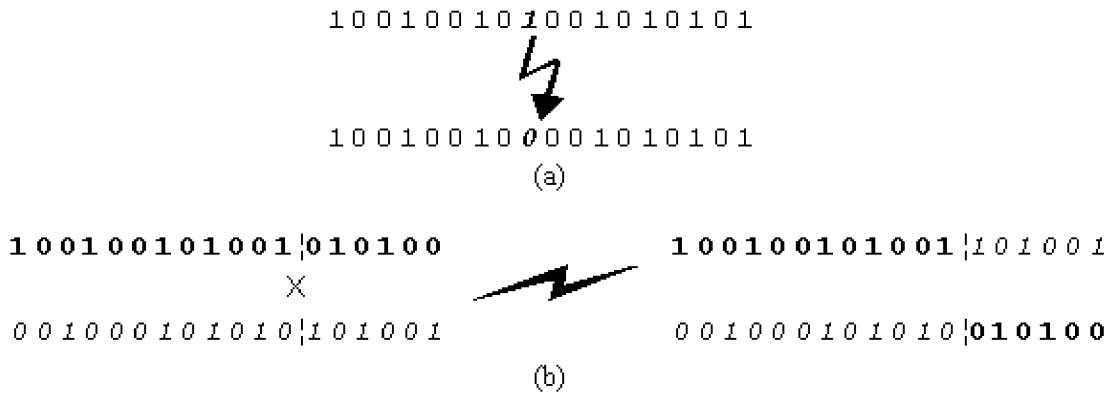
V rámci genetického algoritmu můžeme chápat biologického jedince jako lineární řetězec symbolů fixní délky, který charakterizuje daný chromozom. Potom hovoříme o populaci chromozomů, které se s určitou pravděpodobností mezi sebou reprodukují v závislosti na jejich hodnotě fitness.

Specifickou vlastností genetického algoritmu je využívání operátoru křížení a mutace. Tato skutečnost odlišuje genetický algoritmus od většiny ostatních evolučních algoritmů. Pokud bychom operátor křížení z genetického algoritmu vyjmuli, dostali bychom algoritmus velmi podobný horolezeckému algoritmu [3].

Operátor mutace vnáší do populace chromozomů stochastický prvek, který může zabránit degradaci celé množiny a umožnit tak vznik jedince s vyšší fitness než jeho předchůdce. Mutace je založena na náhodné změně chromozomu. Ve stávajícím kódu genotypu jsou s určitou pravděpodobností změněny některé části. Tímto postupem je zajištěna rozmanitost chromozomů, která zabraňuje jejich případné konvergenci k jednomu jedinému řešení, které nemusí být pro daný účel žádoucí.

Operátor křížení je založen na kombinaci dvou chromozomů zvaných rodiče. V kódech chromozomů jsou náhodně zvoleny body křížení vymezující úseky kódu, které budou navzájem vyměněny. Křížení se nazývá n-bodové, kde n udává počet bodů křížení. Po provedení této operace získáváme dva nové chromozomy (potomky), které získaly určité charakteristické vlastnosti svých rodičovských chromozomů.

Příklad použití obou operátorů znázorňuje obrázek 2.1.



Obrázek 2.1: Genetické operátory: (a) příklad mutace 9. genu, (b) příklad jednobodového křížení na pozici 12

Chromozomy nacházející se v dané populaci vstupují do reprodukčního procesu, ve kterém vznikají noví jedinci (chromozomy). S určitou pravděpodobností je u těchto nových chromozomů možnost, že budou mít větší fitness než jejich rodiče a nahradí tak v populaci chromozomy s nízkou fitness. Tento proces se opakuje do doby, než je nalezeno přijatelné řešení nebo dosaženo maximálního počtu iterací.

Chromozomy jsou do reprodukčního procesu vybírány kvazináhodně. Pokud bychom vybírali pouze chromozomy s největší fitness, mohlo by dojít k ohraničení oblasti, ve které řešení hledáme, což by v konečném důsledku vedlo k nalezení horšího řešení, než by bylo za normálních okolností možné.

Pro renormalizaci fitness se používá několika nejčastějších metod selekce [4]:

Selekce úměrná kvalitě: V tomto případě nedochází ke transformaci účelové funkce. Tato metoda obsahuje několik zásadních omezení a nedostatků. V prvé řadě musí být účelová funkce nezáporná. Dále, pokud populace obsahuje několik jedinců s velmi dobrým ohodnocením, dochází ke značnému vlivu těchto "superjedinců" a k rychlé degradaci populace vlivem velkého selekčního tlaku. A v neposlední řadě, může dojít k úplnému potlačení selekce a to v těch případech, kdy populace pokrývá jen velmi malou část rozsahu účelové funkce.

Metoda okna: Dynamicky upravuje hodnotu účelové funkce. V populaci je nalezen nejhorší jedinec, jehož ohodnocení je následně odečteno od ohodnocení všech ostatních jedinců. Tímto krokem získá nejhorší jedinec nulové ohodnocení a je tak zcela vyloučen z procesu reprodukce. Tento postup však vede k rychlé eliminaci jedinců s nízkým ohodnocením a ztrátě různorodosti populace.

Sigma-selekce: Je založena na podobném principu jako metoda okna. Nedochází však ke snižování fitness v závislosti na nejhorším jedinci. Hodnota, o kterou je fitness upravena, se přímo vztahuje k průměrnému ohodnocení populace. Nedochází tak k rapidní eliminaci špatných jedinců a zároveň je zabráněno stagnaci prohledávacího postupu.

Pořadová selekce: Nachází uplatnění tam, kde lze poměrně obtížně stanovit ohodnocení jedinců, avšak lze nalézt jejich uspořádání. Jedinci jsou pak ohodnoceni sestupně právě v závislosti na jejich pořadí dle kvality. Mezi nejznámější pořadové selekce patří selekce lineární a exponenciální.

Turnajová selekce: Tato metoda obsahuje jak fázi selekce, tak vzorkování. Z populace je vybráno `n` náhodných jedinců bez závislosti na jejich dosavadním ohodnocení a mezi nimi je uspořádán "turnaj". Vyhrává jedinec s nejlepším ohodnocením z tohoto výběru.

Základní a nejjednodušší formou genetického algoritmu je tzv. kanonický nebo jednoduchý genetický algoritmus [5]:

1. Pro všechny členy populace je vygenerován náhodný genotyp.
2. Genotyp každého člena je převeden na fenotyp a je vyhodnocena jeho fitness.
3. Do tzv. prostoru páření (mating pool) jsou umístěny kopie jedinců stávající populace tak, že jedinci s lepší fitness jsou zde zastoupeny vícekrát než jedinci s horší fitness.
4. Náhodně jsou vybráni dva rodiče z prostoru páření.
5. Jsou vygenerováni dva potomci tak, že s určitou pravděpodobností dojde k vzájemnému náhodnému křížení genotypů obou rodičů.
6. S určitou pravděpodobností je aplikována náhodná mutace genotypu obou potomků.
7. Potomci jsou umístěni do nové populace a opakují se kroky 4-7, dokud není nová populace naplněna.
8. Kroky 2-8 se opakují, dokud není dosaženo uspokojivého řešení nebo nebylo dosaženo daného počtu generací.

2.2 Genetické programování

Autorem genetického programování se stal v 90-tých letech minulého století John R. Koza [10]. Genetické programování představuje zřejmě jednu z nejmladších technik evolučních algoritmů. Jedná se o speciální implementaci genetického algoritmu, který používá velmi specifické prostředky. Jeho základní myšlenkou je tzv. automatické programování [5], kde problém spočívá v nalezení programu, který řeší danou úlohu.

Specifickou vlastností genetického programování je zejména to, že genetická informace v chromozomu není reprezentována binárním kódem, ale funkcí. Z toho vyplývá, že na rozdíl od genetického algoritmu neexistují v genetickém programování dva oddělené prostory a význam genotypu a fenotypu je tak stejný.

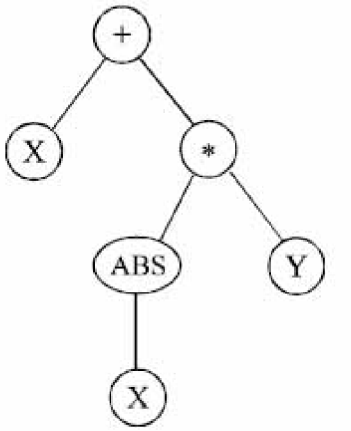
Pokud genetické programovýní vnímáme v rámci jeho původní myšlenky automatického programování, narazíme na další odlišnost od genetického algoritmu. Proces vyhodnocování není založen na několika vstupních hodnotách a jejich adekvátnímu výstupu, nýbrž na spuštění samotného programu, který je uložen v každém chromozomu.

Jedním z hlavních problémů genetického programování byla skutečnost, že funkce byly stále reprezentovány řetězci symbolů. Genetické informace tak nabývaly různých délek, což způsobovalo při křížení značné problémy. Stanovit fixní délku genotypu by bylo velice

neefektivní a pro samotné genetické programování deklasující, obzvláště proto, že počítačové programy samy o sobě mohou nabývat různých délek. V případě pevné velikosti chromozomu by v podstatě byla potlačena celá myšlenka, díky níž genetické programování vzniklo. Pro názornost uvedeme jednoduchý příklad:

```
chromosome_1 = 'A+B/C'
chromosome_2 = '~A/B+C'
křížení v bodě 1
child_1 = '~+B/C'
child_2 = 'AA/B+C'
```

Z příkladu je zřejmé, že výsledek je syntakticky nesprávný. John Koza nabídnul jedno z mnoha řešení v podobě hierarchické stromové struktury, která reprezentuje danou funkci. Výsledný přepis dané funkce můžeme vidět na obrázku 2.2.



Obrázek 2.2: Reprezentace kódu stromovou strukturou

Obdobně jako v genetickém agloritmu, jsou v genetickém programování použity dvě základní operace - křížení a mutace. Ty mohou být doplněny o některé další, mezi něž patří zejména:

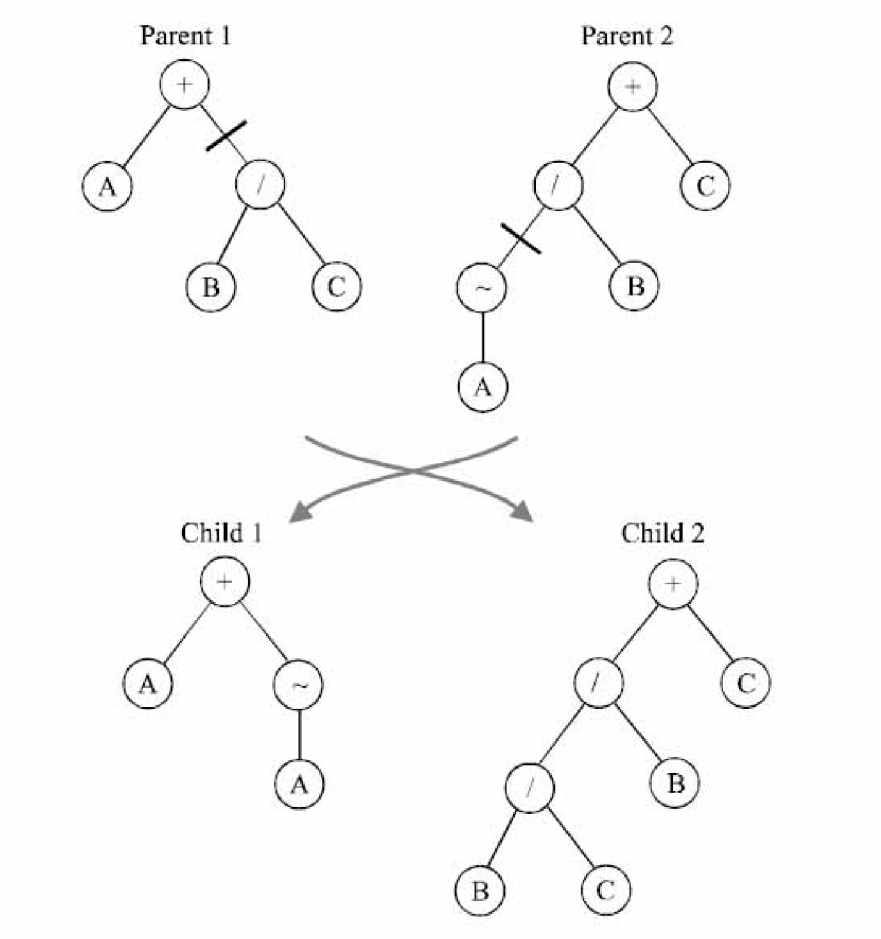
Permutace: Přehození pořadí potomků nějakého uzlu.

Editace: Zjednodušení podstromů, optimalizace (např. výskyt výrazu $1 + 1$ je optimalizován na 2).

Obalení: Zabalení podstromu do jediného uzlu, který je poté používán jako jeden uzel. To znamená, že např. nelze měnit jeho strukturu mutací.

Křížení je aplikováno na náhodně vybrané podstromy bez rizika porušení syntaktické správnosti, jak znázorňuje obrázek 2.3.

V případě mutace je náhodně vybrán uzel daného stromu, který je vymazán, včetně všech podstromů a nahrazen nově náhodně vygenerovaným podstromem (obrázek 2.4).



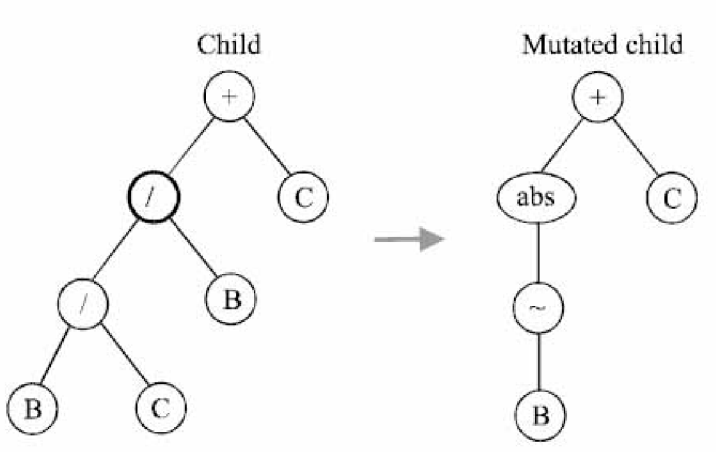
Obrázek 2.3: Křížení dvou částí stromových struktur

Existují však i implementace genetického programování, ve kterých je stromová struktura nahrazena např. obecně orinetovanými grafy (Parallel Algorithm Discovery and Orchestration) [1] nebo jako síť uzlů v kartezském souřadnicovém systému (Cartesian Genetic Programming) [2].

Všechny uvedené funkce v příkladech lze jednoduše přepsat do jazyka LISP a to zejména proto, že se jedná o jazyk, který umožňuje zapsat kombinaci funkcí jako řetězec.

V dnešní době nachází genetické programování uplatnění v mnoha oblastech počítačového inženýrství, elektroniky a dalších. Uveďme například symbolickou regresi, syntézu analogových elektrických obvodů, vyvíjející se obvody nebo hledání pravidel celulárních automatů.

Několik typických příkladů použití genetického programování [4]:



Obrázek 2.4: Operátor mutace aplikovaný na stromovou strukturu

	Úloha	Hledaný algoritmus	Vstup	Výstup
1.	indukce posloupnosti	analytický přepis	index	element posloupnosti
2.	symbolická regrese množiny dat	matematický výraz	nezávislé proměnné	závislé proměnné
3.	optimální řízení	řídící strategie	stavové proměnné	akční veličiny
4.	identifikace a predikce	matematický model systému	nezávislé proměnné	závislé proměnné
5.	klasifikace	rozhodovací strom	hodnoty atributů	přiřazení do třídy
6.	učení se cílenému individuálnímu chování	program popisující chování	vstupy ze senzorů	akce organismu
7.	odvození kolektivního chování	program popisující chování jedince	informace o vztahu jedince ke zbytku kolektivu	akce jedince v kolektivu

2.2.1 Symbolická regrese

Symbolická regrese patří mezi základní aplikace genetického programování [3]. Jejím principem je nalézt funkci, která by ideálně aproximovala body z tréninkové množiny. Jedná se o rozšíření regresní analýzy, kde pro danou tréninkovou množinu a funkci hledáme takové parametry funkce, aby co nejlépe pokryly tyto body. Jelikož v regresní analýze dochází pouze ke změně koeficientů funkce, má tato metoda velice omezený obor funkcí. Zevšeobecněním regresní analýzy dostáváme metodu aplikace genetického programování, kterou John Koza nazval symbolická regrese.

Pro zápis funkce v symbolické regresi se používá syntaktického stromu.

1. Funkcionální vrcholy odpovídají jednoduchým operacím, které mohou být n-árního typu.

2. Koncové vrcholy odpovídají konstantám nebo nezávislým proměnným.

Nezávislé proměnné jsou na závěr ohodnoceny reálnými čísly a syntaktický strom může být vyhodnocen některou z metod, nejčastěji metodou back-tracking. Z obrázku je zřejmé, že syntaktický strom umožňuje značnou variabilitu zápisu funkce a to zejména co se týče priorit.

V případě SR lze bez vážných problémů aplikovat operace (křížení, mutace, ...) nad syntaktickým stromem tak, jak byly představeny v kapitole o genetickém programování.

2.2.2 Gramatická evoluce

Jedna z nejnovějších evolučních technik kombinující model genetického algoritmu a genetického programování. Stejně tak jako genetické programovaní se gramatická evoluce používá pro automatické programování [4]. Základní odlišností však je, že je tato technika obecnější a použitelná pro libovolný jazyk popsatelný bezkontextovou gramatikou. Rozdíl lze také spatřit v reprezentaci rešení, které je v genetickém programování v podobě syntaktického stromu, kdežto v gramatické evoluci jde o binárně zakódování. Jako hlavní genetické operátory jsou použity jednoduché jednobodobé křížení a jednoduchá bodová mutace.

2.3 Evoluční strategie

Autory evoluční strategie se stali v 60-tých letech minulého století v Německu Bienert, Rechenberg a Schwefel [16, 17]. Evoluční strategie patří mezi historicky první úspěšné stochastické algoritmy. Na rozdíl od genetického algoritmu, který pracuje s binární reprezentací proměnných, je v evoluční strategii využito reprezentace řetězci reálných čísel (i zde však existují vyjímky). Základním operátorem evoluční strategie je mutace.

Mutace je v evoluční strategii aplikována dle předpisu

$$x' = x + N(0, \sigma)$$

kde $N(0, \sigma)$ je vektor nezávislých náhodně generovaných čísel s normálním rozložením , nulovou střední hodnotou a standardní odchylkou σ; x je aktuální hodnota chromozomu a x' je nová hodnota chromozomu. Nový chromozom je akceptovatelný, pokud je jeho řešení lepší než u původního chromozomu.

Směrodatná odchylka u operátoru mutace se v průběhu evoluce dynamicky mění dle určitých pravidel. Mezi nejznámější patří pravidlo 1:5. Změna směrodatné odchylky přímo závisí na koeficientu úspěšnosti, tedy poměru akceptovatelných řešení a celkovému počtu iterací, ve kterém byla úspěšnost sledována. Hraniční hodnotou je v tomto případě právě $\frac{1}{5}$ [3]. V případě, kdy je koeficient úspěšnosti větší než hraniční hodnota, dochází ke zmenšení standardní odchylky a naopak. Tento algoritmus zajišťuje v případě větších úspěchů zvětšení kroku vyhledávání. Naopak v případě nalezení podstatně větší míry neakceptovatelných řešení, zmenšení kroku.

Změna směrodatné odchylky:

pro $\varphi(k) < \frac{1}{5} \rightarrow \sigma' = C_d\sigma$
pro $\varphi(k) > \frac{1}{5} \rightarrow \sigma' = C_i\sigma$
pro $\varphi(k) = \frac{1}{5} \rightarrow \sigma' = \sigma$

kde $\varphi(k)$ je koeficient úspěšnosti, volitelná konstanta $C_d < 1$ zmenšuje velikost standardní odchylky
a volitelná konstanta $C_i > 1$ zvyšuje velikost standardní odchylky.

Pokud je v evoluční strategii používán operátor křížení, existuje několik nejpoužívanějších metod, jak tento operátor implementovat:

Průměrem Vektor hodnot potomka je dán průměrem jednotlivých hodnot vektorů rodičů.

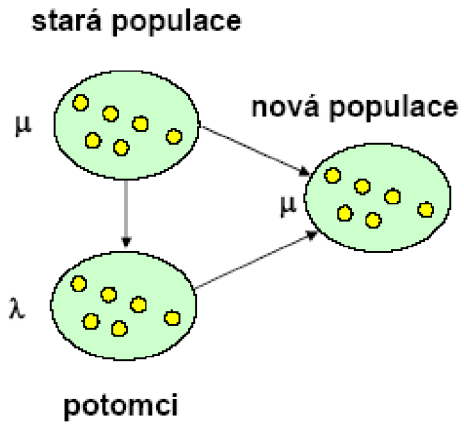
Diskrétní křížení Vektor hodnot potomka je tvořen náhodným výběrem hodnot vektorů rodičů.

Lineární kombinace V tomto případě vznikají dva potomci lineární kombinací vektorů svých rodičů.

V rámci evoluční strategie existují dvě základní metody selekce:

1. selekce $(\mu+\lambda)$ (obrázek 2.5) - po vygenerování λ potomků postupuje do další generace μ nejlepších jedinců z množiny $\mu + \lambda$. Jedná se tedy o společnou množinu rodičů a potomků.

2. selekce (μ, λ) (obrázek 2.6) - po vygenerování λ potomků postupuje do další generace μ nejlepších jedinců z takto vygenerované množiny. Vybírá se tedy pouze z množiny potomků.

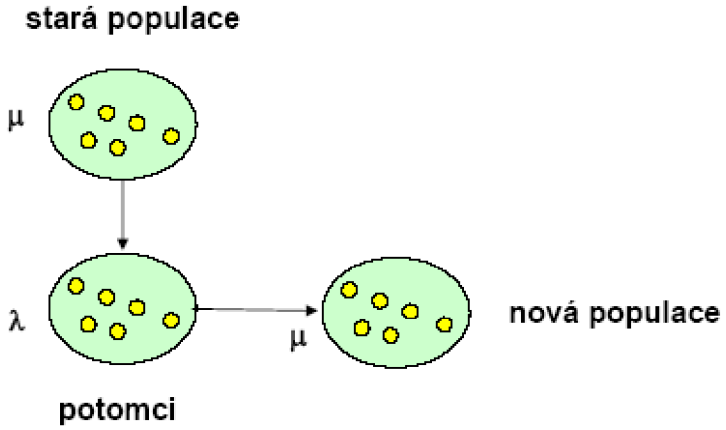


Obrázek 2.5: Selekce $(\mu + \lambda)$

Mezi první formy evoluční strategie patří strategie (1+1), která používá pouze dva členy a to jednoho rodiče a jednoho potomka. Jedná se o jednoduché vylepšení slepého stochastického algoritmu o dynamickou korekci směrodatné odchylky, v závislosti na počtu akceptovaných řešení.

Poměrně zajímavým rozšířením evoluční strategie je zavedení autoevoluce řídících parametrů.

1. Korekce standardní ochylky podle pravidla 1:5.



Obrázek 2.6: Selekce (μ, λ)

2. K vektoru hodnot je přiřazen adekvátní vektor směrodatných odchylek, které jsou postupně korigovány.

3. K vektoru hodnot a směrodatných odchylek je zaveden vektor natočení ve směru hledaného minima, který je taktéž korigován.

2.4 Evoluční programování

Autorem evolučního programmování se stal v 60-tých letech minulého století Lawrenc Fogel. Jde o uzavřenější variantu evoluční strategie, která byla vyvinuta nezávisle a také dříve. Cílem bylo evolučním postupem odvodit chování konečného automatu tak, aby byl schopen predikovat změny prostředí, v němž se nachází [6].

Prostředí je v tomto případě popsáno jako posloupnost symbolů z konečné abecedy. Výstupem je automat předpovídající další symbol této posloupnosti. Kvalita automatu je následně hodnocena spolehlivostí predikce.

Bylo navrženo pět mutačních operátorů pro modifikaci populace N náhodně vygenerovaných automatů:

1. změna závislosti výstupního symbolu na vnitřním stavu,

2. změna přechodové funkce vnitřních stavů automatu,

3. změna počátečního stavu automatu,

4. odstranění vnitřního stavu,

5. přidání vnitřního stavu.

Kapitola 3

Celulární automaty

Celulární automaty zavedli ve 40-tých letech minulého století Stanislaw Ulam a John von Neumann. Společně pracovali na samoreplikujících systémech. Základem se stal systém na sledování strojové reprodukce, operující pouze se základními strojovými instrukcemi [21].

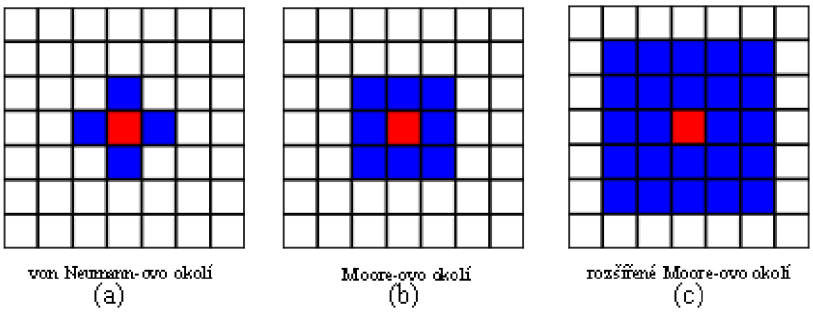
Celulární automat je matematický model fyzikálního systému, jehož prostor a čas jsou diskrétní a fyzikální veličiny nabývají diskrétních hodnot z konečné množiny. Základním stavebním kamenem celulárního automatu je buňka. Zpravidla se předpokládá, že počet buněk automatu je nekonečný, avšak v praxi je prostor celulárního automatu omezen. Buňky za hranicí celulární struktury potom mívají buď přesně definovaný stav nebo jsou cyklicky propojeny s buňkami opačné strany. Vytvářejí tak např. kruh, anuloid apod. Buňky celulárního automatu se nacházejí v jednom stavu z konečné množiny stavů. Jsou nejčastěji uspořádány do pravidelných struktur jako pole (jednorozměrný CA), mříže (dvourozměrný CA), případně do dalších vícerozměrných struktur [3].

Celulární automat se vyvíjí v diskrétních časových krocích. Následující stav buňky celulárního automatu je dán jejím aktuálním stavem a stavem buněk v definovaném sousedství této buňky. Tato závislost se nazývá lokální přechodová funkce (pravidlo). Lokální přechodové funkce mohou být pro všechny buňky automatu společná (uniformní celulární automat) nebo pro každou buňku individuální (hybridní - neuniformní celulární automat). Přechodem všech buněk do následujícího stavu je vytvořen jeden vývojový krok celulárního automatu. Přechod všech buněk automatu do následujícího stavu se nazývá vývojový krok celulárního automatu.

Jak bylo zmíněno, jedním z faktorů, kterým je dán následující stav buňky, je její okolí. V praxi existuje několik nejpoužívanějších typů okolí buňky, jako např. v 2D celulárních automatech von Neumannovo, Mooreovo nebo rozšířené Mooreovo okolí (obrázek 3.1).

Celulární automat je tedy definován čtyřmi základními parametry: strukturou elementárních buněk, konečným počtem stavů buněk, okolím buňky a algoritmem výpočtu následujícího stavu buňky celulárního automatu, nazývaného lokální přechodová funkce.

Celulární automat se osvědčil jako vhodný výpočetní model pro simulaci fyzikálních systémů, tvořených mnoha diskrétními prvky s lokálními interakcemi popsanými diferenciálními rovnicemi.



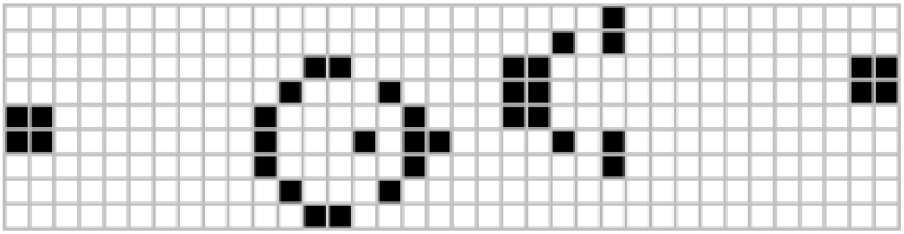
Obrázek 3.1: Nejpoužívanější typy okolí buňky 2D celulárního automatu: (a) von Neumannovo, (b) Mooreovo, (c) rozšířené Mooreovo

3.1 Binární celulární automaty

Jednou z nejznámějších aplikací celulárních automatů se stala v 70-letech minulého století tzv. hra "Life" (život buněk) [7] (obrázek 3.2). Autorem se stal John Conway. Narozdíl od původního von Neumannova celulárního automatu s 29 pravidly, se jednalo o celulární automat s binárnímy stavy buněk ("live-black", "dead-white") a následujícími lokálními předhodovými pravidly:

- Pokud má živá buňka přesně dva nebo tři živé sousedy, zůstává živá.
- Pokud má mrtvá buňka přesně tři živé sousedy, stává se živou.
- Ve všech ostatních případech hyne přemnožením (> 3) nebo v důsledku osamocení (< 2).

Tato "hra" bez "hráčů" prezentuje evoluci, která vychází pouze z počátečního stavu a nepotřebuje žádné jiné vstupní informace.



Obrázek 3.2: Příklad stavu celulárního automatu hry Life

Binární celulární automat představuje podtřídu celulárních automatů s dvěma stavy buňky. Základní vlastnosti binárního celulárního automatu popsal Stephen Wolfram a vytvořil tak základ teorie celulárních automatů [21]. Binární celulární automat můžeme rovněž chápat jako paralelně pracující aritmetickou jednotku. S využitím tohoto přístupu byly popsány paralelní násobičky, struktury pro zpracování obrazů a rozpoznávání znaků a další aplikace.

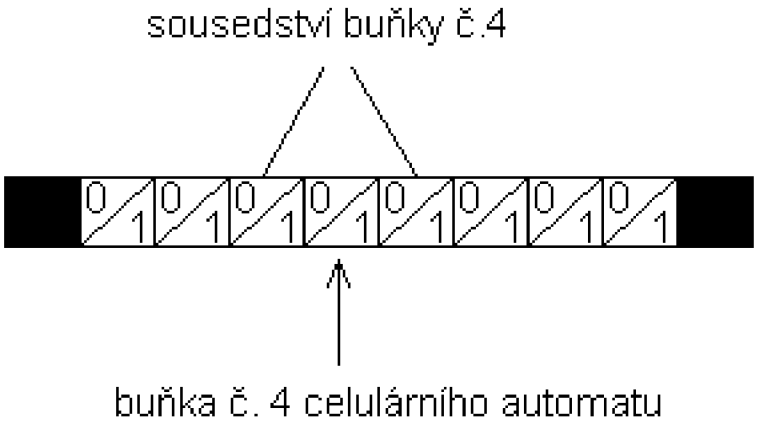
3.2 Hybridní - neuniformní celulární automaty

V rámci této práce se zaměříme na jednu specifickou skupinu celulárních automatů, kterou jsou hybridní - neuniformní celulární automaty. Specifickou vlatností a tedy i základním rozdílem mezi uniformním a neuniformních CA je počet lokálních přechodových funkcí a jejich přiřazení. V oboru neuniformních celulárních automatů nejsou přechodová pravidla pro všechny buňky automatu společná, nýbrž je každé buňce zvlášť přiřazena vlastní lokální přechodová funkce. Tento fakt činí tuto strukturu flexibilnější oproti variantě s jedinou lokální přechodovou funkcí. Uniformní celulární automaty jsou speciální variantou neuniformních celulárních automatů a to právě v případě, kdy jsou všechny lokální přechodové funkce neuniformního systému identické.

Samotnou množinu neuniformních celulárních automatů nyní ještě více omezíme a to podmínkami, které bude splňovat celulární automat použitý pro experimentální výsledky v této práci (obrázek 3.3). Těmito podmínkami je přesné stanovení dimenze `d` celulárního automatu, která bude v našem případě `d=1`. Dále omezíme množinu stavů, kterých mohou buňky celulárního automatu nabývat a to na symboly `{0, 1}`. A nakonec zavedeme konečnost celulárního automatu, tedy omezení na určitý počet buněk. Zavedením těchto podmínek, dostáváme binární jednorozměrný neuniformní celulární automat s konečným počtem buněk, který lze formálně definovat následovně:

Definice 1 [8, 18]: Sedmici $A = (Q, N, R, z, b_1, b_2, c_0)$ nazveme binárním jednorozměrným neuniformním celulárním automatem s konečným počtem buněk, kde

- $Q = 0, 1$ je binární množina stavů,
- $N : N \subseteq \mathbb{Z}$ je velikost sousedství,
- $z \in \mathbb{N}$ udává celkový počet buněk celulárního automatu,
- $b_1, b_2 \in Q^n$ jsou hodnoty okrajových podmínek,
- c_0 je počáteční konfigurace,
- $R : S \rightarrow (Q^N \rightarrow Q)$ zobrazení přiřazující každé buňce automatu $S = 1, 2, ..., z$ lokální přechodovou funkci $\delta_1, ..., \delta_z$, kde $\delta_i : Q^N \rightarrow Q$, nazveme přepisovací pravidlo, $i \in S$.



Obrázek 3.3: Příklad jednorozměrného binárního celulárního automatu: velikost sousedství je v tomto případě 1, černě jsou vyznačeny okrajové buňky automatu, čísla udávají možné stavy buněk.

Kapitola 4

Development

Development je jednou z dalších technik inspirovaných biologickými ději. Základní principy jsou odvozeny od ontogeneze, která popisuje individuální vývin jedince ze zygoty ve vyspělý vícebuněčný organismus. Jednotlivé fáze vývinu zárodečné buňky se dají charakterizovat následovně [11]:

Zárodečné dělení: V této fázi dochází k početnému dělení zárodečné buňky, jehož výsledkem je tzv. blastula - shluk buněk.

Organizace buněk: V embryu dochází k prostorové a časové organizaci buněk. V prvním stupni je vytvořen prostorový koncept organismu. V průběhu druhého stupně dochází k vytváření základní zárodečné vrstvy.

Morfogeneze: Změna tvarových vlastností - dochází k přemisťování buněk, utváří se základ vnitřností. Tento proces se nazývá gastrulace.

Odlišení buněk: Jednotlivé buňky získají vlastní strukturu a funkci, vznikají tak různě typy buněk.

Růst: Dochází k rozmnožování buněk a růstu buněčné hmoty - organismus nabývá na velikosti.

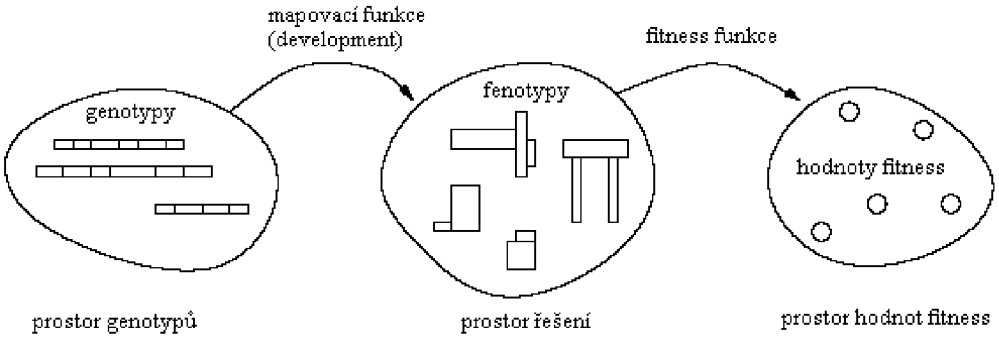
V klasickém evolučním návrhu je řešení daného problému přímo zakódováno do chromozomu (genotypu). Chromozom, resp. jeho kód, má pak jednoznačný ekvivalent v množině řešení, který lze přímo kvalitativně ohodnotit pomocí fitness funkce.

Development v evolučním návrhu mění poměrně zásadním způsobem právě takto zmíněné pojetí zakódování jednotlivých řešení. Nejedná se o evoluci genotypů přímo vázaných na fenotypy, nýbrž o evoluci předpisu pro konstrukci cílového objektu z tzv. embrya (obvykle triviální řešení daného problému). Samotné embryo nemusí být předem definováno a může být taktéž předmětem evoluce. Chromozomy tak v určité podobě obsahují mapovací funkci genotypu na fenotyp, která se stává předmětem evoluce jak ukazuje obrázek 4.1.

V oblasti developmentu můžeme pozorovat dva základní přístupy [14]:

1. **Development s konečným počtem kroků**: tento přístup je použit zejména tam, kde známe výsledný objekt a předmětem evoluce je samotný postup získání tohoto objektu. Jednotlivá řešení pak mohou být snadno porovnána.

2. **Development s nekonečným počtem kroků**: v tomto případě dané řešení nereprezentuje jeden konkrétní objekt. Výsledkem je množina objektů, které lze generovat



Obrázek 4.1: Princip techniky developmentu v evolučních algoritmech

postupnou aplikací předpisu, přičemž v každém kroce nebo v určité sekvenci kroků, je vždy vygenerován platný výsledek.

Kapitola 5

Aplikace celulárních automatů jako vývojových modelů

Celulární automaty lze s výhodou využít ve zmiňované technice developmentu. Můžeme tu pozorovat přímou souvztažnost ke konstrukci cílového objektu, který je sestavován na základě vývoje celulárního automatu. Samotné embryo je zde zastoupeno počáteční konfigurácí automatu a samo o sobě může být též předmětem evoluce. Jednotlivé vývojové stupně celulárního automatu, pak lze substituovat s určitým vývojovým krokem výsledného objektu.

V této práci se zaměříme na specifickou oblast developmentu ve spojení s evolučním návrhem a celulárními automaty. Pokusíme se nalézt celulární automat, který dle určitých principů dokáže v konečném počtu aplikací lokálních přechodových funkcí, vygenerovat požadovaný kombinační logický obvod.

5.1 Návrh kombinačních logických obvodů

V oblasti kombinačních logických obvodů bylo již evolučních technik využito v řadě případů. Jako příklad můžeme uvézt kartézské genetické programování [20], kde chromozom představuje matici hradel a jejich propojení. V našem případě však nepůjde o evoluční vývoj matice, která přímo reprezentuje daný obvod, nýbrž o evoluci celulárního automatu, který generuje požadovaný obvod postupně, v jednotlivých krocích.

Jak bylo zmíněno výše, v oblasti celulárních automatů jsme vybrali velmi specifickou skupinu a to binární jednorozměrný neuniformní celulární automat. Možné stavy jednotlivých buněk celulárního automatu jsou známy, avšak je nutné specifikovat, jak bude definováno okolí buněk. Zde se předem neomezíme jen na dva nejbližší sousedy dané buňky. Ponecháme velikost okolí variabilní, avšak s jednou podmínkou, že rozsah okolí bude souměrný na obě strany od aktuální buňky. Tedy počet uvažovaných buněk z jedné strany bude identický počtu buněk ze strany druhé.

Jelikož pracujeme s neuniformním celulárním automatem, bude pro každou buňku celulárního automatu specifikována individuální lokální přechodová funkce. Tímto způsobem značně naroste celkový počet přepisovacích pravidel každé buňky automatu. Další skutečnost, která bude významně ovlivňovat počet přepisovacích pravidel, je velikost okolí. Jak jsme zmínili, velikost okolí není předem omezena a sama o sobě může přepisovacích pravidel ještě zvýšit. Proto bude v některých složitějších případech nutné omezit počet přepisovacích pravidel v lokální přechodové funkcí jedné buňky na určitou velikost a stavům,

které nebudou popsány, přiřadit jediné souhrnné přepisovací pravidlo.

V tomto okamžiku nám chybí nadefinovat poslední zásadní prvek a tím je generování výsledného kombinačního obvodu daným celulárním automatem. Pro tento účel rozšíříme celulární automat uvedený v definici 2 o generativní schopnosti. Každý stav buňky a jejího okolí v jednotlivých vývojových stupních celulárního automatu nebude pouze zdrojem pro výpočet následujícího stavu, ale také generátorem určitého hradla a jeho vstupů. V každém vývojovém kroku celulárního automatu nám tak vznikne jeden paralelní stupeň výsledného kobminačního logického obvodu. Takový celulární automat můžeme popsat následovně:

Definice 2: Celulární automat s generativními schopnostmi je algebraická struktura $A = (Q, N, \Sigma, R, \lambda, z, b_1, b_2, c_0)$, kde

- $Q = 0, 1$ je binární množina stavů,
- $N : N \subseteq \mathbb{Z}$ je velikost sousedství,
- $\Sigma \neq \emptyset$ je konečná množina symbolů,
- $z \in \mathbb{N}$ udává celkový počet buněk celulárního automatu,
- $b_1, b_2 \in Q^n$ jsou hodnoty okrajových podmínek,
- c_0 je počáteční konfigurace,
- $\lambda : Q^N \rightarrow \Sigma$ je funkce generující pro každé přechodové pravidlo, tvaru $q_{-1}q_0q_1 \rightarrow q'$ nějaký symbol ze Σ,
- $R : S \rightarrow (Q^N \rightarrow Q)$ zobrazení přiřazující každé buňce automatu $S = 1, 2, ..., z$ lokální přechodovou funkci $\delta_1, ..., \delta_z$, kde $\delta_i : Q^N \rightarrow Q, i \in S$.

Nutno uvést, že cílový kombinační obvod je vygenerován po provedení určitého konečného počtu vývojových kroků celulárního automatu.

5.2 Příklad konstrukce kobminačního logického obvodu

Pro názornost uvedeme konkrétní příklad celulárního automatu a finální konstrukci výsledného kombinačního logického obvodu. Velikost celulárního automatu je přímo úměrná počtu vstupů kombinačního obvodu v poměru 1:1. Jelikož se jedná o neuniformní celulární automat, existuje pro každou buňku celulárního automatu individuální lokální přechodová funkce. Lokální přechodová funkce je závislá jak na aktuálním stavu buňky, tak na jejím okolí, které je v tomto konkrétním případě určeno levým a pravým sousedem. Cílovým řešením celulárního automatu je schopnost vygenerovat kombinační obvod, jehož výstupem budou námi požadované hodnoty.

Celulární automat odpovídá definici 3 a má následující vlastnosti:

- velikost celulárního automatu: 3
- velikost okolí: 1
- okrajové podmínky: `0xxx0`
- inicializační stav: `01100`
- počet vývojových kroků: 3

Číslo pravidla	0	1	2	3	4	5	6	7
Schéma pravidla v CA	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
Následující stav buňky	`1`	`1`	`1`	`1`	`1`	`1`	`1`	`1`
Generovaný symbol (hradlo)	NXOR (0 2)	NOT (1 2)	NOR (0 1)	NXOR (0 2)	NOR (0 2)	XOR (1 2)	XOR (0 1)	XOR (1 2)

Tabulka 5.1: Lokální přechodová funkce a generované symboly buňky 1

Číslo pravidla	0	1	2	3	4	5	6	7
Schéma pravidla v CA	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
Následující stav buňky	`0`	`1`	`0`	`1`	`1`	`1`	`0`	`1`
Generovaný symbol (hradlo)	AND (0 1)	NOT (0 1)	AND (1 2)	NOT (0 1)	NXOR (0 1)	NOR (1 2)	LINE	OR (0 1)

Tabulka 5.2: Lokální přechodová funkce a generované symboly buňky 2

Tabulky 5.1, 5.2 a 5.3 obsahují lokální přechodové funkce a generované symboly pro danou kombinaci stavů buněk, v definovaném sousedství buňky automatu. Generovaný symbol se skládá z názvu hradla a jeho vstupů, které se vážou na předchozí vygenerovaný stupeň kombinačního logického obvodu. V případě, že ještě nebyl žádný stupeň obvodu vygenerován, jedná se přímo o vstupy kombinačního obvodu. V případě symbolu NOT, který je jednovstupový, je rozhodným vstupem vždy první z uvedených. Pro zbývající buňky automatu, tedy buňky č. 0 a č. 4, nejsou žádná pravidla uvedena, jelikož se jedná o okrajové podmínky automatu a ty samy osobě nepřecházejí do žádného jiného stavu.

První stupeň kombinačního obvodu je generován přímo z inicializačního stavu celulárního automatu. Pro jednotlivé buňky platí následující:

1. Uvažujeme tabulku 5.1 a hledanou posloupnost symbolů `011`. Aplikujeme odpovídající pravidlo č. 3. Generujeme hradlo NXOR se vstupy č.0 a č.2. Následující stav buňky je `1`.

2. Uvažujeme tabulku 5.2 a hledanou posloupnost symbolů `110`. Aplikujeme odpovídající pravidlo č. 6. Generujeme vodič se vstupem ze stejné pozice, na které je vodič vygenerován. Následující stav buňky je `0`.

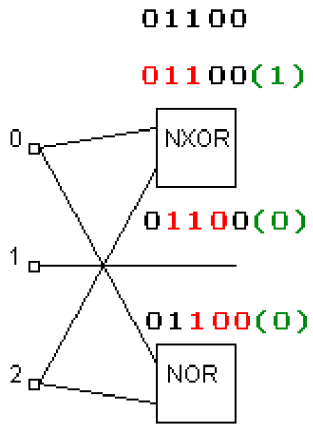
3. Uvažujeme tabulku 5.3 a hledanou posloupnost symbolů `100`. Aplikujeme odpovídající pravidlo č. 4. Generujeme hradlo NOR se vstupy č.0 a č.2. Následující stav buňky je `0`.

Následující stav celulárního automatu je tedy `01000`. Výsledný stupeň kombinačního logického obvodu můžeme vidět na obrázku 5.1.

Druhý stupeň kombinačního obvodu vygenerujeme obdobně jako první, s tím rozdílem, že neuvažujeme inicializační stav automatu, ale stav vygenerovaný v předešlém vývojovém kroku. Nutno také připomenout, že jednotlivými vstupy jsou nyní výstupy posledního vygenerovaného stupně kombinačního obvodu. Pro jednotlivé buňky platí následující:

Číslo pravidla	0	1	2	3	4	5	6	7
Schéma pravidla v CA	000	001	010	011	100	101	110	111
Následující stav buňky	0	0	1	0	0	0	0	0
Generovaný symbol (hradlo)	LINE	NAND (0 2)	NOT (1 2)	NAND (0 2)	NOR (0 2)	NOR (0 2)	NAND (0 2)	LINE

Tabulka 5.3: Lokální přechodová funkce a generované symboly buňky 3



Obrázek 5.1: Kombinační obvod po prvním vývojovém kroku CA. Řada symbolů představuje aktuální stav automatu, červeně je vyznačen stav a okolí uvažované buňky a zeleně následující stav buňky.

1. Uvažujeme tabulku 5.1 a hledanou posloupnost symbolů `010`. Aplikujeme odpovídající pravidlo č.2. Generujeme hradlo NOR se vstupy č.0 a č.1. Následující stav buňky je `1`.

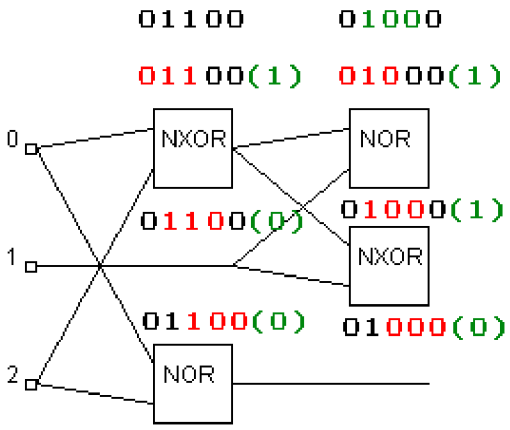
2. Uvažujeme tabulku 5.2 a hledanou posloupnost symbolů `100`. Aplikujeme odpovídající pravidlo č.4. Generujeme hradlo NXOR se vstupy č.0 a č.1. Následující stav buňky je `1`.

3. Uvažujeme tabulku 5.3 a hledanou posloupnost symbolů `000`. Aplikujeme odpovídající pravidlo č.0. Generujeme vodič. Následující stav buňky je `0`.

Následující stav celulárního automatu je `01100`. Po druhém vývojovém stupni můžeme vidět kombinační logický obvod na obrázku 5.2.

Jelikož jsme se dostali do stejného stavu celulárního automatu, jako byl počáteční stav, je poslední stupeň kombinačního obvodu totožný s prvním stupněm. Výsledný obvod vygenerovaný pomocí celulárního atuomatu si můžeme prohlédnout na obrázku 5.3.

V předchozích několika odstavcích jsme si představili algoritmus konstrukce výsledného kombinačního logického obvodu pomocí binárního jednorozměrného uniformního celulárního automatu. Na něm lze jasně pozorovat rozdíl od generování stejných obvodů pomocí kartézského genetického programování, jak jsme uvedli výše. Ukázali jsme si též, že počet pravidel je i v takto jednoduchém příkladu poměrně velký a s velikostí automatu a okolí ještě značně



Obrázek 5.2: Kombinační obvod po druhém vývojovém kroku CA. Řada sybmolů představuje aktuální stav automatu, červeně je vyznačen stav a okolí uvažované buňky a zeleně následující stav buňky.

naroste. Jenom poznamenejme, že vygenerovaný obvod uvedený na obrázku 5.3 představuje jednobitovou úplnou sčítačku.

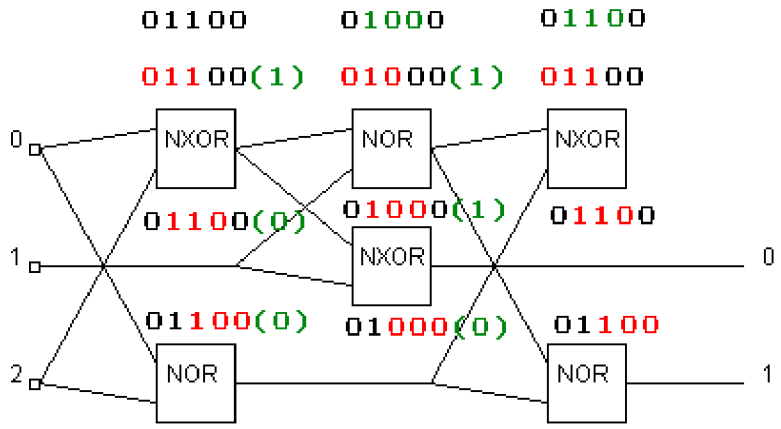
5.3 Polymorfní logické obvody

V oblasti logických obvodů existuje další skupina, nazývaná polymorfní logické obvody, na kterou lze metodu developmentu ve spojení s evolučními algoritmy použít. Klasický přístup návrhu multifunkčních systémů je založen na přepínání výstupů jednotlivých subsystémů, které reprezentují určitou funkci [19]. Polymorfní obvody jsou naproti tomu založeny na zcela jiném principu. Logická hradla nepředstavují jednu funkci jak je tomu u běžných obvodů. Polymorfní hradlo může reprezentovat několik funkcí zároveň (obrázek 8.8) s tím, že aktivní je vždy jedna z možných v závislosti na další externí veličině. Tímto externím činitelem může být např. hodnota napětí, teplota, intenzita záření apod. Pro různé hodnoty externí veličiny pak logický obvod nabývá jiného funkčního významu. Příklady polymorfních hradel můžeme vidět v tabulce 8.8.

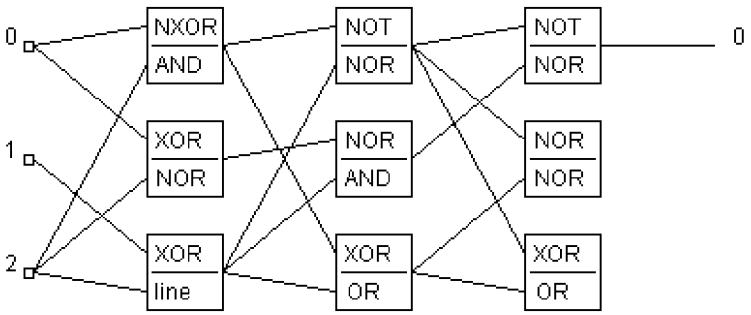
hradlo	řídící hodnoty	externí veličina	počet tranzistorů
AND/OR	27/125C	teplota	6
AND/OR/XOR	3.3/0.0/1.5V	externí zdroj napětí	10
AND/OR	3.3/0.0V	externí zdroj napětí	6
AND/OR	1.2/3.3V	napětí	8
NAND/NOR	3.3/1.8V	napětí	6

Tabulka 5.4: Příklad polymorfních hradel [12]

V rámci této práce prověříme i tuto oblast logických obvodů a pokusíme se navrhnout multifunkční kombinační logický obvod. K již navrženému postupu přidáme externí činitel v podobě binárního vstupu jednotlivých hradel obvodu. Stavy tohoto vstupu budou přepínat mezi dvěmi logickými funkcemi každého hradla.



Obrázek 5.3: Výsledný kombinační obvod. Řada symbolů představuje aktuální stav automatu, červeně je vyznačen stav a okolí uvažované buňky a zeleně následující stav buňky.



Obrázek 5.4: Příklad polymorfního kombinačního logického obvodu. Každé hradlo může nabývat jednoho ze dvou významů. Vrchní symbol je použit pro obvodovou funkci reprezentovanou hodnotou 0 a spodní symbol pro funkci reprezentovanou hodnotou 1. Obrázek představuje plně funkční tříbitovou boolovskou symetrii (0) / řadicí síť (1).

Kapitola 6

Evoluční algoritmus

V jedné z úvodních kapitol jsme popsali některé vybrané techniky z oboru nazvaného evoluční algoritmy. Naznačili jsme také, že v oblasti developmentu využijeme právě tohoto oboru a aplikujeme evoluční techniky na vytvoření struktury, která bude představovat generátor výsledného objektu. Nyní je na čase si uvést jednu z těchto technik, která bude použita pro evoluci celulárního automatu a která co nejlépe přísluší dané problematice.

Jak jsme uvedli výše, předmětem evoluce bude v rámci tohoto projektu binární jednorozměrný neuniformní celulární automat s generativními schopnostmi, který bude sloužit jako generátor výsledného kombinačního logického obvodu. Předmětem evoluce budou dílčí parametry celulárního automatu, které si nyní uvedeme:

Počáteční konfigurace celulárního automatu představuje vstup pro generování prvního stupně logického obvodu. Ve své podstatě reprezentuje v oblasti developmentu embryo, které není předem známo a samo o sobě je jedním z prvků, které jsou předmětem evoluce. Důležitým faktem je skutečnost, že počáteční konfigurace obsahuje též okrajové podmínky celulárního automatu, které zůstávají pro každý stav automatu neměnné.

Lokální přechodové funkce představují předpis pro konstrukci výsledných kombinačních logických obvodů. Uvedli jsme, že použitý celulární automat bude mít určité generativní schopnosti. Z tohoto důvodu nebudou předpisy lokálních přechodových funkcí obsahovat jen aktuální stav buňky, včetně jejího okolí a následujícího stavu. Součástí bude též generovaný symbol, v našem případě jedno z možných hradel a označení vstupů hradla. Velmi důležitým faktorem, který jsme již uvedli, je neuniformita celulárního automatu. Tato vlastnost přináší rozšíření v podobě individuálních lokálních přechodových funkcí pro každou buňku celulárního automatu. Předmětem evoluce se tak stává celý soubor těchto funkcí, představujících jádro výsledného generátoru.

Pro každý z těchto parametrů lze vytvořit binární řetězec, který jej bude reprezentovat. Tato skutečnost nám dovoluje využít v rámci tohoto projektu techniky, příslušející do množiny evolučních algoritmů - genetický algoritmus.

6.1 Genetický algoritmus

V této části si představíme stěžejní parametry a implementaci některých zásadních prvků genetického algoritmu, jako např. složení chromozomu, operátor křížení, mutace a selekční metodu.

Chromozom

Jak jsme zmínili výše, předmětem evoluce jsou lokální přechodové funkce a počáteční konfigurace automatu. Jednou z nejdůležitějších částí evolučního algoritmu je kvalitní kódování daného řešení. V tomto případě je chromozom (obrázek 6.1) rozdělen na dvě části:

1. počáteční konfigurace celulárního automatu a
2. lokální přechodové funkce.

Uvedli jsme, že okolí buňky jednorozměrného celulárního automatu bude pro jednotlivé experimenty volitelné. Dále pak, že pracujeme s binárním celulárním automatem a tudíž jednotlivé buňky mohou nabýt jednoho z předem definované množiny stavů $\{0, 1\}$. Z těchto vlastností celulárního automatu vyplývá, že maximální možný počet přepisovacích pravidel jedné buňky je dán vztahem $2^{1+\texttt{velikost\_okolí}*2}$. Vzhledem k možnému velkému nárustu přepisovacích pravidel lokálních přechodových funkcí, který by mohl značně ovlivnit časovou složitost výsledného výpočtu, zavedeme možnost definovat výchozí (defaultní) přepisovací pravidlo. Ve výsledném chromozomu tak nemusí dojít k vygenerování přepisovacích pravidel pro všechny možné varianty stavu buňky a jejího okolí. Na situace, které nebudou pokryty individuálními přepisovacími pravidly bude s výhodou využito právě výchozí (defaultní) přepisovací pravidlo. Z této vlastnosti snadno odvodíme, že velikost chromozomu může být pro jednotlivé případy odlišná.

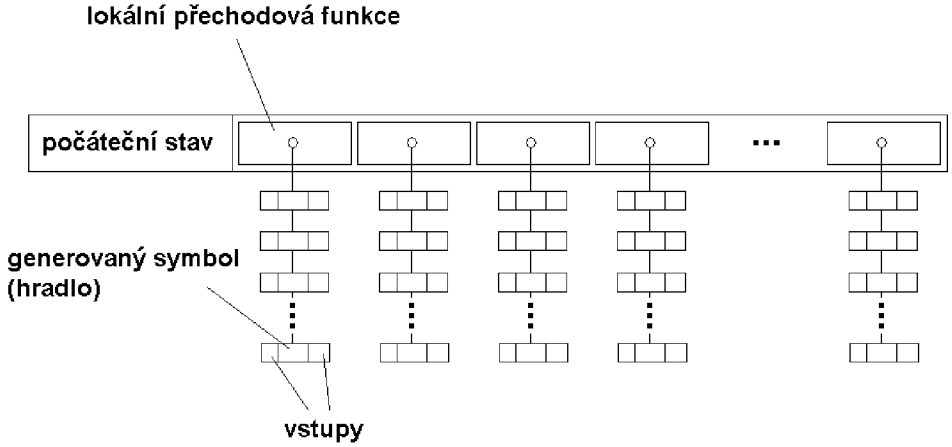
Nesmíme zapomenout na další dva prvky, které musejí přepisovací pravidla lokální přechodové funkce chromozomu obsahovat. Tím prvním je bezesporu výsledný generovaný symbol, tedy jedno z možných hradel (v případě polymorfních obvodů dvě hradla), které lze použít ke konstrukci výsledného kombinačního logického obvodu. Tím druhým jsou jednotlivé vstupy těchto hradel. V rámci této práce budeme většinou pracovat s hradly o dvou vstupech (AND, OR, XOR, ...), ale můžeme se setkat i s jednovstupovým (NOT) nebo s řešením, které bude představovat pouhý vodič. Tyto případy budou kódovány identicky s dvoustupovými hradly, avšak při výsledném generování bude brán v úvahu jen maximální možný počet vstupů hradla.

V tomto okamžiku je na místě připomenout jednu zásadní skutečnost. Genetický algoritmus je použit ve spojení s technikou developmentu a tudíž tu neexistuje přímá vazba mezi genotypem a fenotypem. Jako fenotyp by v našem případě mohl vystupovat výsledný celulární automat, avšak není tomu tak. Cílovým objektem je v našem případě kobminační logický obvod, s jehož interpretací pracujeme např. při vyhodnocování fitness. Pomocí genetického algoritmu se sice snažíme evolvovat celulární automat, ale v následné návaznosti na kombinační obvod. Celulární automat tak slouží pouze jako generátor výsledného řešení. Proto jej nelze považovat za fenotyp.

Fitness

Fitness nebo také kvalitativní ohodnocení chromozomu nám udává, jak moc se aktuální řešení blíží optimálnímu. Jedná se o ukazatel, který v evolučním procesu hraje poměrně zásadní roli, jelikož v závislosti na jeho hodnotě může dojít k určitým událostem s daleko větší pravděpodobností. Správné nastavení fitness funkce bývá jedním z nejobtížnějších procesů v oblasti evolučních algoritmů.

Fitness musí vystihovat kvalitu nejen vzhledem k optimálnímu řešení, ale také vzhledem k méně kvalitním řešením ostatních chromozomů. Jako příklad můžeme použít jednu



Obrázek 6.1: Schéma chromozomu genetického algoritmu: první část tvoří počáteční stav celulárního automatu, následující části pak lokální přechodovou funkci pro jednotlivé buňky automatu.

z předchozích prací zabývající se aproximací bodů pomocí symbolické regrese [15]. Uvažujme dvě řešení (funkce), jejichž body na daných souřadnicích budou ideálně pokrývat zadané body. Pokud bychom stanovili velikost fitness pouze na základě absolutního rozdílu zadaných a výsledných bodů, nebude fitness plně poskytovat informaci o kvalitě řešení. V tomto případě nám totiž fitness funkce nezahrnuje informace o průběhu funkce v celém sledovaném intervalu. Dochází tedy k situaci, kdy dvě zmiňovaná řešení mají stejné ohodnocení, ovšem i v případě kdy jedna z nich bude v požadovaném intervalu nespojitá. Fitness funkce pak nepodává zcela pravdivý obraz kvality řešení, jelikož spojitý výsledek je zajisté kvalitativně lepší než nespojitý. V našem případě se dopustíme v případě fitness funkce drobné nedůslednosti, avšak zdůvodníme, proč je tomu tak.

V rámci našeho projektu pracujeme s kombinačními logickými obvody a na počátku známe jak vstup, tak požadovaný výstup. Fitness funkce bude tedy zohledňovat odlišnosti ve výstupu řešení daného chromozomu a požadovaného výstupu. V tomto případě si vystačíme s počtem rozdílných bitů. Čím více rozdílných bitů u výstupu řešení chromozomu a požadovaného řešení, tím vyšší fitness. V případě, kdy bude fitness nulové, bylo nalezeno řešení požadovaného kombinačního logického obvodu. Nyní uveďme již zmiňovanou drobnou nedůslednost ve stanovení fitness. Opět může nastat situace, kdy budou vygenerována dvě řešení reprezentující požadovaný kombinační obvod (tedy obě s nulovým fitness) a každé z nich bude zkonstruováno z rozdílného počtu hradel. V tomto případě by logicky mělo být lépe ohodnoceno řešení s menším počtem hradel, avšak v našem případě tuto skutečnost pomineme. Důvodem je časová náročnost určení počtu hradel, která jsou objektivně využita v kombinačním logickém obvodu. Většina výsledků bude pravděpodobně obsahovat nějaké hradlo, které je v konečném důsledku irelevantní. Tato hradla by bylo třeba eliminovat a to u každého chromozomu každé generace evoluce zvlášť. Proto tento faktor nebude fitness funkce zohledňovat.

Ještě uveďme jedno specifikum, které budeme při vyhodnocování fitness používat. Vzhledem k tomu, že výstup z vygenerovaného obvodu může obsahovat více vodičů než požado-

vaný výstup, můžeme v těchto případech nalézt více než jedno řešení. Počet možných řešení jednoho chromozomu je dán počtem permutací výstupu bez opakování. Pro všechny tyto kombinace lze individuálně vypočítat rozdíl mezi výstupem generovaného obvodu a požadovaným výstupem, tedy několik fitness funkcí. Z časového hlediska je však dosti náročné prověřovat všechny možné kombinace výstupu, zejména pak u vícebitových kombinací, proto se zaměříme jen na možnosti `n` po sobě následujících bitů výstupu. Každý vyhodnotíme zvlášť a hodnotou fitness bude nejúspěšnější z výsledků.

Operátor křížení a mutace

Operátor křížení a mutace patří mezi základní operátory genetického algoritmu. Jedná se také o jednu z vlastností, kterou se genetický algoritmus odlišuje od většiny ostatních evolučních algoritmů. Principy těchto operátorů jsme uvedli v kapitole pojednávající o evolučních algoritmech. Nyní provedem aplikaci na konkrétní problematiku, kterou je v našem případě celulární automat.

Oba operátory lze aplikovat přibližně dvěma způsoby:

1. Křížení a mutace budou probíhat nad lokální přechodovou funkcí jedné konkrétní buňky. V případě křížení dojde k vzájemné výměně přechodových funkcí příslušné buňky dvou chromozomů a v případě mutace k vygenerování zcela nové lokální přechodové funkce.

2. Křížení a mutace budou probíhat jen nad jedním přepisovacím pravidlem lokální přechodové funkce. V případě křížení dojde k vzájemné výměně jednoho přepisovacího pravidla přechodové funkce a v případě mutace k vygenerování pravidla nového.

V rámci prvotních testů se prokázal první přístup jako méně efektivní než druhý a proto oba operátory budou aplikovány na drobnější struktury v rámci celého objektu, jak je popsáno v případu 2.

Selekční metoda

Selekční metoda je poměrně podstatným prvkem při reprodukčním procesu. Do reprodukčního procesu je nutné vybírat jedince s určitým ohledem na jejich úspěšnost vůči požadovanému řešení. Tedy chromozomy s lepším ohodonocením (menší hodnotou fitness), mají větší pravděpodobnost vstoupit do reprodukčního procesu, než méně úspěšné chromozomy (s větší hodnotou fitness). Selekční metoda musí brát také ohledy na časovou náročnost algoritmu a na celkové možností řešeného problému.

V rámci tohoto projektu využijeme turnajové selekce, která splňuje veškeré kladené nároky. Náhodně bude vybráno `n` chromozomů, mezi nimiž bude uskutečněn tzv. “turnaj”. Ten spočívá ve vybrání 2 nejlepších chromozomů dle hodnot fitness z takto selektované množiny. Takto vybraní jedinci potom vstupují do reprodukčního procesu.

Generování následující populace

V rámci reprodukčního procesu, kam vstupují chromozomy z aktuální populační množiny, vznikají noví jedinci. Existuje několik způsobů jak se k takto nově vzniklé množině jedinců zachovat a jak vytvořit populaci následující:

- Množina vygenerovaných jedinců je větší než velikost následující populace - jsou vybráni nejlepší jedinci.

- Množiny aktuální populace a vygenerovaných jedinců je sjednocena a je vybrán určitý počet nejlepších jedinců.
- Do nové generace je přeneseno vždy několik málo nejlepších jedinců z aktuální populace a zbytek je doplněn výsledky reprodukčních procesů.
- Nově vygenerovaní jedinci představují přímo následující populaci.

Jednotlivé metody mohou mít poměrně podstatný vliv na vývoj populace a celý evoluční proces. V rámci prvotního nastavení genetického algoritmu bylo přistoupeno k aplikování každé z těchto metod a porovnání dílčích výsledků. Metody, které nějakým způsobem aplikovaly upřednostňování nejlepších jedinců - elitismum, se ukázaly jako méně efektivní než tvorba populace jen z výsledků reprodukčního procesu. V případech různých variant elitismu, docházelo poměrně rychle ke konvergenci k jednomu řešení, což vedlo k méně úspěšným výsledkům.

Kapitola 7

Počáteční nastavení parametrů algoritmu

V rámci celého algoritmu existuje několik velmi zásadních parametrů, které mohou mít značný vliv na celý průběh evoluce a na úspěšnost výsledku. Ještě než tedy přistoupíme k samotným experimentálním pokusům, je žádoucí zjistit přibližné výchozí hodnoty těchto parametrů. Úmyslně uvádíme přibližné hodnoty, jelikož každý budoucí experimentální pokus se zaměřením na jiný kombinační obvod, je svým způsoem individuální. Pro každý obvod tak mohou platit mírně odlišné hodnoty, avšak určité rizikové hranice budou velmi podobné a proto je třeba je detekovat ještě před samotnými pokusy. Testy budou prověřovat následující nastavení parametrů:

- velikost populace,
- procentuální pravděpodobnost mutace lokálních přechodových funkcí nového jedince,
- počet jedinců využitých v turnajové selekci,
- procentuální pravděpodobnost mutace počátečního stavu automatu u nového jedince,
- počet přepisovacích pravidel lokální přechodové funkce jedné buňky celulárního automatu.

Základním obvodem, který použijeme, bude jednobitová úplná sčítačka a maximální počet generací omezíme na 1000.

7.1 Velikost populace

Velikost populace určuje množství chromozomů a tedy i počet řešení v daném okamžiku vývoje. Počet chromozomů může významně ovlivnit úspěšnost algoritmu a to v případě, kdy bude poměrně malý. Na druhou stranu, se zvyšující se velikostí populace, výrazně rostou časové nároky na vývoj jedné generace a v konečném důsledku i na celkové provádění pokusů.

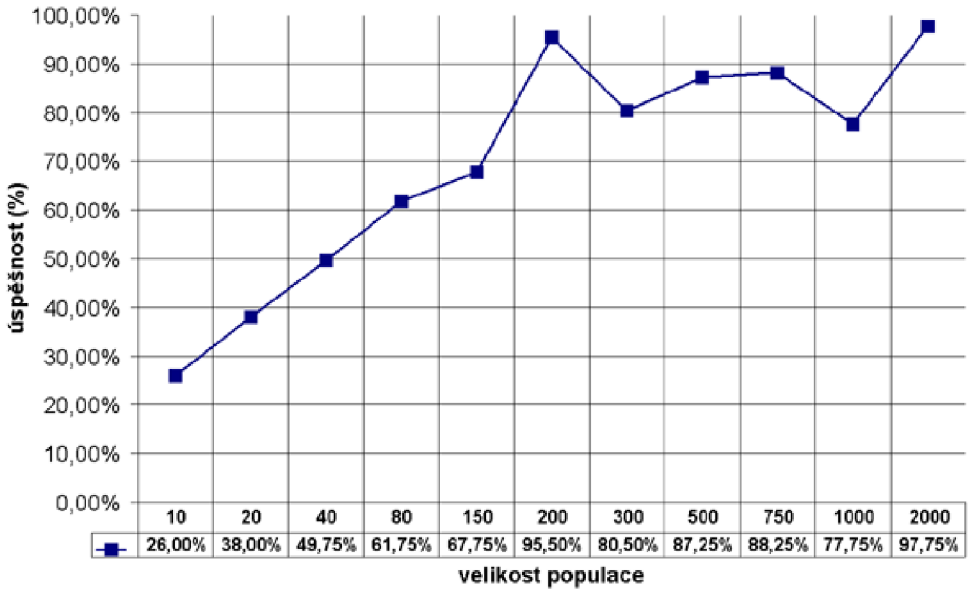
Podívejme se nyní na výsledky testů zaměřených právě na zmiňovaný parametr (400 nezávislých experimentů pro každé nastavení parametru). V grafu na obrázku 7.1 můžeme sledovat výrazný nárůst úspěšnosti algoritmu v první části grafu, kde byla velikost populace

v řádu desítek jedinců. Mezi hodnotami 80 až 200 jedinců dochází k postupné stabilizaci a při vyšších hodnotách objemu populace nejsou již patrná žádná enormní zlepšení.

Z grafu na obrázku 7.2 je patrné, že i rychlost nalezení řešení, tedy generace, ve které bylo odpovídající řešení nalezeno, se s velikostí populace zvyšuje. Opět je patrný výrazný zlom mezi hodnotami 80 až 200, kde dochází ke stabilizaci.

Můžeme též sledovat poměrně velkou úspěšnost kolem honoty 2000. Velká populace v tomto případě umožňuje vygenerování velkého množství potomků, kteří mohou být poměrně rozmanití a řešení je tak u tohoto typu obvodu nalezeno relativně brzy. Avšak časové nároky na nalezení řešení v tomto případě vzrostly 3-4x.

Doporučené hodnoty byly stanoveny na 80 až 200 jedinců v populaci.



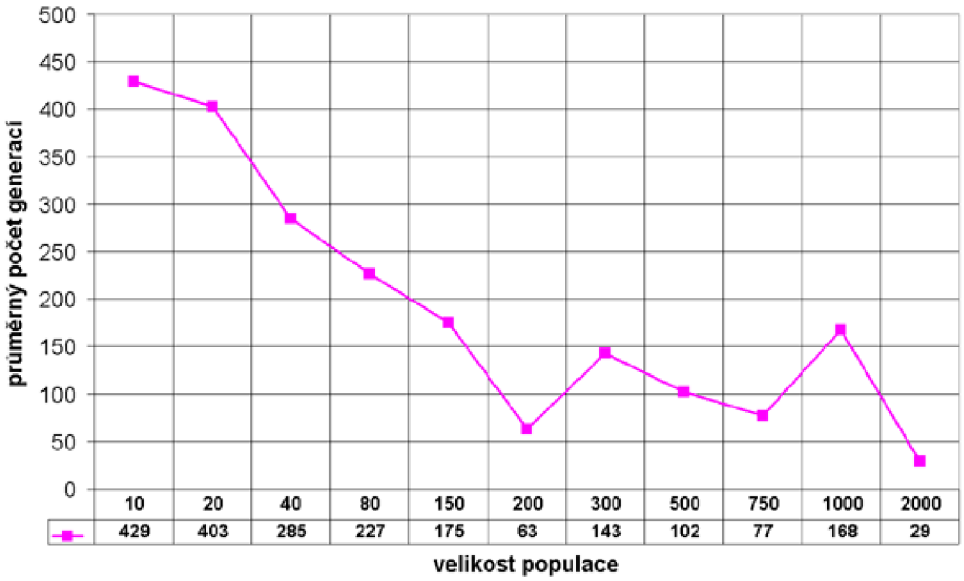
Obrázek 7.1: Grafické znázornění výsledků závislosti celkové úspěšnosti nalezení řešení na velikosti populace genetického algoritmu.

7.2 Pravděpodobnost mutace přechodových funkcí

Operátor mutace patří k nedílným součástem genetického algoritmu. Vnáší do celé evoluce rozmanitost chromozomů a může zabránit případné konvergenci k lokálnímu minimu či maximu. Mutace je v našem případě rozdělena na dvě části. Parametr, který nyní podrobíme testování, se týká lokálních přechodových pravidel výsledného celulárního automatu. Jak jsme uvedli výše, mutace je aplikována na drobnější struktury - přepisovací pravidlo lokální přechodové funkce jedné buňky automatu.

Operátor mutace je vždy aplikován s určitou pravděpodobností na nového jedince. Výsledky testů jsou vyneseny v grafech na obrázcích 7.3 a 7.4 (100 nezávislých experimentů pro každé nastavení parametru). Z nich je patrný jasný vliv a potřeba aplikování tohoto operátoru. Ke stabilizaci dochází kolem hodnoty 40%. Vyšší hodnoty potom přinášejí velmi podobné výsledky. Doporučené jsou tedy hodnoty od 40% výše.

Poměrně zajímavým úkazatelem je průměrný počet generací, za které byl vygenerován úspěšný jedinec v případě, kdy operátor mutace nebyl vůbec aplikován, tedy pravdě-



Obrázek 7.2: Grafické znázornění průměrného počtu generací potřebného k nalezení úspěšného řešení pro testované velikosti populace genetického algoritmu.

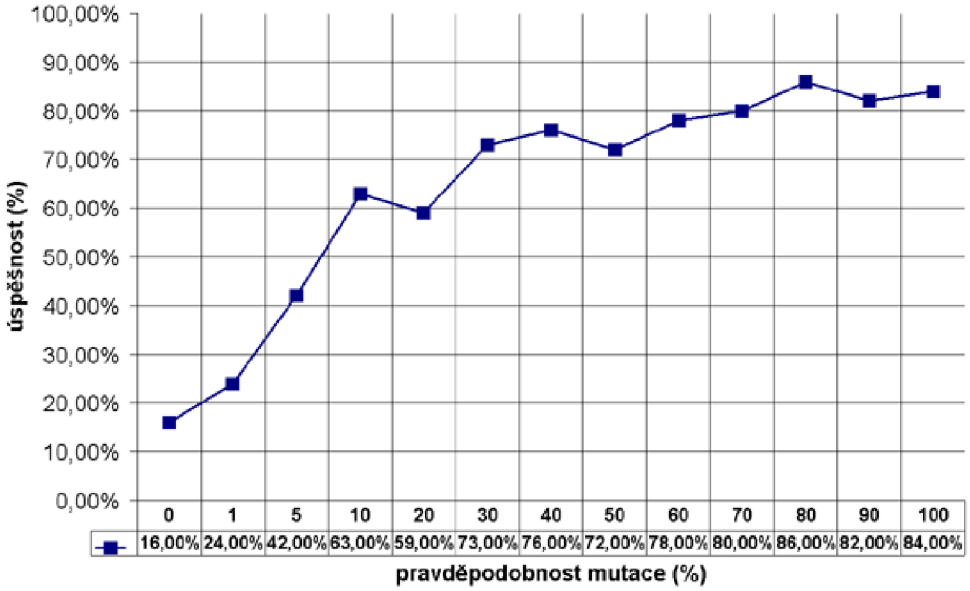
podobnost byla 0% (obrázek 7.4). Mohlo by se zdát, že je algoritmus v tomto případě schopen nalézt řešení daleko rychleji než za jiných okolností. Je však třeba vzít v úvahu výsledek z grafu na obrázku 7.3, pro tu samou pravděpodobnost, kde vidíme výrazně nižší úspěšnost algoritmu. V tomto malém množství případů nedošlo pravděpodobně ke konvergenci řešení a algoritmus měl možnost ve velmi nízkém počtu generací nalézt optimální výsledek. V ostatních případech již nebyl úspěšný, jelikož rozmanitost populace byla na velmi nízké úrovni a neexistoval způsob, jak tuto rozmanitost zajistit. V komplikovanějších situacích lze očekávat, že by úspěšnost za těchto podmínek ještě výrazně poklesla a proto se hodnota pravděpodobnosti `0%` jeví jako nepoužitelná.

7.3 Počet jedinců v turnajové selekci

Jako selekční metodu v genetickém algoritmu, jsme již dříve vybrali turnajovou selekci. V ní vstupuje do tzv. turnaje několik náhodně vybraných jedinců a vyhrává ten s nejlepším ohodnocením. Poměrně často se používá množina o dvou prvcích.

Výsledky nám shrnují grafy na obrázcích 7.5 a 7.6 (100 nezávislých experimentů pro každé nastavení parametru). Velmi důležitým závěrem je skutečnost, že aplikování této selekční metody v její nejjednodušší variantě (2 soutěžící jedinci), má velmi výrazný vliv na úspěšnost algoritmu. Jednoprvková množina ve své podstatě reprezentuje eliminaci této metody a představuje zcela náhodné vybírání jedinců do reprodukčního procesu.

Na druhou stranu je však patrné, že objemnější množiny soutěžících jedinců nevedou již k žádnému výraznému zlepšení. Z tohoto důvodu, a také proto, že nárůst časové náročnosti je přímo úměrný počtu soutěžících jedinců, je doporučená hodnota 2.



Obrázek 7.3: Graf znázorňuje vliv pravděpodobnosti aplikování operátoru mutace na celkovou úspěšnost algoritmu. Mutace se týká lokálních přechodových funkcí výsledného celulárního automatu

7.4 Pravděpodobnost mutace počátečního stavu

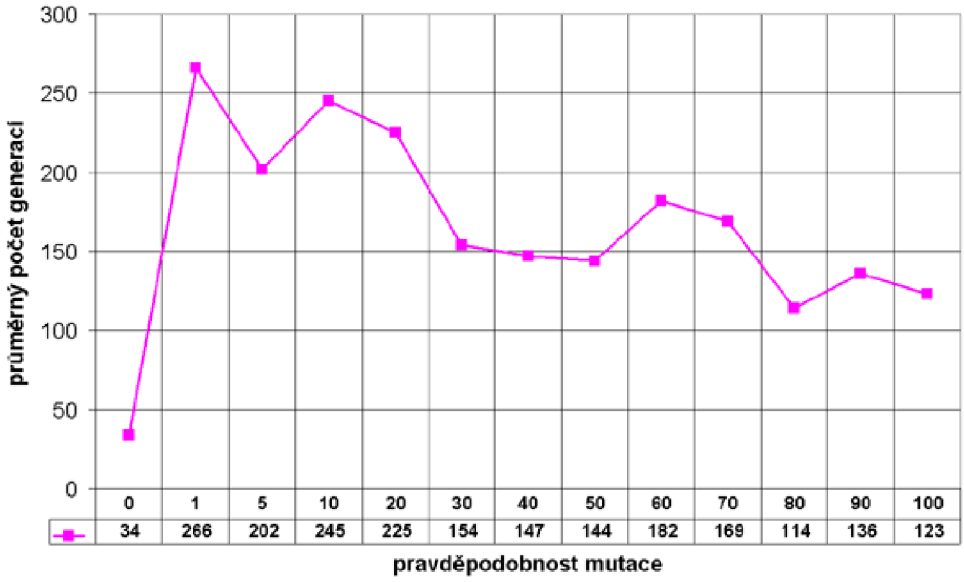
Jak jsme uvedli výše, operátor mutace se dělí na dvě části. První část již byla otestována a popsána. Část druhou tvoří aplikování operátoru mutace na počáteční stav výsledného celulárního automatu. Operace je opět prováděna na úrovni přepisovacích pravidel lokálních přechodových funkcí. Pokud s určitou pravděpodobností dojde k aplikování operátoru, je změněn výchozí stav pouze pro jednu buňku automatu.

Z grafu na obrázku 7.7 je patrné, že na úspěšnost algoritmu má tento parametr poměrně zanedbatelný vliv (100 nezávislých experimentů pro každé nastavení parametru). Nedochází k výraznějším výkyvům či anomáliím. Zajímavější jsou hodnoty z grafu na obrázku 7.8. Zde můžeme pozorovat významné zpomalení nalezení řešení, s narůstající pravděpodobností aplikování operátoru mutace na počáteční stav automatu. V konečném důsledku lze s vyššími hodnotami očekávat i pokles úspěšnosti algoritmu. Z tohoto důvodu bude vhodné omezit operátor mutace jen na lokální přechodové funkce a tento neaplikovat.

7.5 Počet přepisovacích pravidel lokálních přechodových funkcí

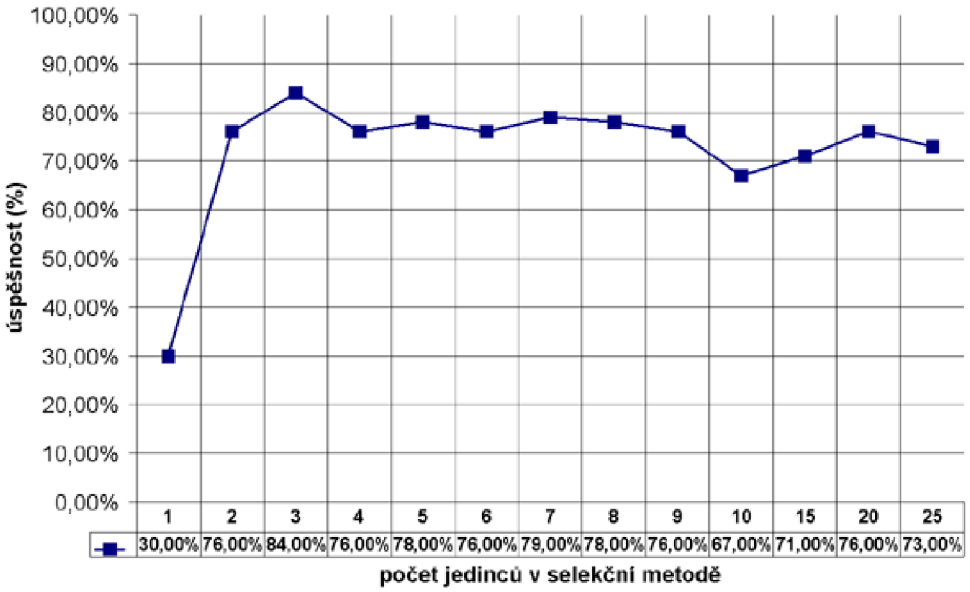
Maximální počet přepisovacích pravidel lokálních přechodových funkcí je velmi závislý na velikosti okolí buněk celulárního automatu. V případě velkého okolí může být tento počet poměrě vysoký, čímž se zvyšuje paměťová i časová náročnost celého algoritmu. Z těchto důvodů je žádoucí výsledný počet přepisovacích pravidel pro jednu buňku automatu omezit a ostatním nedefinovaným stavům přidělit jedno výchozí pravidlo.

Počet přepisovacích pravidel lokálních přechodových funkcí byl předmětem následujícího testu, jehož grafické znázornění výsledků můžeme vidět na obrázcích 7.9 a 7.10 (100 nezávislých experimentů pro každé nastavení parametru). Zde můžeme pozorovat výrazné zhoršení

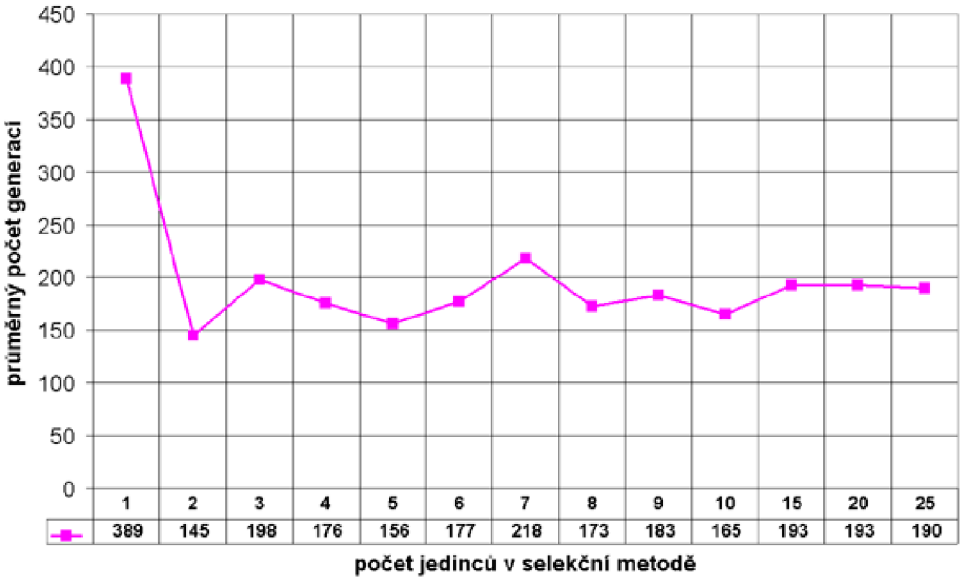


Obrázek 7.4: Grafické znázornění průměrného počtu generací potřebného k nalezení úspěšného řešení při různých hodnotách pravděpodobnosti mutace lokálních přechodových funkcí.

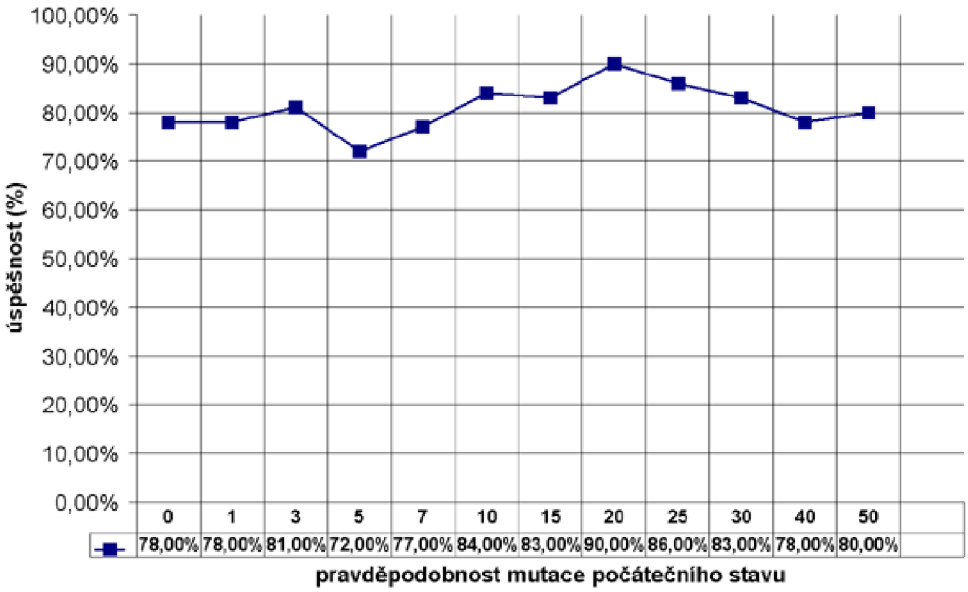
účinnosti algoritmu s klesajícím počtem pravidel. Zde se lze domnívat, že tato skutečnost je způsobena narůstajícím vlivem výchozího pravidla. S klesajícím počtem pravidel dochází k zvyšování pravděpodobnosti aplikování výchozího pravidla, tedy pro danou buňku vygenerování vždy stejného hradla se stejnými vstupy. Z tohoto důvodu je vhodné zachovávat vysoký poměr individuálních pravidel k výchozímu.



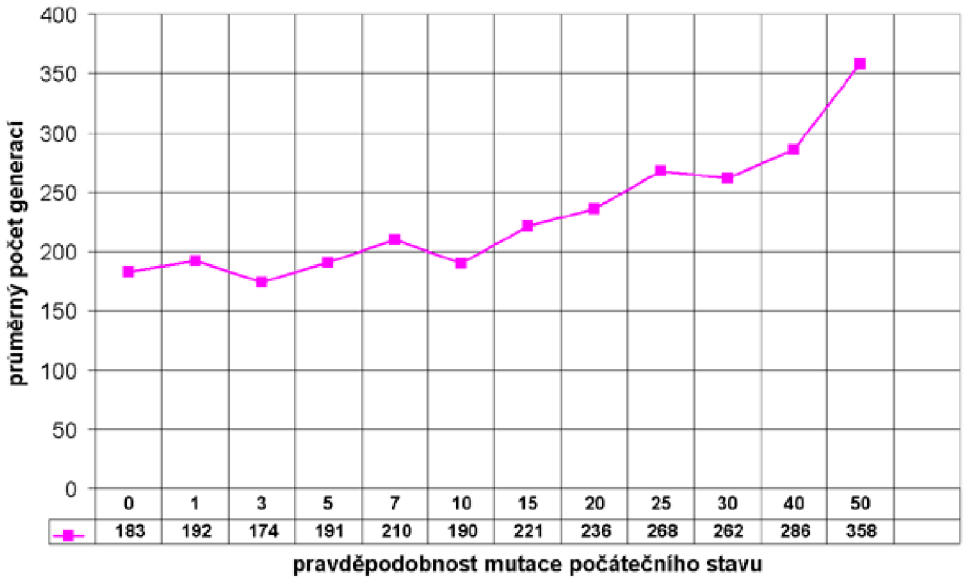
Obrázek 7.5: Graf vykresluje závislost úspěšnosti algortimu na počtu jedinců, kteří jsou vybráni do turnajové selekce.



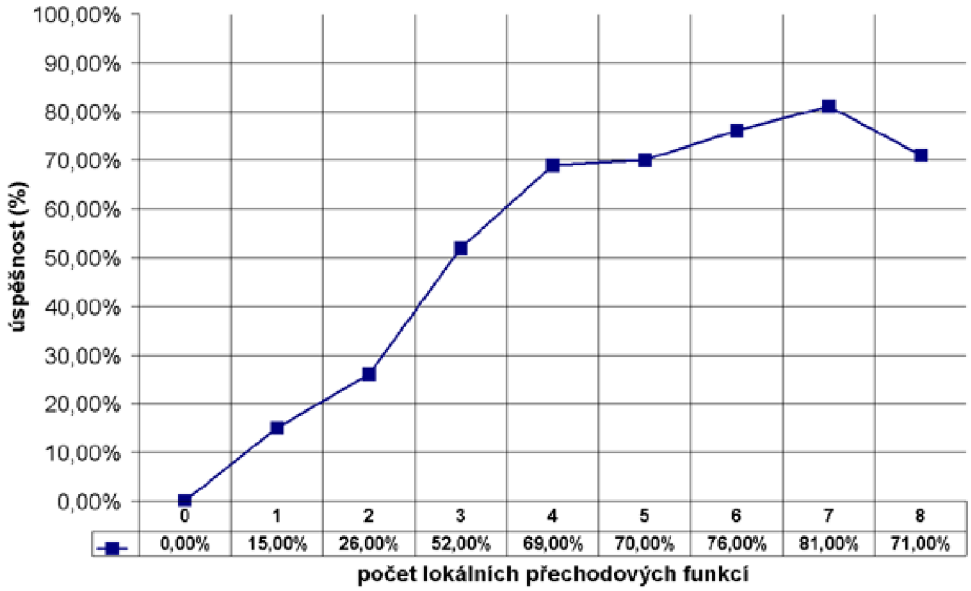
Obrázek 7.6: Grafické znázornění průměrného počtu generací potřebného k nalezení úspěšného řešení při různých velikost množin turnajové selekce.



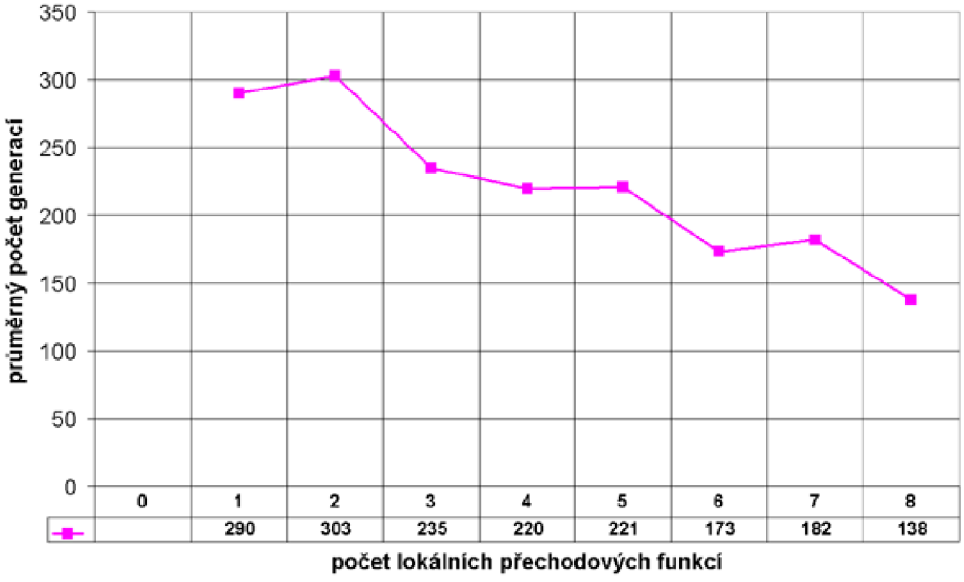
Obrázek 7.7: Graf znázorňuje vliv pravděpodobnosti aplikování operátoru mutace na celkovou úspěšnost algoritmu. Mutace se v tomto případě týká počátečního stavu výsledného celulárního automatu.



Obrázek 7.8: Grafické znázornění průměrného počtu generací potřebného k nalezení úspěšného řešení při různých hodnotách pravděpodobnosti mutace počátečního stavu celulárniho automatu.



Obrázek 7.9: Grafické znázornění výsledků závislosti celkové úspěšnosti nalezení řešení na počtu přepisovacích pravidel lokálních přechodových funkcí.



Obrázek 7.10: Grafické znázornění průměrného počtu generací potřebného k nalezení úspěšného řešení pro testované počty přepisovacích pravidel lokálních přechodových funkcí.

Kapitola 8

Experimentální výsledky

Výše uvedený algoritmus nám umožňuje zkoumat poměrně širokou škálu různorodých obvodů. Teoreticky lze říci, že jej lze použít na jakýkoliv logický obvod u kterého známe odezvy na jednotlivé kombinace vstupních hodnot. Prakticky však tato skutečnost neznamená, že bude nalezeno požadované řešení. Aby mohla být stanovena fitness hodnota každého jedince, je třeba ověřit reakci obvodu na všechny vstupní hodnoty. Se zvyšujícím se počtem vstupů obvodu tak dochází k poměrně velkému nárustu časové složitosti této operace a tedy i celého algoritmu. U složitějších obvodů je nutno počítat s faktem, že počet generací potřebný k nalezení požadovaného řešení bude narůstat. Se zvyšující se složitostí obvodu narůstá jak časová složitost vývoje jedné generace, tak potřeba zvyšovat maximální počet generací, po který bude daný obvod evolvován. Časová složitost je tak hlavním omezujícím faktorem prováděných experimetů. Z tohoto důvodu byly některé experimenty ukončeny ve chvíli, kdy by následující rozšíření testovaného typu obvodu přesáhl realizovatelnou hranici.

V rámci experimentů bylo pro konstrukci kombinačních logických obvodů využito následujících hradel: AND, OR, XOR, NOT, NAND, NOR, NXOR. Jednotlivé uspořádání těchto prvků je dáno výsledným celulárním automatem, který pokud je výsledek úspěšný, slouží jako generátor požadovaného řešení. U některých obvodů existuje určitý výčet hradel, která se pro daný typ obvodu obvykle používají. V těchto speciálních případech byly využity jen hradla z tohoto výčtu. U případů, kterých se tato skutečnost týká budou použitá hradla uvedeny.

Do experimentů byly vybrány následující typy kombinačních logických obvodů:

- sčítačky
- násobičky
- boolovská symetrie
- řadicí sítě
- paritní obvody
- mediánové obvody
- polymorfní obvody

Pro každý typ obvodu byla provedena řada nezávislých experimentů. Přesný počet nebyl předem stanoven a záleželo zejména na časové složitosti jednotlivých pokusů. Pro výchozí

nastavení algoritmu byly použity závěry z předcházející kapitoly. V řadě případů docházelo k experimentálním korekcím s cílem dosáhnout lepších výsledků a efektivnosti algoritmu pro daný typ obvodu. Celkem bylo provedeno na 21493 experimentů, které souhrnně trvaly přibližně 1698 hodin. Jednotlivé experimenty lze rozdělit dle jejich časové náročnosti do několika segmentů, jak ukazuje tabulka 8.1. V následujících kapitolách uvedeme výsledky experimentů pro jednotlivé typy obvodů. Dále provedeme srovnání s prací zaměřenou na stejnou problematiku a využívající uniformních celulárních automatů [13].

časová náročnost evoluce	provedené experimenty	úspěšné experimenty
$T(e) <= 2$ min.	16110	15298
2 min. $> T(e) <= 10$ min.	2605	1108
10 min. $> T(e) <= 30$ min.	1837	153
$T(e) > 30$ min.	941	70

Tabulka 8.1: Segmentace experimentů dle jejich časové náročnosti. $T(e)$ představuje celkový čas evoluce daného pokusu.

8.1 Sčítačky

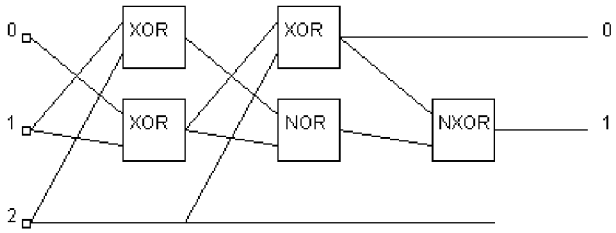
Sčítačky patří mezi základní kombinační logické obvody. Uspořádání hradel představuje u sčítaček poměrně rozmanitou strukturu a řadí je tak mezi náročnější obvody. Nejjednodušším obvodem byla v experimentech jednobitová sčítačka se kterou neměl algoritmus výraznější problémy. S narůstajícím počtem vstupů se zvyšovaly časové nároky na jednotlivé testy a celková úspěšnost algoritmu měla klesavou tendenci. Komplexní přehled o provedených experimentech podává tabulka 8.2.

typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [h]
1 b. sčítačka	1100	100,00%	< 1 h
1 b. sčítačka s přenosem	283	76,68%	1 h
2 b. sčítačka	365	26,58%	62,4 h
2 b. sčítačka s přenosem	386	1,30%	221,2 h
3 b. sčítačka	173	0,00%	309,4 h

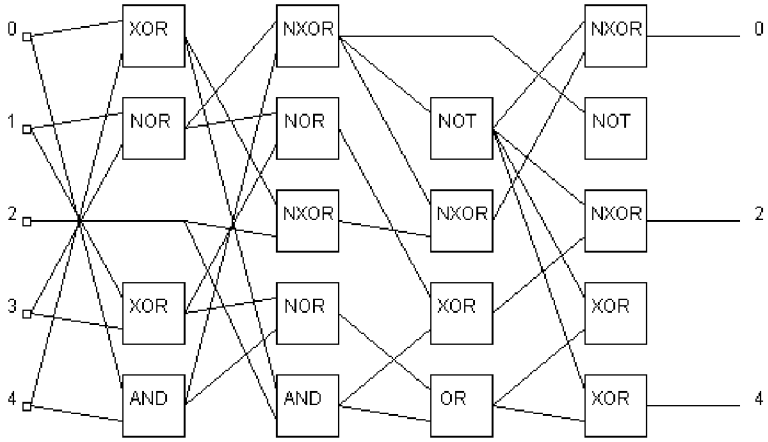
Tabulka 8.2: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných sčítaček.

Dvoubitová sčítačka s přenosem, představuje nejsložitější funkční obvod, který byl vygenerován. V případě experimentů s uniformním celulárním automatem [13] měl výsledný obvod přesně definovány pozice výstupních bitů (0, 2, 4). Toto uspořádání plynulo z praktických zkušeností, kdy se ukázalo, že právě tato konfigurace generuje většinu úspěšných řešení. V našem případě dokázal algoritmus vygenerovat jak obvod se zmíněným rozvržením výstupu (obrázek 8.2), tak i s výstupy jiné konfigurace (obrázek 8.3). Na obrázku 8.1 je pak uvedeno jedno z úspěšných řešení jednobitové sčítačky s přenosem.

V oblasti tříbitových sčítaček byly prováděny časově velmi náročné experimenty. I přes relativně velký počet experimentů, se nepodařilo nalézt plně funknční obvod. Nejlepší řešení,



Obrázek 8.1: Jednobitová sčítačka s přenosem

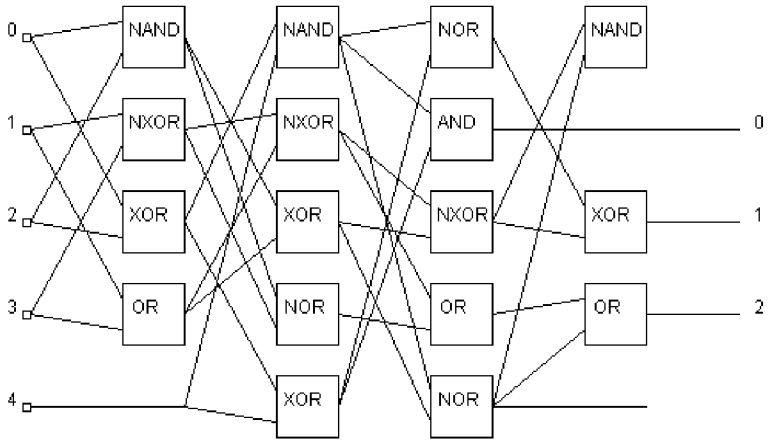


Obrázek 8.2: Dvoubitový sčítačka s přenosem s pevně definovaným výstupem (0, 2, 4)

které bylo vygenerováno, obsahovalo na výstupu odchylku 4 bity. Textová reprezentace tohoto obvodu je uvedena na obrázku 8.4.

8.2 Násobičky

V případě násobiček se podařilo pomocí evolučního algoritmu vygenerovat maximálně čtyřvstupovou verzi (příklad na obrázku 8.5). Úspěšnost nalezení těchto obvodů, jak naznačuje tabulka 8.3, byla vzhledem k časové náročnosti poměrně vysoká. Experimenty se složitějšími obvody požadovaly poměrně vysoké časové nároky. V oblasti tříbitových násobiček se nepodařilo nalézt plně funkční řešení. Také odchylky jednotlivých experimentů od požadovaného řešení byly poměrně velké. Toto může být zapříčiněno poměrně složitou strukturou propojení a počtu hradel, jež jsou charakteristické pro složitější obvody, kterými násobičky jsou.



Obrázek 8.3: Další varinta dvoubitové sčítačky s přenosem. Tentokrát bylo rozvržení výstupu součástí evoluce.

```
OR_0 {2, 5} ;NXOR_1 {1, 4} ;XOR_2 {0, 3} ;NXOR_3 {2, 5} ;AND_4 {0, 3} ;AND_5 {1, 4} ;
OR_0 {0, 5} ;line_1 {1, 2} ;line_2 {2, 4} ;NXOR_3 {3, 5} ;NOT_4 {4, 5} ;AND_5 {1, 4} ;
AND_0 {0, 3} ;NOR_1 {0, 3} ;line_2 {2, 4} ;NOR_3 {4, 5} ;AND_4 {0, 3} ;NAND_5 {1, 4} ;
AND_0 {0, 3} ;NOR_1 {0, 3} ;line_2 {2, 4} ;NXOR_3 {3, 5} ;NOR_4 {0, 1} ;XOR_5 {0, 4} ;
AND_0 {0, 3} ;NOR_1 {0, 3} ;(out)line_2 {2, 4} ;(out)NXOR_3 {3, 5} ;(out)NOR_4 {0, 1} ;(out)XOR_5 {0, 4} ;
```

Obrázek 8.4: Textová reprezentace nejlepšího řešení v experimentech s tříbitovými sčítačkami. Obvod má odchylku ve 4 bitech. Každá ředa představuje jeden stupeň kombinačního logického obvodu. V závorkách jsou uvedeny čísla vstupů vázaných na předchozí stupeň obvodu. Hradla označená příznakem OUT označují výstupy obvodu.

8.3 Boolovská symetrie

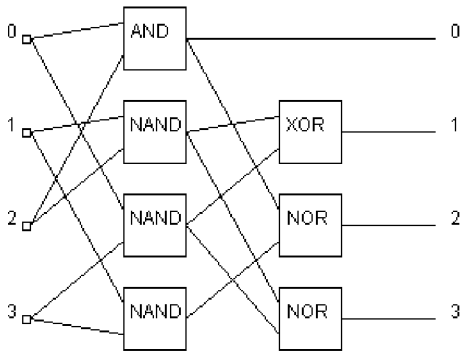
Boolovská symetrie ověřuje, zda jsou hodnoty vstupu symetrické podle středu. Pokud je tato podmínka splněna, je výstupem obvodu `1` v opačném případě je výstup nulový. V této třídě obvodů se podařilo vygenerovat poměrně kvalitní řešení i u vícevstupových variant, než v předcházejících případech. Výraznějších komplikací dosáhl algoritmus až u osmivstupového obvodu, avšak i tento případ se podařilo vygenerovat a stal se tak nejsložitější nalezenou boolovskou symetrií. Celkový přehled úspěšnosti experimentů nám ukazuje tabulka 8.4.

V případě uniformního celulárního automatu [13] byl nejúspěšnějším vygenerovaným obvodem sedmivstupová varianta boolovské symterie. Po odstranění redundantních komponent, obsahoval výsledný obvod 8 hradel. Sedmibitové boolovské symetrie byly sice v našem případě časově náročnější, avšak úspěšnost algoritmu byla velmi vysoká. Bylo také nalezeno mnohem vhodnější řešení než v případě uniformního celulárního automatu.

Na obrázku 8.6 je jedno z velmi kvalitních řešení sedmibitové boolovské symetrie. Po eliminaci redundantních komponent, dostáváme obvod se 7 hradly. Prostřední vstupní bit obvodu není zcela logicky využit, jelikož nemá na výsledek funkce vliv. Z obrázku je patrné využití dvou hradel NOT. Jelikož na výstupech, na které jsou tyto dvě hradla vázána, nejsou závislá žádná jiná hradla, lze provézt přesun obou negací do předcházejícího kroku.

typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [ht]
2 b. násobička	136	34,56%	39,8 h
3 b. násobička	117	0,00%	118,7 h

Tabulka 8.3: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných násobiček.



Obrázek 8.5: Dvoubitová násobička

Výsledný obvod je uveden na obrázku 8.7.

Řešení představuje sedmibitovou boolovskou symetrii s 5 hradly a 3 stupni obvodu. Význam jednotlivých částí je zřejmý. Hradla XOR slouží jako komparátor symetricky umístěných hodnot. Pokud je na jakémkoli výstupu tohoto stupně `1`, je celkový výsledek funkce nulový. Zbývající dvě hradla slouží pro přenos výsledku na výstup.

Oproti uniformnímu celulárnímu automatu, se povedlo vygenerovat plně funkční řešení, které obsahuje jeden vstupní bit navíc. Výsledný obvod po odstranění redundantních komponent je znázorněn na obrázku 8.8. Opět lze provézt drobnou úpravu a přesunout logický význam hradla NOT do předcházejícího stupně. Dostáváme tak řešení s 8 hradly.

8.4 Řadicí sítě

Řadicí sítě tvoří první typ obvodu u kterého omezíme použitá hradla. Pro výpočet minima a maxima se běžně používájí logické obvody AND a OR. Z tohoto důvodu bylo těchto hradel využito v převážné většině experimentů. U obvodů s menším počtem vstupů byly provedeny také pokusy s kompletní množinou implementovaných hradel. Na obrázku 8.9 je uveden příklad takovéto plně funkční řadicí sítě. Algoritmus i v těchto případech využil pouze hradel AND a OR a jejich negace.

Tabulka 8.5 poskytuje přehled úspěšnosti experimentů jednotlivých typů řadicích sítí. V některých případech byly prováděny pokusy s parametry genetického algoritmu, které omezily maximální počet generací vývoje. Tato skutečnost se odrazila v poklesu úspěšnosti u některých typů řadicích sítí.

V oblasti řadicích sítí se povedlo vygenerovat poměrně koplikovaná propojení hradel. Srovnáme-li výsledky s uniformním celulárním automatem [13], nepodařilo se nalézt více-

typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [h]
2 b. boolovská s.	3000	100,00%	< 1 h
3 b. boolovská s.	2000	100,00%	< 1 h
4 b. boolovská s.	464	100,00%	< 1 h
5 b. boolovská s.	79	75,95%	5,3 h
6 b. boolovská s.	70	74,29%	40,6 h
7 b. boolovská s.	24	83,33%	17,5 h
8 b. boolovská s.	40	12,50%	63,0 h
10 b. boolovská s.	59	0,00%	95,4 h

Tabulka 8.4: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných boolovských symetrií.

vstupovou řadicí síť, než v tomto případě. V oblasti šestibitových obvodů tohoto typu však bylo vygenerováno řešení, které obsahuje o 2 hradla méně, než v případě experimentů s uniformním celulárním automatem (obráze 8.10).

typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [h]
2 b. řadicí s.	3000	100,00%	< 1 h
3 b. řadicí s.	181	66,30%	3,4 h
4 b. řadicí s.	713	6,59%	18,0 h
5 b. řadicí s.	264	8,33%	68,2 h
6 b. řadicí s.	91	5,49%	61,2 h
7 b. řadicí s.	24	0,00%	55,4 h

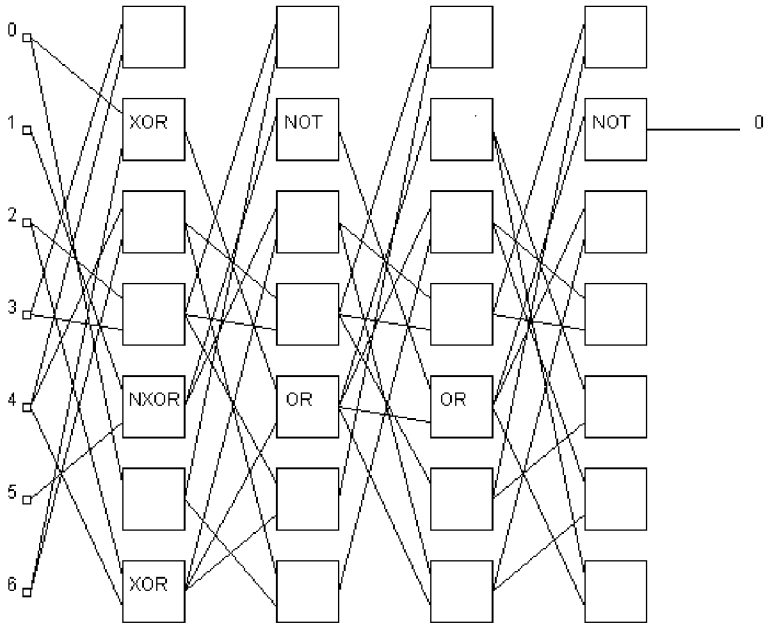
Tabulka 8.5: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných řadicích sití.

8.5 Mediánové obvody

Mediánové obvody jsou principielně velmi podobné řadicím sítím. Funkcí kombinačního logického obvodu je nalézt prostřední prvek v seřazené posloupnosti. Je nutné si uvědomit, že prostřední prvek lze hledat pouze v posloupnostech s lichým počtem prvků. Z tohoto důvodu pracovaly všechny experimetny s mediánovými obvody o lichém počtu vstupů.

Podobně jako u řadicích sítí, byla v případě tohoto typu obvodu omezena množina použitých hradel. Opět byly využity hradla AND a OR, která se běžně používájí pro hledání minima a maxima. Komplexní informace o provedených experimentech jsou uvedeny v tabulce 8.6.

Nejsložitějším vygenerovaným plně funkčním obvodem byl v této třídě sedmibitový mediánový obvod (obrázek 8.12). Srovnání s uniformním celulárním automatem není v tomto případě možné, jelikož mediánové obvody publikované v příslušné práci [13] nejsou plně funkční 8.11. Pro příklad můžeme uvézt některé vstupy na které uvedené obvody reagují chybně. V případě pětivstupého obvodu jde např. o posloupnost `10001` a u sedmivstupové varianty např. `1000011`.



Obrázek 8.6: Úspěšné řešení plně funkční sedmivstupové boolovské symetrie, vygenerované algoritmem po odstranění redundantních komponent

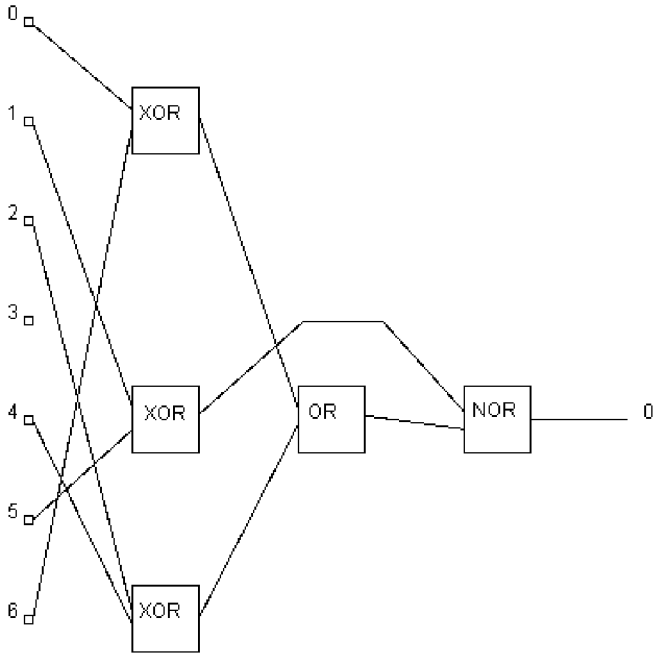
typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [h]
3 b. mediánový o.	1000	100,00%	< 1 h
5 b. mediánový o.	348	24,14%	50,7 h
7 b. mediánový o.	28	3,57%	29,3 h
9 b. mediánový o.	18	0,00%	56,3 h

Tabulka 8.6: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných mediánových obvodů.

8.6 Paritní obvody

Ve této třídě obvodů byly experimenty pro zjednodušení prováděny jen na obvodech sudé parity. Na obrázku 8.13 je uveden osmibitový paritní obvod, který byl vygenerován za použití všech implementovaných hradel. Po odstranění redundantních komponent lze zjistit, že algoritmus ke konstrukci paritního obvodu využil pouze hradla XOR a jeho negace. Tento jev není náhodný, jelikož pro konstrukci paritních obvodů se běžně používá logické funkce XOR. Z tohoto důvodu, bylo pro evoluci náročnějších typů paritních obvodů, použito pouze hradlo XOR.

Z tabulky 8.7 je patrná velmi vysoká úspěšnost algoritmu v této třídě obvodů. Podařilo se bez vážnějších problémů vygenerovat vícevstupové varianty. Můžeme také pozorovat významný nárůst časové náročnosti, která byla v tomto případě hraniční veličinou. Maximální počet vstupů, který byl u daného typu obvodu testován, byl 12. Fuknění řešení dvanáctibitového paritního obvodu je znázorněno na obrázku 8.14.



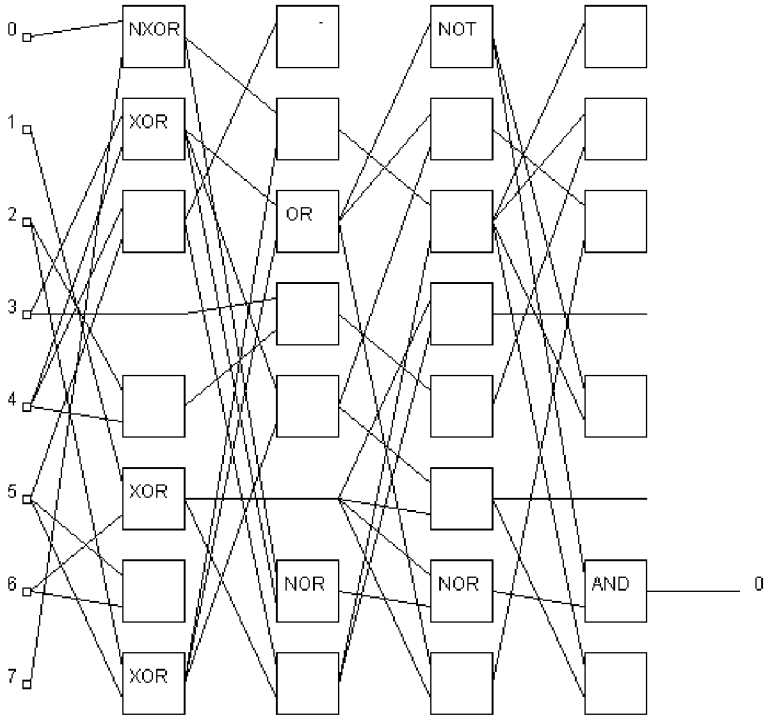
Obrázek 8.7: Upravený obvod z obrázku 8.6. Sedmivstupová boolovská symetrie s 5 hradly.

typ obvodu	provedené experimenty	úspěšnost [%]	časová náročnost [h]
8 b. paritní o.	2022	100,00%	52,6 h
9 b. paritní o.	1188	100,00%	54,0 h
10 b. paritní o.	172	100,00%	53,7 h
11 b. paritní o.	11	100,00%	48,4 h
12 b. paritní o.	3	66,67%	52,6 h

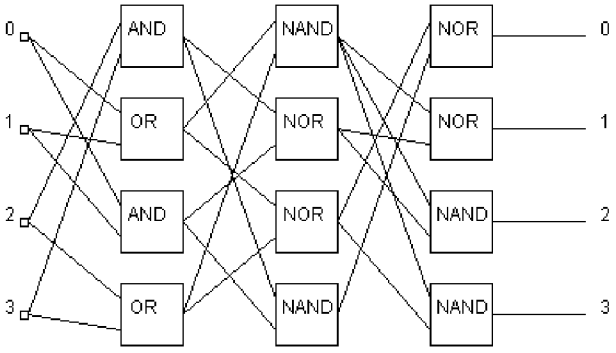
Tabulka 8.7: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných paritních obvodů.

8.7 Polymorfní obvody

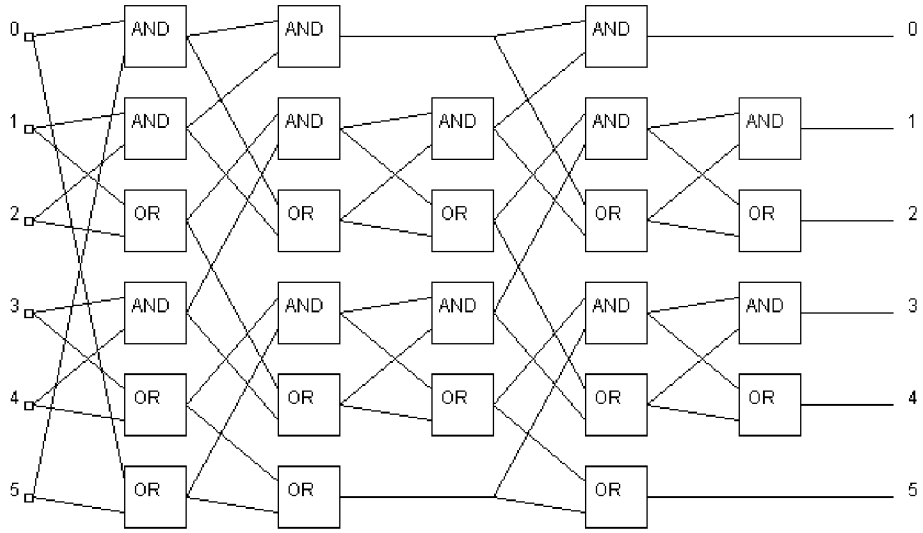
Polymorfní obvody tvoří speciální třídu kombinačních logických obvodů. Jak bylo zmíněno výše, každé hradlo může představovat více funkcí, mezi nimiž je voleno v závislosti na vnějším činiteli. V našem případě, jsme se zaměřili na hradla se dvěmi funkcemi, mezi nimiž je vybíráno, na základě dodatečného binárního vstupu. Do experimentů bylo vybráno několik variant obvodů, některé z nich inspirovány prací pojednávající o návrhu polymorfních obvodů [12]. Experymety byly celkově náročnější, jelikož bylo, na rozdíl od klasických kombinačních logických obvodů, hledáno řešení dvou funkcí. Pro počáteční nastavení algoritmu bylo využito předchozích výsledků. I s těmito složitějšími experimety si v mnoha případech dokázal algoritmus poradit. Souhrnné výsledky ukazuje tabulka 8.8. Příklady plně funkčních vygenerovaných obvodů jsou znázorněny na obrázcích 8.15, 8.16, 8.17, 8.18, 8.19, 8.20, 8.21.



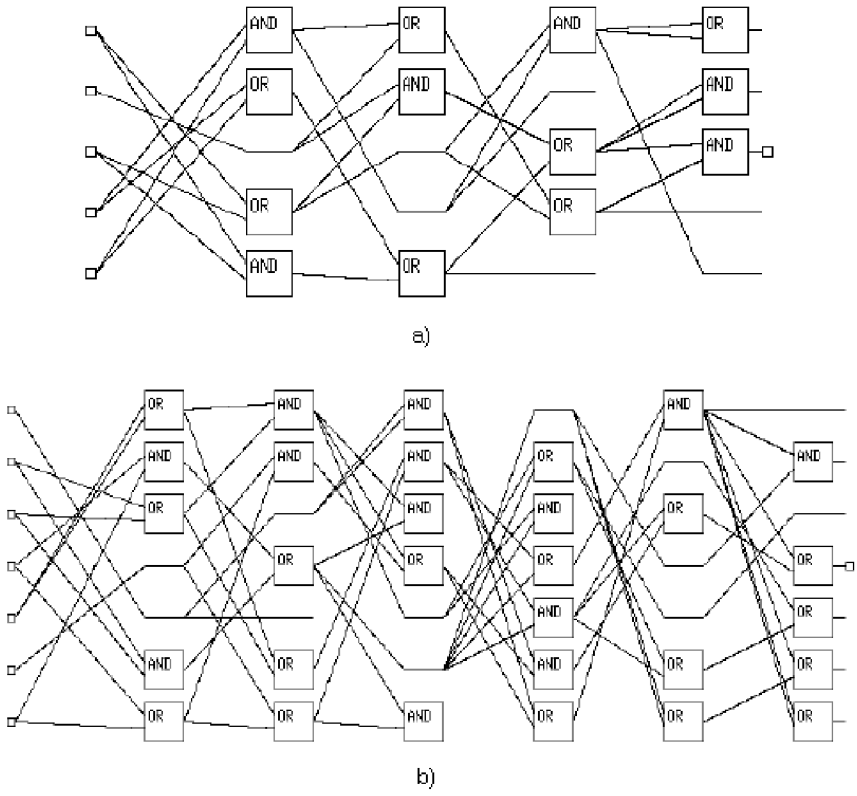
Obrázek 8.8: Osmibitová boolovský symetrie po odstranění redundantních komponent.



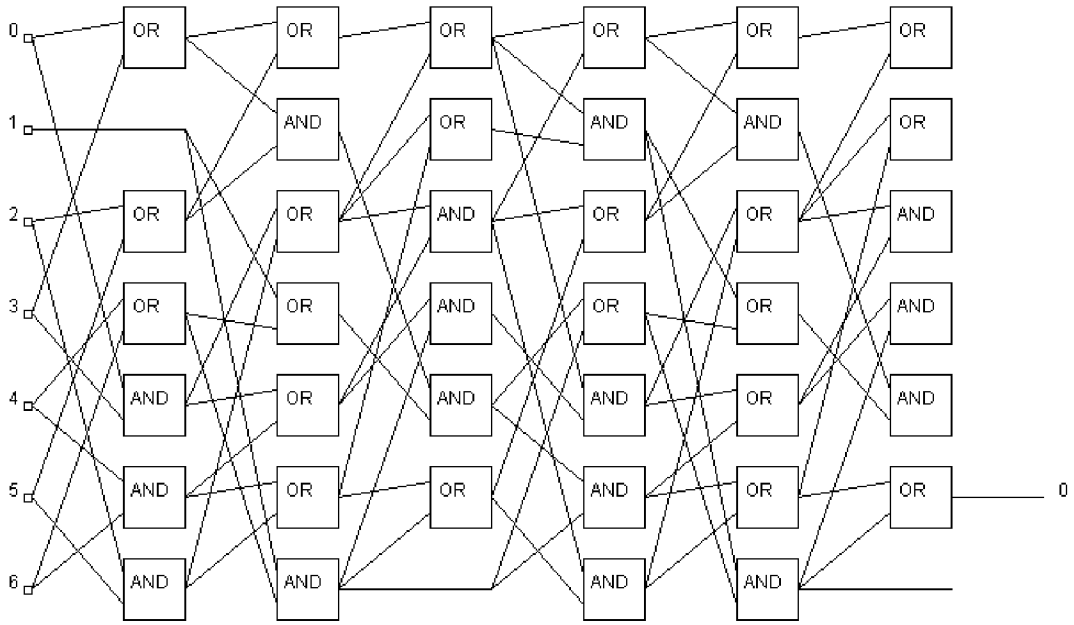
Obrázek 8.9: Čtyřbitová řadicí síť vygenerovaná za použití všech implementovaných hradel



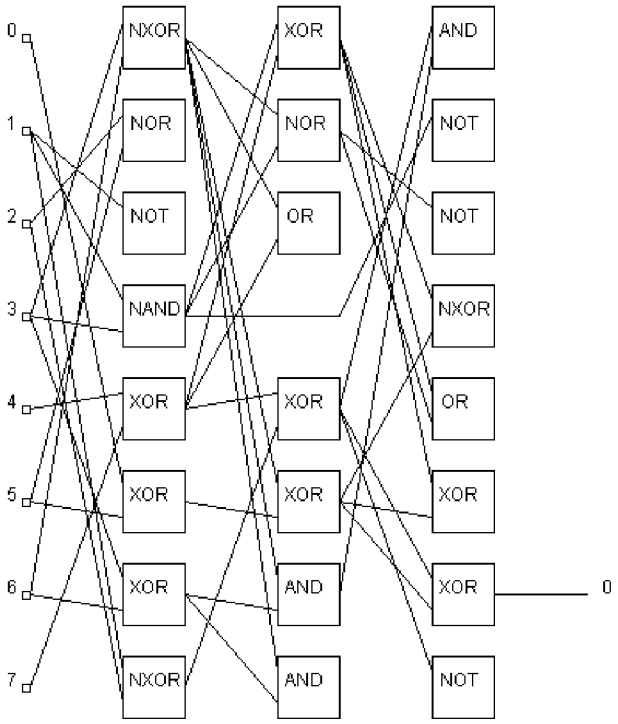
Obrázek 8.10: Příklad jednoho z nejúspěšnějších vygenerovaných řešení - šestibitová řadicí síť



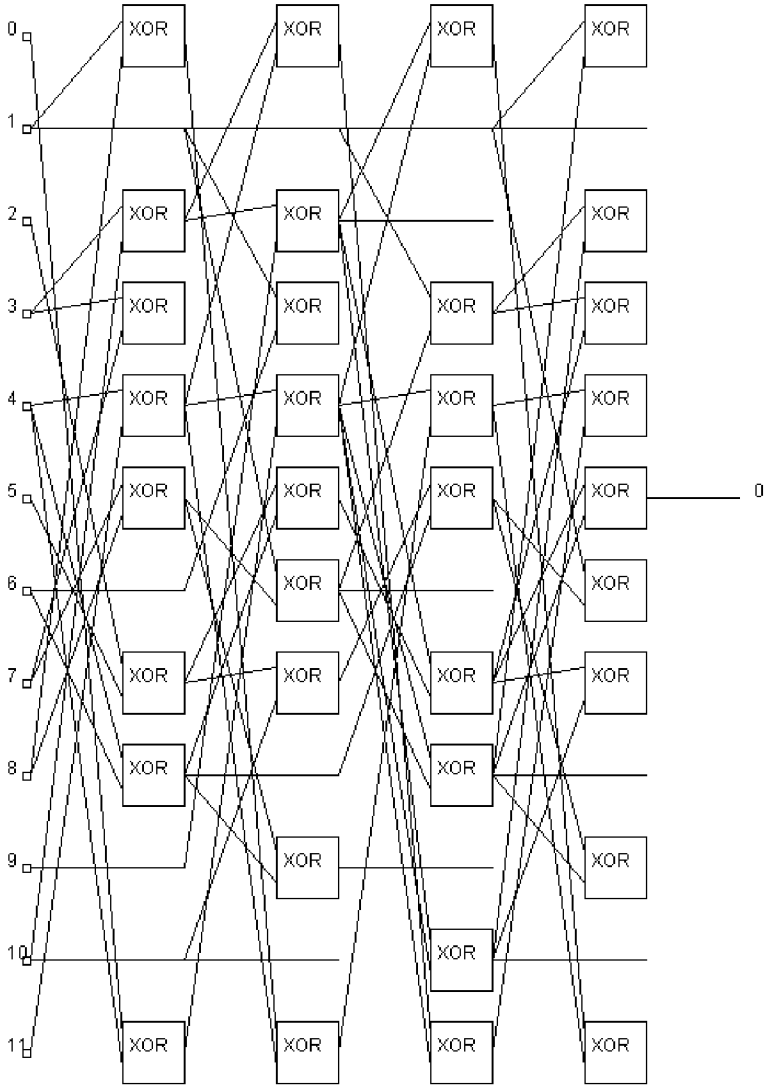
Obrázek 8.11: Chybně vygenerované obvody z [13]: a) pětivstupový mediánový obvod, b) sedmivstupový mediánový obvod.



Obrázek 8.12: Sedmibitový mediánový obvod



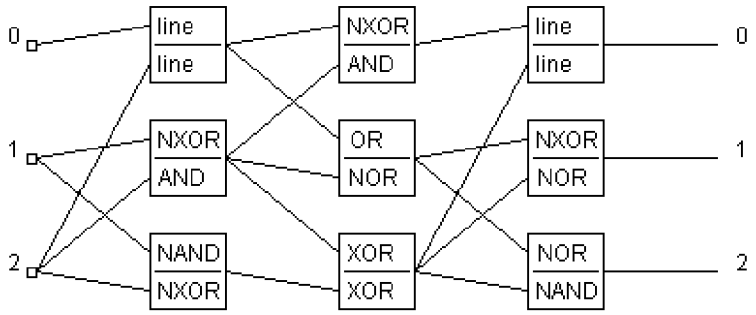
Obrázek 8.13: Osmibitový paritní obvod s využitím všech implementovaných hradel



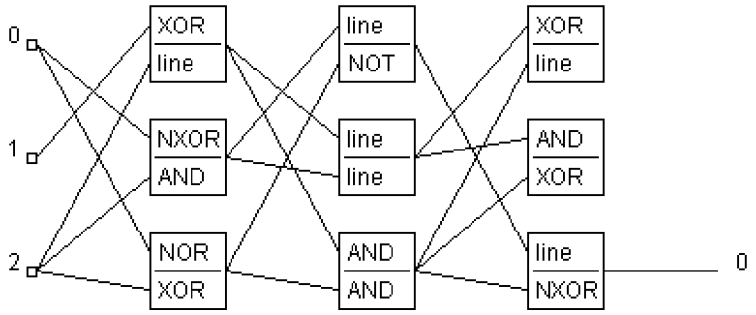
Obrázek 8.14: Dvanáctibitový paritní obvod

typ obvodu	provedené exp.	úspěšnost [%]	č. náročnost [h]
1 b. úplná sčítačka / 3 b. řadicí s.	1124	0,53%	44,7 h
2 b. sčítačka / 2 b. násobička	210	0,00%	43,8 h
2 b. sčítačka / 4 b. řadicí s.	209	0,00%	42,7 h
3 b. boolovská s. / 3 b. mediánový o.	1000	92,20%	4,7 h
5 b. boolovská s. / 5 b. mediánový o.	92	1,09%	43,9 h
5 b. boolovská s. / 5 b. paritní o.	164	11,59%	43,8 h
3 b. mediánový o. / 3 b. paritní o.	1000	93,70%	4,4 h
5 b. mediánový o. / 5 b. paritní o.	92	1,09%	43,5 h
2 b. násobička / 4 b. řadicí s.	283	0,00%	44,1 h
6 b. paritní o. / 6 b. boolovská s.	51	0,00%	42,9 h
7 b. paritní o. / 7 b. boolovská s.	19	10,53%	42,9 h

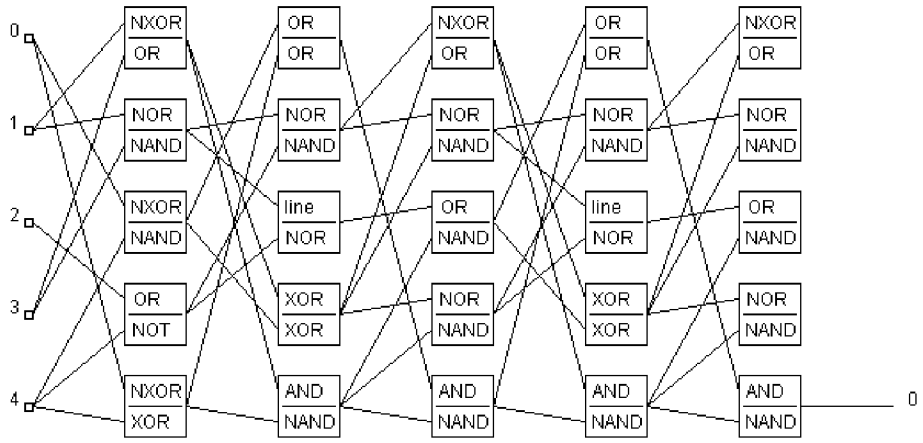
Tabulka 8.8: Počet experimentů, úspěšnost algoritmu a časová náročnost pro jednotlivé typy zkoumaných polymorfních obvodů.



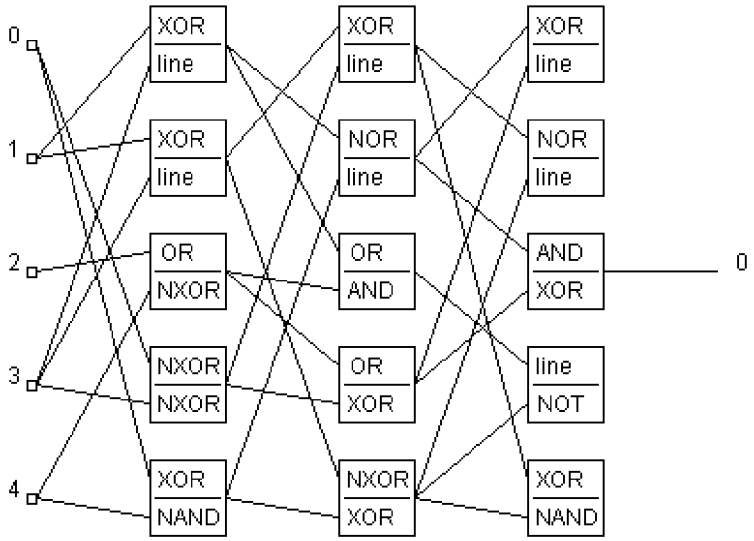
Obrázek 8.15: Jednobitová sčítačka s přenosem (0) / tříbitová řadicí síť (1)



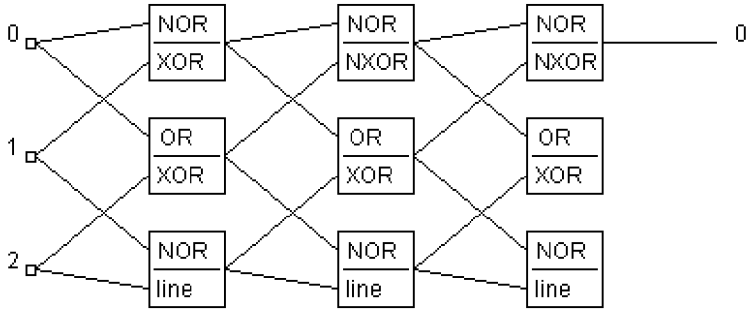
Obrázek 8.16: Tříbitová boolovský symetrie (0) / tříbitový mediánový obvod (1)



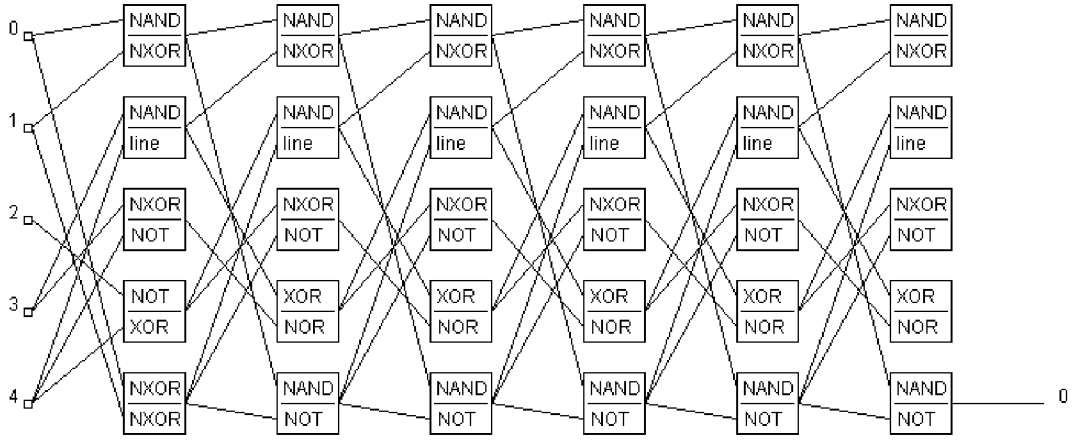
Obrázek 8.17: Pětibitová boolovský symetrie (0) / pětibitový mediánový obvod (1)



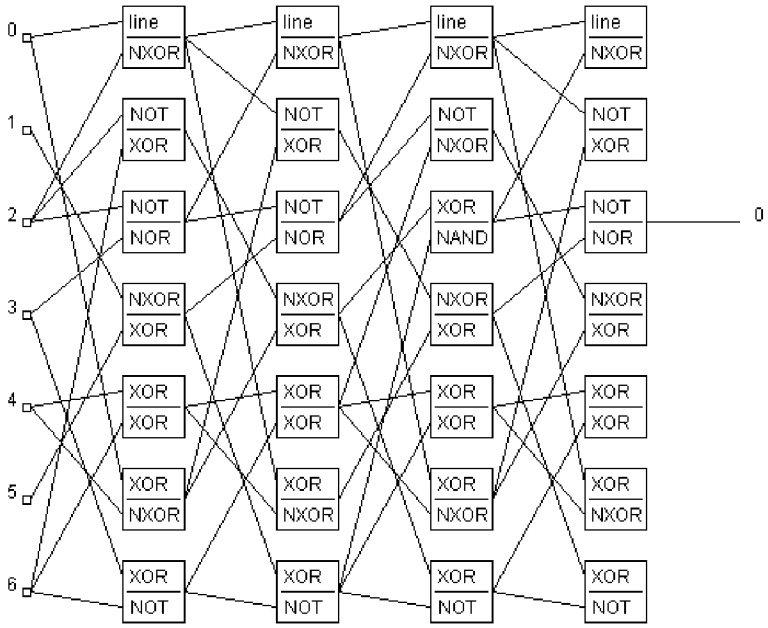
Obrázek 8.18: Pětibitová boolovský symetrie (0) / pětibitový paritní obvod (1)



Obrázek 8.19: Tříbitový mediánový obvod (0) / tříbitový paritní obvod (1)



Obrázek 8.20: Pětibitový mediánový obvod (0) / pětibitový paritní obvod (1)



Obrázek 8.21: Sedmibitový paritní obvod (0) / sedmibitová boolovská symetrie (1)

Kapitola 9

Diskuze

V předcházejících částech jsme prověřili konstrukci kombinačních logických obvodů za pomoci developmentu založeném na vývoji binárního jednorozměrného neuniformního celulárního automatu. Prezentované výsledky ukazují, že implementovaný algoritmus je do jisté míry schopen nalézt řešení zadané úlohy.

Ve třídě obvodů s jednodušší sítí hradel, se podařilo nalézt řešení i pro výrazně vyšší počet vstupů. U obvodů se složitější strukturou bylo mnohem náročnější vygenerovat odpovídající řešení. Celkově se podařilo v této kategorii nalézt dvoubitovou sčítačku s přenosem a dvoubitovou násobičku. Ve třídě polymorfních obvodů byla situace ještě více komplikovanější. Polymorfní obvod reprezentuje dvě funkce zároveň. I přesto dokázal algoritmus nalézt některé varianty z této specifické třídy.

Nelze jednoznačně říci, že pro náročnější zadání není algoritmus schopen nalézt plně funkční řešení. Základním omezujícím faktorem prováděných experimentů byla časová náročnost. Se zvyšujícím se počtem vstupů obvodů narůstaly také časové požadavky na jednotlivé pokusy. V některých třídách obvodů, bylo velmi komlikované provést dostatečný počet experimetů, natož otestovat správnost počátečního nastavení algoritmu. Ve srovnání s uniformním celulárním automatem, je neuniformní verze daleko komplikovanější. Neobsahuje jednu společnou lokální přechodovou funkci a velikost vyhledávacího prostoru je tak mnohem větší. Na druhou stranu nám tato vlastnost dává daleko větší možnosti v konstrukci výsledného kombinačního loického obovdu. Ukázalo se, že algoritmus s neuniformní verzí celulárního automatu dokáže generovat stejně kvalitní a v řadě případů i lepší řešení, než v případě uniformního celulátního automatu.

Výrazným činitelem ovlivňující časovou náročnost je výpočet fitness hodnoty. Tato je stanovena na základě porovnání všech vstupních a odpovídajích výstupních hodnot. I přesto, že vyhodnocování reakce na dané vstupy bylo pro jednotlivé stupně obvodu prováděno paralelně, byly časové nároky znatelné. V této oblasti by bylo vhodné experimentovat s fitness funkcemi založenými na jiných principech, tak aby se značně redukovala časová složitost.

U tříd obvodů jakými jsou např. sčítačky a násobičky by bylo vhodné provést větší množství experimetů se zaměřením na počáteční nastavení algoritmu. Vliv jednotlivých parametrů může být značný a úspěšnost algoritmu by se mohla celkově zvýšit.

Kapitola 10

Závěr

V předcházejících kapitolách jsme představili techniku developmentu založeného na vývoji neuniformního celulárního automatu. Pomocí těchto nástrojů jsme se snažily navrhnout různé typy základních kombinačních logických obvodů a zvláštní třídu nazvanou polymorfní obvody. Předmětem evoluce byly lokální přechodové funkce a počáteční konfigurace celulárního automatu pro danou třídu úloh.

V kategorii klasických kombinačních logických obvodů se podařilo navrhnout např. dvoubitovou sčítačku s přenosem, osmibitovou boolovskou symetrii nebo sedmibitový mediánový obvod. Ve třídě polymorfních obvodů se podařilo vygenerovat např. sedmibitový paritní obvod / sedmibitová boolovská symetrie nebo pětibitový mediánový obvod / pětibitový paritní obvod.

Celková úspěšnost řešení neuniformního celulárního automatu je plně srovnatelná s uniformní verzí. V řadě případů došlo k vygenerování kvalitativně lepších řešení.

Pro budoucí vývoj se doporučuje experimentovat s prvky uvedenými v předchozí kapitole.

Literatura

[1] A. Teller a kol. *PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System.* Technical Report, Carnegie Mellon University, 1995. CMU-CS-95-101.

[2] J. F. Miller a kol. *Cartesian Genetic Programming.* Proceeding of WueoGP2000, LNCS, Vol. 1802, s. 121-132, Springer-Verlag, 2000.

[3] V. Kvasnička a kol. *Evolučné algoritmy.* STU Bratislava, 2000. ISBN 80-227-1377-5.

[4] V. Mařík a kol. *Umělá inteligence 4.* Academia Praha, 2003. ISBN 80-200-1044-0.

[5] P. Bentley. *Evolutionary Design by Computers.* Morgan Kaufmann, 1999. ISBN 1-55860-605-X.

[6] L. J. Fogel. *Artificial Intelligence through Simulated Evolution.* J. Wiley, 1966. ISBN 0-471-33250-X.

[7] M. Gardner. *The fantastic combination of John Conway's new solitair game 'Life'.* Scientific American 223 s. 120-123, 1970.

[8] J. Gruska. *Foundations of Computing.* Int. Thomson Publishing Computer Press, 1997.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975. ISBN 0262581116.

[10] J. R. Koza. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999. ISBN 1558605436.

[11] Bentley P. J. Kumar, S. *On Growth, Form and Computers.* Morgan Kaufmann Publisher, San Francisco, 1999.

[12] Sekanina L. *Design Methods for Polymorphic Digital Circuits.* Department of Computer Systems, Faculty of Information Technology, Brno University of Technology.

[13] Bidlo M. *Generování grafů celulárními automaty.* Ústav počítačových systémů, Fakulta informačních technologii, VUT Brno, 2005.

[14] Bidlo M. *Introducing New Fundamental Classification of Development for Evolutionary Design: Theory and Applications.* Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, 2006.

[15] Hejč M. *Aproximace křivek pomocí genetického programování.* Ústav počítačových systémů, Fakulta informačních technologii, VUT Brno, 2005.

[16] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog, 1973. ISBN 3-7728-0373-3.

[17] H. P. Schwefel. *Numerical Optimization for Computer Models.* J. Wiley, 1981. ISBN 0471099880.

[18] M. Sipper. *Evolution of Parallel Cellular Machines - The Cellular Programming Approach.* Springer, Berlin Heidelberg New Your, 1997.

[19] Zebulum M. Keymeulen D. Stoica, A. *Polymorphics Electronics.* Center for Integrated Space Microsystems, Jet Propulsion Laboratory, California Institute of Technology, Pasadena CA 91109, USA.

[20] Sekanina L. Vašíček, Z. *Evoluční návrh kombinačních obvodů.* Elektrorevue, 2004/43.

[21] S. Wolfram. *A New Kind of Science.* Wolfram Media, 2002. ISBN 978-1579550080.

Dodatek A

Projektová dokumentace

Implementovaný algoritmus byl napsán v jazyce Java. Následuje stručný přehled a popis implementovaných tříd. V závorkách je vždy uveden opdpovídající zdrojový soubor. Podrobnější dokumentaci obsahují zdrojové soubory.

A.1 Popis implementovaných tříd

Circuit (Circuit.java)

Reprezentuje jednotlivá hradla, umožňuje jejich vyhodnocování a identifikaci.

Seznam metod:

- `Circuit ()`: konstruktor.
- `boolean evaluate (BitSet _circuit, boolean _input1, boolean _input2, boolean _line)`: vyhodnocení daného obvodu s definovanými hodnotami vstupů.
- `String toString (BitSet _circuit)`: převod hradla na textovou reprezentaci.

GeneticAlgorithm (GeneticAlgorithm.java)

Třída reprezentující genetický algoritmus.

Seznam metod:

- `GeneticAlgorithm (int _populationSize, byte _mutationChance, byte _mutationStateChance, BitSet[] _inputs, BitSet[] _outputs, byte _outputSize)`: konstruktor.
- `void initialize (short _maxRules, byte _automataSize, byte _nbrhoodSize, byte _maxCycles)`: inicializace počátečního nastavení genetického algoritmu.
- `void makeEvolutionStep (short _tournamentSize)`: provedení jednoho evolučního kroku.
- `void evaluateFitness()`: vyhodnocení fitness hodnoty všech jedinců populace.
- `double getAvgFitness()`: získání průměrné fitness hodnoty populace.
- `void Chromosome getBestChromosome()`: získání nejlepšího jedince aktuální populace.

Chromosome (Chromosome.java)

Reprezentuje chromozom (celulární automat) v genetickém algoritmu.

Seznam metod:

- `Chromosome (short _maxRules, byte _automataSize, byte _nbrhoodSize)`: konstruktor.
- `HashMap generateRules (short _maxRules)`: pseudonáhodné generování pravidel celulárního automatu.
- `BitSet makeRule()`: pseudonáhodné vygenerování jednoho pravidla.
- `BitSet[][] getSchema(byte _cycles)`: získaní kódovaného schématu po provedení definovaného počtu přechodů celulárního automatu.
- `String getTextSchema(byte _cycles, byte _outputSize)`: získání textové reprezentace výsledného obvodu.
- `String getTable(BitSet[] _input, BitSet[] _output, byte _cycles, byte _outputSize)`: získání pravdivostní tabulky v závislosti na zadaném vstupu výsledného obvodu.
- `String getChromosomeTextCode()`: získání textové reprezentace přechodových pravidel celulárního automatu.
- `void drawSchema(Graphics g, byte _cycles, byte _outputSize)`: vykreslení výsledného obvodu do grafického kontextu.
- `BitSet[] evaluate(BitSet[] _input, byte _cycles)`: vyhodnocení fitness chromozomu.
- `int[] getResults(BitSet[] _inputs, BitSet[] _outputs, byte _outputSize, byte _cycles)`: vyhodnocení rozdílů mezi požadovanými a generovanými výstupy obvodu.
- `void setFitness(int _fitness)`: nastavení fitness hodnoty.
- `void setOutputStartBit(byte _startBit)`: určení prvního výstupního bitu obvodu.
- `int getFitness()`: získání fitness hodnoty chromozomu.
- `int Chromosome[] crossing(Chromosome _crossChr)`: křížení dvou chromozomů.
- `int void mutate()`: mutace přechodových pravidel celulárního atuomatu.
- `int void mutateInitialState()`: mutace počátečního stavu celulárního automatu.
- `Object clone()`: kopírování instance třídy Chromosome.
- `BitSet getInitialState()`: získání počátečního stavu celulárního automatu.

INIFile (INIFile.java)

Externí knihovna. Popis viz. zdrojový soubor.

IO (IO.java)

Třída pro načítání dat kombinačního logického obvodu z externího souboru.

Seznam metod:

- `IO (String filename)`: konstruktor.
- `BitSet[] getInputs()`: získání požadovaných vstupů obvodu.
- `BitSet[] getOutputs()`: získání požadovaných výstupů obvodu.

Main (Main.java)

Hlavní třída aplikace.

Seznam metod:

- `void checkArgs(String[] args)`: ověření předaných argumentů.
- `void main(String[] args)`: hlavní smyčka aplikace.

Output (Output.java)

Třída pro ukládání do výstupních souborů.

Seznam metod:

- `Output (String _directory)`: konstruktor.
- `void write(String _text)`: zapsání textu do souboru.
- `void write(BufferedImage image)`: zapsání grafiky do souboru.

Settings (Settings.java)

Třída pro načtení nastavní algoritmu.

Seznam metod:

- `Settings(String iniFile)`: konstruktor.

A.2 Překlad programu

Zdrojové soubory, pro práci s klasickými typy kombinačních obvodů, jsou umístěny v adresáři `src\clasic_circuits`. Zdrojové soubory, pro práci s polymorfními obvody, jsou umístěny v adresáři `src\polymorf_circuits`. Pro překlad programu byly vytvořeny překladové dávky `Makefile` a `make.bat`. Podmínkou je mít nainstalován kopilátor Javy a pro spuštění virtuální stroj.

Překlad: `make`

Spuštění:

klasické typy obvodů: `make run_clasic SETTINGS_FILE IO_FILE`
`run_clasic.bat SETTINGS_FILE IO_FILE`
polymorfní obvody: `make run_polymorf SETTINGS_FILE IO_FILE`
`run_polymorf.bat SETTINGS_FILE IO_FILE`

Smazání přeložených souborů: `make clean`

Dodatek B

Programová dokumentace

Aplikace byla navržena tak, aby splňovala kritérium jednoduchosti a přístupnosti při zachování všech požadovaných a nezbytných parametrů a prvků.

Datové výstupy

Všechny datové výstupy jsou ukládány do uživatelsky specifikovaného adresáře definovaného v souboru s nastavením. Jméno každého souboru je označeno časovou značkou.

Do souboru jsou ukládány následující informace:

- `Generation count` - počet generací evoluce.
- `Best chromosome (circuit) fitness` - fitness hodnota nejlepšího jedince.
- `Best circuit schema` - textova reprezentace výsledného obvodu. Každý řádek obsahuje jeden stupeň obvodu ve tvaru:
  `NÁZEV HRADLA_POZICE {VSTUPY}, ..., NAZEV-HRADLA_POZICE {VSTUPY}` .
- `Best circuit table` - pravdivostní tabulku obvodu ve tvaru:
  `{VSTUP}{POŽADOVANÝ VÝSTUP}{GENEROVANÝ VÝSTUP}`. Vypsány jsou vždy bity v logické hodnotě `1`.
- `Automata size` - velikost celulárního automatu.
- `Neighborhood cell size` - velikost sousedství.
- `Automata initial state` - počáteční stav celulárního automatu. Označeny jsou bity v logické hodnotě `1`.
- `Automata cycles number` - počet stupňů generovaného obovdu.
- `Automata rules` - přechodová pravidla celulárního automatu ve tvaru:
  `{STAV} > GENEROVANÉ HRADLO {VSTUPY} > NS: NÁSLEDUJÍCI STAV BUŇKY`.

Vstupní parametry

Vstupní parametry se předávájí pomocí externího souboru. Příklad je uveden v souboru `settings.ini`. Struktura je následující:

```
[EVOLUTION]
Population_Size = 150   ;velikost populace
Mutation_Chance = 40    ;% pravděpodobnost mutace pravidel
Mutation_State_Chance = 0   ;% pravděpodobnost mutace počátečního stavu
Max_Generation_Count = 10000    ;maximální počet generací evoluce
Test_Count = 1000   ;počet prováděných testů
Tournament_Count = 2    ;počet soutěžících jedinců v turnajové selekci

[AUTOMATA]
Automata_Size = 3   ;velikost celulárního automatu
Output_Size = 1     ;velikost výstupu
Rules_Count = 100   ;maximální počet přepisovacích pravidel jedné buňky
Neighbrhood_Size = 1    ;velikost sousedství
Cycles_Count = 3    ;počet iterací celulárního automatu při generování obvodu

[SYSTEM]
Output_Directory = out  ;adresář pro výstupní data
```

Parametry obvodu

Parametry obvodu se předávají pomocí externího souboru. Příklady lze nalézt v adresáři `inputs`. Struktura dat je následující:

Klasické obvody: `VSTUPY:VYSTUPY`

Polymorfní obvody: `LOGICKÁ_HODNOTA_FUNKCE VSTUPY:VÝSTUPY`