VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

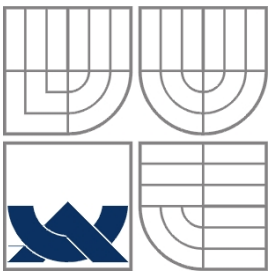
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ RESYNTÉZA KOMBINAČNÍCH OBVODŮ

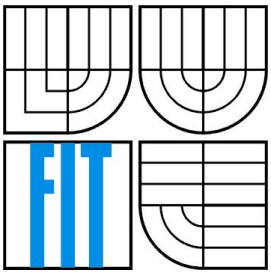
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Ondřej Pták
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ RESYNTÉZA KOMBINAČNÍCH OBVODŮ

EVOLUTIONARY COMBINATIONAL CIRCUIT RESYNTHESIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Ondřej Pták

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. Lukáš Sekanina, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá kombinačními číslicovými obvody a jejich optimalizací. Nejprve jsou představeny hlavní úrovně abstrakce používané při návrhu kombinačních číslicových obvodů. Následně jsou prozkoumány různé metody pro optimalizaci kombinačních číslicových obvodů. Další část této práce je věnována především evolučním algoritmům, jejich společným rysům a variantám: genetickým algoritmům, evolučním strategiím, evolučnímu programování a genetickému programování. Podrobně je popsána varianta genetického programování nazývaná kartézské genetické programování (CGP) a využití CGP v různých oblastech, zejména při syntéze či optimalizaci kombinačních číslicových obvodů. Také jsou představeny některé modifikace CGP a problém škálovatelnosti evolučního návrhu obvodů. V navazující části je popsána metoda pro evoluční resyntézu kombinačních číslicových obvodů. Nejprve je popsán návrh, zejména způsob dělení obvodu na podobvody, poté implementační detaily a nakonec experimenty s touto metodou a jejich výsledky.

Abstract

This project deals with combinational digital circuits and their optimization. First there are presented main levels of abstraction utilized in the design of combinational digital circuits. Afterwards different methods are surveyed for optimization of combinational digital circuits. The next part of this project is mainly devoted to evolutionary algorithms, their common characteristics and branches: genetic algorithms, evolutionary strategies, evolutionary programming and genetic programming. The variant of genetic programming called Cartesian Genetic Programming (CGP) and the use of CGP in various areas, particularly in the synthesis and optimization of combinational logic circuits are described in detail. The project also discusses some modifications of CGP and the scalability problem of evolutionary circuit design. Consequential part of this thesis describes the method for evolution resynthesis of combinational digital circuits. There is description of design, especially the method of splitting circuits into subcircuits, and implementation details. Finally experiments with these method and their results are described.

Klíčová slova

kartézské genetické programování, evoluční resyntéza, kombinační obvody, číslicové obvody, minimalizace, optimalizace

Keywords

cartesian genetic programming, evolutionary resynthesis, combinational circuits, digital circuits, minimization, optimization

Citace

Ondřej Pták: Evoluční resyntéza kombinačních obvodů, diplomová práce, Brno, FIT VUT v Brně, 2013

Evoluční resyntéza kombinačních obvodů

Prohlášení

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

……
Ondřej Pták
20. 5. 2013

Poděkování

Děkuji prof. Ing Sekaninovi, Ph.D. za odborné vedení a přínosné konzultace teoretických i praktických problémů.

Obsah

1 Úvod

V dnešní době je elektronika, obzvláště digitální, všude kolem nás. Nalezneme ji například v mobilních telefonech, videokamerách, autorádiích a v mnoha dalších přístrojích. Číslicová technika se vyskytuje v nových zařízeních, které si bez ní na začátku jednadvacátého století nedokážeme představit, například počítače či vesmírné sondy. Digitalizace však výrazně změnila i předměty dříve známé, jako jsou telefony, měřicí přístroje či dětské hračky. Mnoho oborů je dnes na číslicové technice závislých, nebo ji alespoň s výhodou využívá, například k zrychlení či zlevnění činnosti.

Složitost i počet předmětů využívajících číslicovou techniku neustále roste. Mooreův zákon platí již více než 40 let a říká, že počet tranzistorů v procesorech se zdvojnásobuje přibližně každé dva roky, podobně jako kapacita pevných disků [1]. Podle Českého statistického úřadu vlastnilo v roce 2010 více než 95 % českých domácností mobilní telefon, ale pouhé desetiletí nazpět jen 30 %. S počítači je to podobné, počet domácností vlastnících alespoň jeden v roce 2005 byl 30 %, v roce 2010 již dvakrát větší. Průměr zemí Evropské unie v roce 2005 činil 58 %, o pět let později dokonce 74 %. Obdobný nárůst je i v počtu domácností připojených k Internetu. Pro srovnání bylo roku 2011 přes 60 % českých domácností připojeno, v Holandsku však celých 94 % [2].

Další vývoj digitální techniky se dnes týká především zmenšování komponent a zrychlování jejich činnosti, ale čím dál významněji i snižování spotřeby. K tomu se využívá například paralelní zpracování informace, zmenšování základních prvků a v neposlední řadě optimalizace stávajících obvodů, kterou se zabývá tato práce.

Tato kapitola i následující kapitoly týkající se číslicových obvodů vychází z knihy [3]. Číslicové obvody mohou být kombinační nebo sekvenční. Výstupy kombinačního obvodu závisí jen na aktuálních hodnotách vstupů, zatímco sekvenční obvody mají paměť a výstup tak závisí i na předchozích vstupních hodnotách. Syntéza sekvenčních obvodů je nad rámec této práce, která se věnuje pouze syntéze kombinačních obvodů. Syntéza číslicových obvodů je převod formálního popisu chování obvodu, například pravdivostní tabulky, do logického schématu složeného z logických hradel. Vlastní syntéza obvykle vychází z pravdivostní tabulky nebo z výrazů Booleovy algebry, která popisuje požadované chování.

Minimalizace obvykle navazuje na syntézu, neboť po syntéze obvyklými metodami bývá obvod zbytečně velký. Pokud například obvod vznikl dle výrazu v podobě sumy součinů (sum of products, SOP), roste počet komponent exponenciálně s počtem proměnných. Cílem optimalizace logických obvodů je minimalizovat jeden či více parametrů: počet tranzistorů/hradel/LUT, plochu na křemíkové desce, zpoždění signálu nebo spotřebu energie. Základní metoda minimalizace je úprava výrazů pomocí axiomů a teorémů Booleovy algebry, což je ovšem obtížně algoritmizovatelné. Dobrou pomůckou jsou Karnaughovy mapy, které využívají schopnosti lidského mozku snadno rozpoznat určité objekty, v tomto případě objekty reprezentující booleovskou sousednost. Další metoda, Quine-McCluskey, je také založena na hledání booleovské sousednosti, ale je zde algoritmizovatelný postup, který může snadno vykonávat i počítač. Dalšími metodami jsou různá vylepšení algoritmu Quine-McCluskey či heuristiky, které naleznou dobrá, i když ne vždy optimální řešení. Tyto metody jsou vhodné pro obvody s jedním výstupem. Obvody s více výstupy se dají řešit jako samostatné jedno-výstupové obvody, což je ovšem neefektivní. Některé metody však umí i tyto obvody dobře optimalizovat. Některé optimalizační metody využívají různé pokročilé datové

struktury, například binární rozhodovací diagramy (BDD) nebo and-invert grafy (AIG), a používají sofistikované transformace nad nimi.

Kromě těchto tradičních metod existují i další, například iterační heuristiky nebo minimalizace pomocí kartézského genetického programování (CGP), které se budu především věnovat. Patří do skupiny evolučních algoritmů, které jsou inspirovány biologickou evolucí a jsou obvykle velmi náročné na výpočetní výkon. Pro použití této metody není potřeba mít detailní znalosti řešeného problému. Naprosto postačuje, jsme-li schopni problém zakódovat do chromozomu, ohodnotit a modifikovat kandidátní řešení [16,17].

Tato diplomová práce navazuje na semestrální projekt, jehož cílem bylo zpracování studie o metodách konvenční syntézy číslicových obvodů a možnostech syntézy pomocí CGP. Protože jsou však dnešní obvody velmi komplexní a doba ohodnocení kandidátního řešení v CGP roste exponenciálně s počtem vstupů, je vhodné optimalizovat obvod pomocí CGP po částech. V rámci budoucí DP bude uveden návrh metody rozdělení celého obvodu na menší podobvody a následné nahrazování podobvodů optimalizovanými podobvody.

Diplomová práce je členěna do pěti kapitol. Po této úvodní kapitole následují dvě teoretické kapitoly, kapitola o navržené metodě a závěr. Nejprve jsou v druhé kapitole popsány číslicové obvody. Konkrétně bude uvedeno, na jaké úrovni abstrakce je můžeme navrhovat (2.1), jak mohou být reprezentovány (2.3) a minimalizovány (2.4). Podkapitola 2.3 popisuje detailněji logická hradla, neboť na úrovni hradel bude pracovat navrhovaná metoda. Třetí kapitola se věnuje biologií inspirovaným metodám řešení problémů. Jsou popsány evoluční algoritmy, jednotlivé varianty těchto algoritmů, jejich společné a rozdílné vlastnosti jsou uvedeny v kapitole 3.2. Vlastní metodě CGP se věnuje celá podkapitola 3.3. Dále jsou uvedeny ukázky využití CGP a další užitečné modifikace. Nakonec je pozornost věnována problému škálovatelnosti evolučního návrhu a jak jej lze řešit. Čtvrtá kapitola popisuje navrženou optimalizační metodu. Podkapitola 4.1 popisuje návrh systému, druhá podkapitola implementační detaily a podkapitola 4.3 prezentuje výsledky experimentů provedených s implementovanou metodou. Závěrečné shrnutí obsahuje kapitola 5.

2 Číslicové obvody

Číslicové neboli digitální obvody pracují se dvěma diskrétními hodnotami: logickou 1 a logickou 0. Všechny logické obvody lze rozdělit na kombinační a sekvenční. Tyto dvě kategorie se liší podle výstupní odezvy, která je u kombinačních obvodů závislá jen na vstupních proměnných a u sekvenčních navíc závisí na stavu vnitřní paměti, tedy na předchozích hodnotách vstupů. Paměť obvodu může být vytvořena například z logických hradel se zpětnou vazbou jako tzv. klopný obvod.

Nejdůležitějšími parametry číslicových obvodů jsou: velikost, spotřeba energie a pracovní frekvence. Některé nebo všechny tyto parametry je nutné zohlednit při návrhu číslicového systému.

Stručný přehled problematiky číslicových obvodů prezentovaný v této kapitole byl zpracován podle knihy [3].

2.1 Úroveň abstrakce

Při návrhu, optimalizaci i realizaci kombinačního číslicového systému využijeme různé úrovně abstrakce. Na vyšší úrovni se snáze navrhuje, na nižší můžeme dosáhnout lepších parametrů výsledných obvodů. V následujících odstavcích jsou představeny jednotlivé úrovně od nejnižší po nejvyšší.

Fyzická realizace

S touto nejnižší úrovní obvykle pracují výrobci integrovaných obvodů. Jde například o navrhování rozmístění prvků na čipu, tvar a složení tranzistoru, masky pro jednotlivé vrstvy či vyvedení rozhraní na pouzdro integrovaného obvodu. Díky vývoji na této úrovni jsou základní logické obvody stále menší a rychlejší, případně mají nižší spotřebu či vyšší spolehlivost.

Úroveň přepínačů (tranzistorů)

Na této úrovni je obvod reprezentován grafem propojených přepínačů řízených signálem. Z této podoby lze snadno vytvořit návrh fyzické realizace, neboť přepínače je možné přímo nahradit unipolárními tranzistory. Obvody na této úrovni jsou velké a hůře čitelné a proto se běžně složitější číslicové obvody navrhují na vyšších úrovních abstrakce. Úroveň tranzistorů však nabízí možnosti pro optimalizaci některých menších podobvodů. Například jednoduchý multiplexor lze sestavit podle návrhu na úrovni hradel z minimálně čtrnácti tranzistorů, ale na této úrovni jej lze sestavit pouze z šesti tranzistorů.

Úroveň hradel

Vyšší úroveň abstrakce využívá místo přepínačů hradla, která sestávají z několika přepínačů a řeší jednoduchou logickou funkci, například logický součet. V této práci navrhuji optimalizaci právě na této úrovni, proto se hradlům věnuje detailněji kapitola 2.2.

Popis chování

Předchozí úrovně popisovaly strukturu obvodu. Na úrovni chování (behaviorální úrovni) je obvod popsán požadovaným chováním a struktura obvodu je známa až po syntéze na nižší úroveň abstrakce, obvykle na úroveň hradel.

Požadované chování obvodu bývá často zadáno pravdivostní tabulkou, logickým výrazem či algoritmem. Další možností zápisu chování obvodu poskytují jazyky pro popis hardware, například VHDL, Verilog či ABEL. Popis je podobný algoritmu a je pro návrháře velmi pohodlný, protože se může soustředit na požadovanou funkcionalitu. Návrh na nižší úrovni potom návrhář může použít na optimalizaci některých částí, které po automatické syntéze nevyhovují jeho požadavkům.

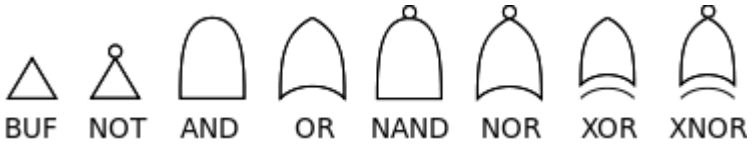
Systémová úroveň

Pokud je cílem vytvořit velmi komplikovaný obvod, například procesor, je i úroveň chování poněkud nedostatečná. Chování celého procesoru je velmi komplexní a bylo by v této podobě málo přehledné. Dobrou možností je dekompozice na jednotlivé subsystémy, které v celkovém pohledu považujeme za abstraktní komponenty s definovanými vstupy a výstupy. Chování těchto komponent (funkčních bloků) je popsáno behaviorálně či strukturálně. Při návrhu je pak snadné a přehledné propojit pomocí sběrnice procesor, paměť a další komponenty, přestože popis funkce většiny z nich je značně obsáhlý. Většina často používaných komponent má svůj popis i optimalizovanou strukturu v nějaké knihovně komponent. Jde například o čítače, převodníky A/D a D/A, registry a mnohé další. Komponenty mohou mít také různou granularitu, neboli některé funkční bloky mohou být jednoduchá hradla, zatímco jiné mohou být složené z tisíců hradel.

2.2 Logická hradla

Logická hradla pracují s dvěma abstraktními hodnotami. Díky tomu můžeme obvody složené z hradel analyzovat či navrhovat pomocí Booleovy algebry nebo různých tabulkových metod. Hradlo má jeden výstup, jeden či více vstupů a provádí (vyčísluje) logickou funkci. Nejjednodušší hradlo je opakovač, který jen zopakuje vstupní hodnotu na výstupní, ovšem se zpožděním (řádově ns). Takovéto hradlo je vhodné zejména pro získání zpoždění na některé větvi, či ke zvýšení logického zisku. Další hradla reprezentují základní logické operace AND, OR, NOT, či o něco komplikovanější XOR, či negace těchto, tedy NAND, NOR, XNOR. Některá hradla, například AND, mohou mít i více vstupů a také lze definovat hradla s dalšími funkcemi, které mohou snížit cenu výsledného obvodu. Příkladem jsou tzv. polymorfní hradla, která přepínají mezi dvěma funkcemi podle hodnoty řídicího signálu (například NAND/NOR, NOR/NAND). Cena výsledného obvodu je nižší než bez použití těchto hradel, ale konvenční metody je dnes ještě neumějí využít [20,24]. V této práci budu využívat následující množinu hradel s jedním až dvěma vstupy, jejichž symboly jsou znázorněny na obrázku 1.

- BUF $Y = A$ opakovač (repeater, buffer)
- NOT $Y = \overline{A}$ negace
- AND $Y = A \cdot B$ logický součin
- OR $Y = A + B$ logický součet
- NAND $Y = \overline{A \cdot B}$
- NOR $Y = \overline{A + B}$
- XOR $Y = \overline{A} \cdot B + A \cdot \overline{B}$ exkluzivní logický součet (nonekvivalence)
- XNOR $Y = \overline{XOR(A, B)} = A \cdot B + \overline{A} \cdot \overline{B}$



Obrázek 1: Jeden ze způsobů značení logických hradel.

Tato hradla však není nutné využívat všechna, neboť každé lze definovat pomocí jiných. Například z hradel AND a NOT, či dokonce jen z hradel NAND, lze sestavit obvod s funkčností jakéhokoli jiného hradla a tedy i jakýkoli kombinační obvod. Této vlastnosti, kdy je možné jednu logickou funkci nahradit pomocí jiných, se říká dualita.

Přestože mají hradla přesně definovanou funkci, mohou se lišit v chování. Hradla se vyrábí v různé technologii, například TTL, CMOS, nebo dříve pomocí diodové logiky. Technologie výroby předurčuje některé vlastnosti hradla, jako jsou provozní napětí, teplota, spotřeba energie, zpoždění mezi změnou na vstupu a odpovídající změnou na výstupu. Velmi důležitou vlastností pro optimalizaci je cena hradla, která může být vyjádřena různě, například počtem tranzistorů, plochou na čipu, spotřebou, či kombinací více parametrů, podle požadované optimalizace.

2.3 Reprezentace obvodů

Číslicový obvod může být definován strukturou nebo chováním. S popisem chování často začínáme, ať již v podobě slovních podmínek či v nějakém HDL jazyku. Chování lze také popsat matematicky, pomocí výrazů Booleovy algebry, pravdivostní tabulkou, nebo pomocí binárních rozhodovacích diagramů (BDD). Naopak struktura obvodu může být popsána grafem na různé úrovni abstrakce: uzly mohou být jednotlivé tranzistory, hradla, ale i celé systémy z tisíců hradel. V následujících podkapitolách jsou jednotlivé reprezentace číslicových obvodů popsány detailněji.

2.3.1 Pravdivostní tabulka

Pravdivostní tabulka definuje obvod jako tzv. „černou krabičku“ („blackbox“), která má jen vstupy a výstupy a nic nevíme o její struktuře. Funkce je v tomto modelu definována tabulkou všech možných kombinací vstupních proměnných a odpovídající odezvou výstupních proměnných. Ve sloupcích jsou proměnné a v řádcích jednotlivé kombinace hodnot. Pro názornost v tabulce 1 je definice dvouvstupového hradla XOR pomocí pravdivostní tabulky.

Tento popis je častý pro zadání vlastností požadovaného obvodu. Obecně má tabulka 2^I řádků, kde I je počet vstupních proměnných. Některé hodnoty výstupů však nemusí být definovány,

protože nás buďto při návrhu nezajímají nebo odpovídají vstupní kombinaci, která nemůže nikdy nastat. Takovýmto stavům se říká „don't care“ (nezajímají nás). Při optimalizaci je pak můžeme považovat za takovou hodnotu, která se více hodí pro zjednodušení obvodu.

A	B	Y=A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Tabulka 1: Pravdivostní tabulka hradla XOR.

2.3.2 Algebraické výrazy

Anglický matematik George Boole vymyslel roku 1854 algebraický systém, který dnes nazýváme Booleova algebra. O 84 let později výzkumník Bellových laboratoří Claude E. Shannon ukázal, že je možné přizpůsobit a využít Booleovu algebru pro analýzu a popis chování logických obvodů, tehdy sestavených z relé. Původní hodnoty nepravda/pravda jsou v této přepínací (switching) algebře reprezentovány stavy 0/1, které mohou být na úrovni fyzické realizace chápany například jako vysoké/nízké napětí pro potřeby návrhu číslicových obvodů.

Booleova algebra je šestice (A, +, ·,',0,1) společně s axiomy uvedenými níže:

- A neprázdná množina proměnných a konstant {0,1}
- + binární operace logický součet (disjunkce, maximum z prvků, OR)
- · binární operace logický součin (konjunkce, minimum z prvků, AND)
- ' unární operace negace (opačný prvek, NOT)
- 0 nejmenší prvek
- 1 největší prvek

Možné úpravy výrazů přepínací algebry definují následující axiomy [30]:

- $X + 0 = X$, $X \cdot 1 = X$ identita
- $X + 1 = 1$, $X \cdot 0 = 0$ nulový prvek
- $X + Y = Y + X$, $X \cdot Y = Y \cdot X$ komutativita
- $(X + Y) + Z = X + (Y + Z)$, $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$ asociativita
- $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$, $(X + Y) \cdot (X + Z) = X + Y \cdot Z$ distributivita

Pro úpravu algebraických výrazů se také používají mnohé teorémy odvozené z těchto axiomů, například tyto:

- $X + X = X$ $X \cdot X = X$ idempotence
- $(X')' = X$ dvojitá negace
- $X + X' = 1$ $X \cdot X' = 0$ komplementarita
- $X + X \cdot Y = X$ $X \cdot (X + Y) = X$ absorpce
- $(X + Y)' = X' \cdot Y'$ $(X \cdot Y)' = X' + Y'$ De Morganovy zákony

Některé z teorémů lze zobecnit na více proměnných, například komutativitu, asociativitu, De Morganovy zákony a další teorémy. Také je možné pomocí těchto teorémů definovat další. Důležitý metateorém o všech předchozích teorémech říká, že teorém zůstane pravdivý, pokud zaměníme 0 za 1 a naopak a také prohodíme binární operátory (+, ·). Úpravami výrazu reprezentující

logickou funkci pomocí výše zmíněných pravidel lze například zjednodušit výraz, vynechat určitý typ operací nebo určit funkční ekvivalenci dvou výrazů.

Pomocí algebraických výrazů jsou obvody obvykle definovány v jednom z následujících formátů, které mají stejnou vyjadřovací sílu, neboli každý výraz v jednom tvaru lze převést na ekvivalentní v ostatních tvarech:

- suma mintermů (kanonická suma, disjunktivní normální forma, DNF),
- seznam mintermů pomocí Σ notace,
- součin maxtermů (kanonický součin, konjunktivní normální forma, CNF),
- seznam maxtermů pomocí Π notace.

Pro popis těchto forem je potřeba definovat některé pojmy (zvýrazněny tučně). **Literál** je proměnná nebo její komplement (negace), například X, X'. **Součinový výraz** (**product term**) je literál, nebo (logický) součin literálů, například X·Y'·Z. **Sumární výraz** (**sum term**) je literál nebo (logická) suma literálů. **Suma součinových výrazů** (**sum-of-products**) a **součin sumárních výrazů** (**product-of-sums**) jsou zřejmé z předchozích. **Normální term** (**normal term**) je součinový či sumární výraz, v němž je každá proměnná obsažena maximálně jednou. **Minterm** *n* proměnných je normální součinový výraz *n* literálů. Například minterm čtyř proměnných může být W·X'·Y·Z'. Obdobně **maxterm** *n* proměnných je normální sumární výraz o *n* literálech.

Existuje blízká korespondence mezi pravdivostní tabulkou a mintermy respektive maxtermy. Minterm odpovídá řádku pravdivostní tabulky, pro nějž je výstup roven jedné, a obdobně maxterm koresponduje s řádkem s nulovým výstupem.

Místo vypsání všech literálů daného výrazu lze použít kompaktnější zápis, kdy pro *n* proměnných je maxterm (respektive minterm) reprezentován *n*-bitovým číslem, jehož hodnota odpovídá číslu řádku v pravdivostní tabulce (pochopitelně číslováno od nuly). Minterm X·Y'·Z odpovídá kombinaci hodnot literálů 101 (1 pro přímou a 0 pro negovanou proměnnou), tedy dekadicky 5. Díky komplementaritě platí pro maxtermy opak: maxterm *n* proměnných označený číslem 5 je X'+Y+Z'. Tímto způsobem můžeme pospat celou pravdivostní tabulku pomocí **kanonických sum** (**canonical sum**), respektive **kanonických součinů** (**canonical product**). Kanonická suma je suma mintermů odpovídajících řádkům pravdivostní tabulky, které produkují na výstupu logickou jedničku a značí se Σ. Obdobně kanonický součin, značený Π, je součin mintermů korespondujících s řádky pravdivostní tabulky, jež generují nulový výstup. Pro názornost následuje funkce dvou proměnných XOR popsaná v obou notacích:

- $XOR = \sum_{A,B}(1,2) = A' \cdot B + A \cdot B'$ (2.1)
- $XOR = \prod_{A,B}(0,3) = (A + B) \cdot (A' + B')$ (2.2)

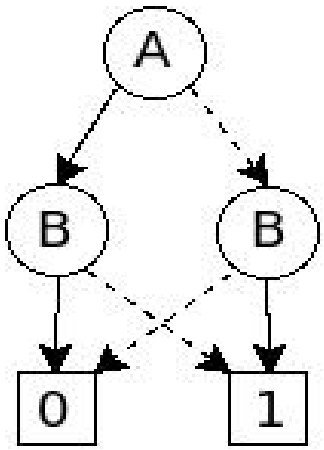
Zkrácené notace se nazývají **seznam mintermů** (Σ), respektive **seznam maxtermů** (Π).

2.3.3 Binární rozhodovací diagramy (BDD)

Binární rozhodovací diagramy jsou acyklické kořenové orientované grafy, jejichž uzly rozhodují podle proměnné o volbě následující hrany při průchodu orientovaným grafem. Nekoncový uzel má vždy dvě výchozí hrany vedoucí k dalším uzlům, jednu pro nulovou (v grafu čárkovaná hrana) a druhou pro jedničkovou hodnotu testované proměnné (plná hrana). Listové uzly reprezentují třídy pro klasifikaci, jeden pro nulové ohodnocení a druhý pro jedničkové ohodnocení. Na obrázku 2 je jednoduchý BDD pro funkci XOR. Při rozhodování o hodnotě výrazu (klasifikování vstupu)

vycházíme z kořene a postupujeme podle hodnot testovaných proměnných až k listům ohodnocených literály 0 a 1. BDD diagramy mohou být redukovány pomocí dvou základních operací: sloučení isomorfních subgrafů a eliminace uzlu, jehož potomci jsou isomorfní [5].

Při práci s číslicovými obvody se obvykle používá varianta BDD nazývaná uspořádaný binární rozhodovací diagram (OBDD). Od předchozího se liší především v tom, že je definované pořadí v uzlech testovaných proměnných při každém průchodu grafem. Tatáž logická funkce může mít s jedním pořadím proměnných lineární velikost BDD vzhledem k počtu vstupů, zatímco při jiném pořadí roste BDD exponenciálně. Pro některé logické funkce dokonce roste velikost BDD vždy exponenciálně. Komplikací je také složitost nalezení nejlepšího pořadí proměnných. Protože se jedná o NP-úplný problém, používají se heuristiky, které naleznou alespoň dobré pořadí, když ne nejlepší. Redukovaný OBDD (ROBDD) je unikátní pro danou funkci a pořadí proměnných, čehož se využívá při zjišťování funkční ekvivalence. Velkou výhodou je efektivní manipulace (například konjunkce či negace) s ROBDD a fakt, že většina důležitých číslicových funkcí je v této reprezentaci poměrně úsporně realizovatelná [5].



Obrázek 2: BDD diagram funkce XOR.

2.3.4 Orientované grafy

Velmi názorný popis struktury obvodu je pomocí grafu. Takový graf musí být orientovaný a acyklický (pro sekvenční obvody však může být cyklický). V grafu jsou definovány vstupy, výstupy, uzly a hrany mezi elementy. Uzlem může být tranzistor, hradlo, nějaký jednoduchý obvod, například jednobitová poloviční sčítačka nebo velké systémy, jako paměti, řadiče apod. V diplomové práci budu využívat pro interní reprezentaci obvodu právě graf složený z logických hradel.

Možným zápisem takového grafu do souboru je tzv. netlist, jenž definuje obvod jako posloupnost instancí (hradel, systémů,...) a spoje mezi nimi. Jednotlivé instance jsou definovány standardní knihovnou (logická hradla) nebo přímo v netlist souboru, například jako subsystém definovaný na jiném místě (hierarchický netlist). Formátů pro tento způsob reprezentace obvodu je mnoho, jedním z nich BLIF (Berkeley Logic Interchange Format) [6], který využívá systém pro syntézu a verifikaci číslicových obvodů ABC [7]. Protože ABC budu využívat v navazující diplomové práci pro počáteční syntézu a některé pomocné operace, jako je mapování na cílovou

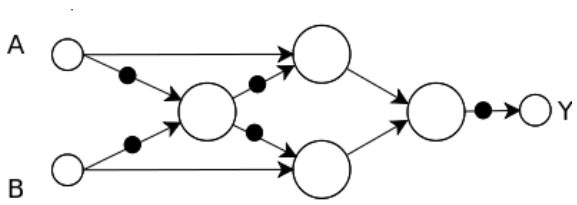
architekturu, zvolil jsem formát BLIF kvůli kompatibilitě. Dalším často používaným formátem je například EDIF (Electronic Design Interchange Format) [8], který je používán již od roku 1987 a od roku 1988 se stal ANSI standardem.

Jiným zajímavým typem grafu je And-invert graf, kterému se z důvodu velké oblíbenosti věnuje blíže následující podkapitola.

2.3.5 And-invert graf (AIG)

AIG graf sestává z tří typů uzlů: primární vstup, primární výstup a AND uzel s dvěma vstupy. Negaci v grafu reprezentuje parametr hrany a znázorňuje se plným kruhem na hraně. Výstavba takového grafu probíhá pomocí tří operací: vytvoření vstupu, vytvoření invertoru a vytvoření AND uzlu. Všechny ostatní operace jsou složeny z těchto základních [9]. Na ukázku obrázek 3 znázorňuje funkci XOR pomocí AIG grafu.

Díky své jednoduchosti je AIG často využíván při syntéze, optimalizaci, či verifikaci číslicových obvodů. Systém ABC [7] využívá AIG jako implicitní vnitřní reprezentaci obvodu. Formát AIGER [10] umožňuje zápis AIG grafu, a to v dvou variantách: v textové a v binární podobě. Textová podoba se hodí především pro aplikace nevyužívající AIGER knihovnu jako výstup. Binární podoba je mnohem kompaktnější a snadněji ji čtou programy, proto se využívá, jde-li nám o výkon. Binární formát má také poněkud přísnější pravidla. Mezi těmito variantami AIGER formátu lze



Obrázek 3: AIG reprezentující funkci XOR.

obvod snadno převádět pomocí programu „aigtoaig“ [10].

2.4 Syntéza a minimalizace

Syntéza číslicových obvodů je realizace logické funkce pomocí konkrétních prvků cílové technologie. Těmito prvky jsou nejčastěji logická hradla, například pro ASIC obvody (application-specific integrated circuit) nebo LUT (lookup table) pro FPGA (field-programmable gate array). LUT umožňuje realizovat složitější funkce než hradla. Tato práce se věnuje pouze syntéze kombinačních obvodů do podoby logických hradel.

Cílem minimalizace výrazů reprezentující číslicové obvody je vytvořit co nejmenší (nejlevnější) obvod, který odpovídá zadané specifikaci. Konkrétně u dvou-úrovňových obvodů (například AND-OR, NAND-NAND, ...) minimalizujeme počet hradel v obou úrovních a v druhé úrovni navíc také počet vstupů. Přímo po syntéze z formátu sumy součinů roste počet možných výrazů (termů), tedy i hradel, exponenciálně vzhledem k počtu vstupů. Minimalizace je tedy více než vhodná. Při návrhu pro PLD (Programmable logic device), FPGA (Field-programmable gate array) a podobné obvody sice máme k dispozici pevný počet prvků na čipu, přesto je minimalizace důležitá a to ze dvou důvodů: může se stát, že se po syntéze náš obvod na čip nevejde a nechceme použít větší a

dražší obvod nebo můžeme využitím menší části čipu snížit spotřebu. Optimalizací se obvykle myslí obecnější proces, při kterém bereme v úvahu nejen cenu (velikost), ale i parametry jako spotřebu, zpoždění a další. V takovém případě je minimalizace součástí komplexnější optimalizace. Můžeme odlišit dva typy optimalizace: dvouúrovňovou a víceúrovňovou. V první variantě minimalizujeme dvouúrovňové obvody, tedy hlavně AND-OR a OR-AND struktury. V druhé variantě uvažujeme složitější obvody o více úrovních.

V následujících kapitolách jsou popsány metody minimalizace. Většinou je minimalizace svázána se syntézou, ale například některé heuristiky vycházejí z již syntetizovaného obvodu na úrovni hradel. Při procesu minimalizace je často třeba brát v úvahu logické hazardy, což je krátkodobý výstup lišící se od správného vlivem různého zpoždění v jednotlivých částech kombinační sítě. Práce s logickými hazardy je však nad rámec tohoto textu, podrobnější informace lze nalézt například v literatuře.

2.4.1 Úprava algebraických výrazů

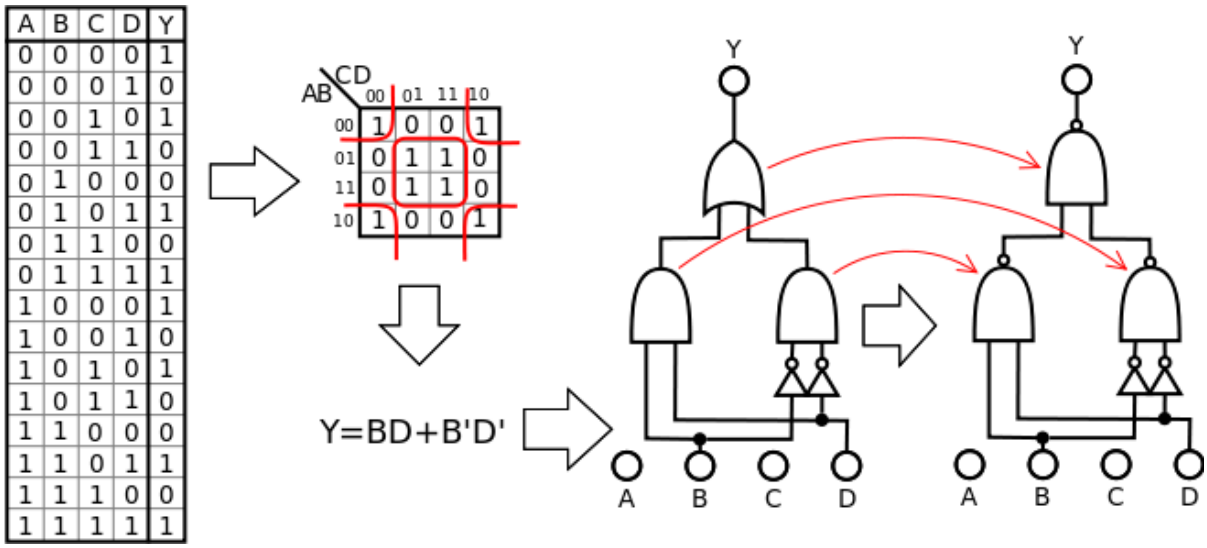
Kanonické sumy (respektive součiny) lze přímo převést na dvouúrovňový graf AND-OR (respektive OR-AND). Takto vzniklý obvod má obvykle daleko k minimálnímu, proto algebraické výrazy minimalizujeme před převodem na hradla. Hradla AND a OR jsou ve většině technologií ovšem dražší, větší a pomalejší než jejich negované ekvivalenty NAND a NOR, proto je vhodné takto vytvořený obvod převést na NAND-NAND, případně NOR-NOR obvod. Toho docílíme vložením dvou invertorů za sebe na všechny vstupy všech hradel druhé úrovně, což nijak nezmění funkci ((X')'=X). Tímto krokem máme v první úrovni NAND (respektive NOR) hradla, případně invertory, pokud druhá úroveň používá přímo globální vstup. Hradla OR (respektive AND) druhé úrovně mají všechny vstupy negované, což můžeme pomocí De Morganových zákonů převést na NAND, respektive NOR $(X' + Y' = (X \cdot Y)')$. Obvod sice bude větší o případné invertory na vstupech druhé úrovně, ale celkově bude levnější. Pomocí dalších zákonů Booleovy algebry je možné obvod postupně zjednodušovat, ale není to jednoduše algoritmizovatelná úloha.

2.4.2 Karnaughovy mapy

Karnaughova mapa je přeuspořádaná pravdivostní tabulka do dvourozměrné podoby, jak znázorňuje obrázek 4. Dovoluje nám identifikovat booleovskou sousednost lépe než pravdivostní tabulka. Hodnota výstupu pro dané ohodnocení proměnných, tedy minterm respektive maxterm, je u pravdivostní tabulky umístěna v řádku identifikovaném hodnotou vstupních proměnných. V Karnaughově mapě umístění hodnoty definují dvě souřadnice, každá definovaná alespoň jednou vstupní proměnnou. Počet prvků mapy je vždy mocninou dvou, stejně jako počet řádků pravdivostní tabulky, tudíž i počet řádků a sloupců. Souřadnice jsou v Grayově kódu místo přímého binárního, díky čemuž se dvě sousední políčka liší v hodnotě právě jedné proměnné. Sousedství buněk je definované ve vertikálním a v horizontálním směru, a to cyklicky, tedy první buňka řádku (respektive sloupce) je sousedem poslední.

Minimalizace pomocí Karnaughových map probíhá tak, že ji nejprve vytvoříme, obvykle z pravdivostní tabulky, poté vyhledáme v mapě oblasti, z nichž sepíšeme součinové výrazy. Jejich suma definuje dvouúrovňový obvod z logických hradel AND, OR a INV (viz obrázek 4).

Oblasti v mapě vytváříme tak, aby jich bylo co nejméně (tedy i součinových výrazů), byly co největší a dohromady obsáhly všechny jedničky a přitom neobsahovaly žádnou nulu. Jednotlivá



Obrázek 4: Ukázka minimalizace logické funkce pomocí Karnaughových map a následná úprava obvodu.

oblast má velikost o mocnině dvou, tvar obdélníku či čtverce a splňuje sousedství definované dříve, což ilustruje obrázek 4. Každá buňka obsahující jedničku musí být alespoň v jedné oblasti. Slučování buněk do většího celku se také nazývá eliminace proměnných, neboť výraz definující větší plochu sestává z menšího počtu literálů. Eliminace proměnných vychází z teorému $X \cdot Y + X \cdot Y' = X$ a je zobecněna i na větší oblasti. Jakákoli oblast o velikosti větší než jedna rekurzivně sestává vždy ze dvou podoblastí poloviční velikosti, které se liší právě v jedné proměnné. Samostatné buňky jsou definovány součinovými výrazy s plným počtem proměnných (mintermy). Pokud již z výrazu nelze odstranit žádná proměnná při zachování definované funkce (nelze zvětšit oblast jedniček), nazývá se takovýto součinový výraz **hlavní implikant** (**prime implicant**).

Sumu součinů z mapy následně vytvoříme sečtením oblastí, které odpovídají součinům, jež jsou tvořeny proměnnými, které jsou v oblasti konstantní. Proměnná je v přímém tvaru, je-li v oblasti rovna jedné a v negovaném pokud je nulová.

Z tohoto formátu lze již přímo sestavit AND-OR graf a v případě potřeby jej převést na NAND-NAND graf, jak popisuje kapitola 2.4.1. Celý tento postup je znázorněn na obrázku 4.

Pomocí Karnaughových map lze minimalizovat i funkce s nedefinovaným výstupem při některých kombinacích vstupů a to tak, že tyto hodnoty v mapě označíme (křížkem či hvězdičkou). Do oblasti je zařadíme, pokud díky tomu získáme méně oblastí, případně oblast zvětšíme. Minimalizujeme-li funkci s více výstupy, existuje modifikace této metody, která umožňuje sdílet společné části výsledného obvodu. Díky komplementaritě je možné postupovat i opačně, tedy vybírat oblasti nul, které nakonec vytvoří součin sum a následně OR-AND graf upravitelný na NOR-NOR. Kvůli exponenciálnímu růstu velikosti mapy vzhledem k počtu vstupních proměnných se Karnaughovy mapy hodí pro minimalizaci funkcí o relativně málo vstupech.

2.4.3 Metoda Quine McCluskey

Tabulková metoda Quine McCluskey vychází ze stejných principů jako Karnaughovy mapy, ale umožňuje minimalizaci funkcí více proměnných. Stejně jako minimalizace pomocí map má tato

metoda shodné dva kroky: nalezení hlavních implikantů a výběr jejich minimální množiny, která pokryje celou funkci (každá kombinace vedoucí k jedničkovému výstupu je obsažena alespoň v jednom implikantu). Algoritmus Quine McCluskey byl vyvinutý původně jako ruční tabulková metoda, ale dnes se využívá hlavně v podobě počítačového programu, který zvládne v rozumném čase minimalizovat funkci o více proměnných. Bohužel prostorová i časová složitost zpracování rostou exponenciálně vzhledem k počtu vstupů, takže i výkonný počítač zvládne v rozumném čase minimalizovat jen funkce o málo vstupech, maximálně okolo tuctu.

2.4.4 Iterativní shoda

Iterativní shoda (iterative consensus) je minimalizační metoda s lepší výpočetní složitostí než metoda Quine McCluskey a její složitost roste vzhledem k počtu součinových výrazů namísto k počtu vstupů. Oproti metodě Quine McCluskey je tato metoda vhodnější k syntéze pro PLD, kde je omezený počet součinových výrazů, ale mnoho dostupných vstupů. Přestože iterativní shoda funguje odlišně, má stejně jako předchozí dvě metody totožné základní kroky: nalezení všech hlavních implikantů a výběr jejich minimální množiny, jež pokryje celou funkci.

2.4.5 Heuristiky

Předchozí metody jsou různě vylepšovány, například pomocí chytře navržených datových struktur, takže obvody zabírají méně místa v paměti, případně počítají rychleji. Kvůli exponenciální složitosti jsou však předchozími, byť vylepšenými metodami prakticky neřešitelné funkce o několika desítkách vstupních proměnných. Pro takové úlohy však existují heuristiky, které pracují relativně rychle a zvládají minimalizovat i složité funkce o mnoha proměnných. Heuristiky nehledají nutně minimální podobu obvodu, ale snaží se o skoro minimální a mají menší časovou či paměťovou složitost, případně oboje.

Velmi úspěšnou heuristikou je například Espresso [15], které minimalizuje vstupní obvod určený dvouúrovňovou funkcí pomocí tzv. krychlí (cubes), definujících součinové výrazy. Další zajímavý program, BOOM [14] (Boolean Minimizer) pochází z ČVUT. Pro minimalizaci funkce s mnoha vstupními proměnnými používá heuristiku BOOM a pro více výstupů heuristiku FC-Min (Find Cover Minimization).

Dalším typem heuristiky je optimalizace syntetizovaného obvodu, případně přímo syntéza pomocí kartézského genetického programování (CGP). Této optimalizační metodě a jejímu využití pro minimalizaci číslicových obvodů se věnuje kapitola 3.

2.4.6 Systém ABC

Existují i komplexnější nástroje nabízející více optimalizačních algoritmů, případně ještě mapování na cílovou architekturu, formální verifikaci či další operace s modely obvodů. Jedním z takových nástrojů je systém ABC vyvíjený na univerzitě v Berkeley [7]. Systém ABC do značné míry vychází ze starších systémů SIS [11], VIS [12] a MVSIS [13], na nichž se univerzita v Berkeley také podílela.

ABC je softwarový systém pro syntézu a verifikaci kombinačních i sekvenčních logických obvodů vyvíjený na univerzitě v Berkeley (Berkeley Verification and Synthesis Research Center). Podporuje různou reprezentaci obvodu, například pravdivostní tabulku, sumu součinů, BDD diagram a AIG graf, s kterým implicitně pracuje. Nad různými modely umožňuje různé funkce pro syntézu, optimalizaci a verifikaci. Vstupem či výstupem pro ABC může být obvod definován v podobě

algebraických výrazů, pravdivostní tabulky, AIG grafu, BLIF formátu [6] a dalších. Systém ABC také umožňuje obvod namapovat na prvky cílové architektury (ASIC, FPGA,...) podle specifikované knihovny elementů. Práce s číslicovými obvody v ABC probíhá pomocí interpretu příkazů, které jsou podrobně popsány v [7]. Tento systém zvládá pracovat s kombinačními obvody o stovkách tisíc a sekvenčními obvody o desítkách tisíc hradel. Doba zpracování je díky různým heuristikám poměrně krátká, řádově vteřiny až minuty [7].

Formát BLIF (Berkeley Logic Interchange Format) reprezentuje číslicový obvod jako netlist [6]. V diplomové práci budu využívat právě tento formát pro ukládání reprezentace obvodu, proto uvádím následující ukázku tohoto formátu ve dvou variantách: obecný graf vycházející z AIG reprezentace (vlevo) a graf reprezentující tentýž obvod po namapování na prvky cílové architektury (vpravo).

```
.model y=a*b+c*d
.inputs a b c d
.outputs y
.names d c n5
11 0
.names b a n6
11 0
.names n6 n5 y
11 0
.end
```

```
.model y=a*b+c*d
.inputs a b c d
.outputs y
.gate NAND A=d B=c O=n5
.gate NAND A=b B=a O=n6
.gate NAND A=n6 B=n5 O=y
.end
```

Syntéza kombinačních obvodů v ABC může probíhat různými způsoby. Nejjednodušší je pouhé namapování na prvky cílové architektury (*map*). Jednoduchou syntézu poskytují například příkazy *rewrite* a *refactor*, společně s vyvažováním AIG (*balance*), lepší výsledky však lze získat iterací *rewrite* a *refacto*r proloženými *balance*. Podle dokumentace ABC jsou poskytovány syntetizační skripty resyn, resyn2 a resyn2rs. Kromě základních operací a nabízených skriptů lze v ABC definovat vlastní skripty pro práci s modely logických obvodů. Následující tabulka ukazuje příklady ABC skriptů pro optimalizaci číslicových obvodů, které jsou porovnány na základě součtu hradel testovaných obvodů. Příkaz *sweep* v ukázkách zjednodušuje logickou síť hradel o hradla bez výstupů, propaguje konstanty a podobně zjednodušuje síť hradel [28].

Skript superchoice obsahuje náslující proceduru [28]:

fraig_store;	*resyn*
fraig_store;	*resyn2*
fraig_store;	*resyn2rs*
fraig_store;	*share*
fraig_store;	*fraig_restore*

Varianta	**Postup**	**Celkem hradel**
1	map; sweep	168279
2	resyn; map; sweep	143308
3	resyn2; map; sweep	136669
4	choice; map; sweep	135245
5	resyn2rs; map; sweep	131637
6	share; map; sweep	128442
7	superchoice; map; sweep	126131
8	20x (superchoice; map; sweep)	113479
9	1000x (superchoice; map; sweep)	106216

Tabulka 2: Porovnání syntetizačních skriptů v ABC na množině obvodů.

Hodonoty převzaty z [28].

Petr Fišer a Jan Schmidt z ČVUT ve svých pracích [28,29] poukázali na nedostatky ABC v efektivitě minimalizace a dosáhli lepších výsledků, pokud iteračně vybírali a optimalizovali podobvod (viz tabulka 2, varianta 9) místo celého obvodu. Průměrně tím vylepšili cenu výsledných obvodů o 9 %, ale v některých případech i o desítky procent. Tento poznatek budu využívat ve své diplomové práci.

3 Přírodou inspirované optimalizační metody

Tato kapitola se stručně zmiňuje o inspiraci přírodou ve výpočetní technice. V podkapitole 3.1 popisuji různé algoritmy prohledávání prostoru řešení. Podkapitola 3.2 vysvětluje principy evolučních algoritmů, jejich varianty a rozdíly mezi nimi. Podkapitola 3.2 popisuje algoritmus CGP a jeho využití. Následně se podkapitola 3.3 věnuje optimalizaci kombinačního logického obvodu pomocí CGP.

Počítání podle přírody (natural computing) zahrnuje nové metody inspirované přírodou a také využití organizmů či netradičních materiálů jako výpočetních platforem. Kromě oblastí jako například kvantové či molekulární počítání sem patří i skupina přístupů nazvaná soft computing, která sestává zejména z neuronových sítí, fuzzy systémů, teorie chaosu a evolučních algoritmů, které budou popsány podrobněji dále. Jednotlivé části soft computingu jsou si podobné v tom, že akceptují nebo dokonce využívají všudypřítomnou nepřesnost reálného světa. Nejde tedy o exaktní výpočty, zato lze takto řešit například úlohy typu: „Čemu je tento vzor podobný?“ nebo „Jak moc má robotická ruka stisknout neznámý předmět, aby nespadl ani se nezdeformoval?“ [17].

Z biologie pochází inspirace například z fylogeneze (vývoj druhů), ontogeneze (vývoj organismu z jediné buňky), epigeneze (vývoj komplexních systémů jako imunitního či nervového systému), dále pak například chování skupin samostatných jedinců či kolonií (například včel nebo mravenců). Z dalších oborů stojí za zmínku fyzika a algoritmus simulovaného žíhání, což je optimalizační metoda vycházející z chování tuhnoucího materiálu, kdy při snižující se teplotě jsou některé strukturální stavy čím dál méně pravděpodobné a systém tak postupně směřuje ke stavu s co nejmenší energií [17,19].

3.1 Algoritmy prohledávání prostoru řešení

Mnoho úloh umělé inteligence lze převést na prohledávání prostoru, který je tvořen možnými řešeními. Dimenze tohoto prostoru odpovídají parametrům dané úlohy, které mohou být diskrétní nebo spojité. V této práci budu dále uvažovat pouze diskrétní problémy. Prostor řešení může být úplný, tedy obsahovat všechna možná řešení, nebo redukovaný, abychom neprohledávali ty možnosti, o kterých víme předem, že jsou nereálné. Takovýto prostor lze prohledat v zásadě dvěma způsoby: systematicky projít všechna potencionální řešení nebo využít heuristiky, které hledají alespoň dostatečně dobrá řešení, aniž by prohledaly všechny možnosti. První metoda vždy nalezne globální optimum, ovšem v některých případech je prostor k prohledání příliš velký, případně ohodnocení kandidátního řešení potřebuje příliš mnoho času na to, abychom prošli celý prostor. Pokud tedy není možné systematické prohledání, použijeme na danou úlohu některou heuristickou metodu. Nejjednodušší taková metoda závisí čistě na náhodě a nazývá se náhodné (slepé) prohledávání. Princip je jednoduchý: jako optimum předdefinujeme nejhorší možnou hodnotu a poté provádíme v cyklu o daném počtu iterací tuto proceduru: vygenerujeme náhodně řešení, to porovnáme s uchovaným nejlepším řešením a pokud je nové řešení lepší, nahradí předchozí nejlepší řešení. Jde o jednoduchou, ale málo efektivní metodu, neboť nemá žádnou strategii prohledávání založenou na dříve prozkoumaných hodnotách. Nicméně při nekonečném počtu kroků vždy nalezne řešení. Pro

prostory s velkým množstvím lokálních extrémů může dosahovat lepších výsledků než mnohé informované metody. Prezence či absence strategie využívající informace z předchozích hodnot rozděluje prohledávací algoritmy na informované a neinformované. Kromě popsaného slepého prohledávání jsou všechny dále uvedené metody hledání informované.

Horolezecký algoritmus je podobný slepému prohledávání. Namísto náhodného zvolení další prohledávané pozice horolezecký algoritmus prohledá v prostoru náhodnou množinu sousedních řešení a nejlepší z nich se stane „středem“ prohledávání v následující iteraci. Tato sousední řešení se obvykle volí náhodně, například negací jednoho bitu stávajícího řešení (tzv. mutací). Výpočetní náročnost značně ovlivňuje velikost okolí, které se prohledává, neboť se vyhodnocuje více potenciálních řešení v každé iteraci. Tato metoda je výborná pro hledání lokálních extrémů, ovšem kvůli náchylnosti k uváznutí v nich často nenalezne globální optimum. Tuto náchylnost lze snížit prohledáváním širšího okolí, ovšem za cenu větší výpočetní náročnosti. Slepé prohledávání a horolezecký algoritmus patří mezi základní prohledávací heuristiky a existuje více jejich modifikaci. Jde například o horolezecký algoritmus s učením, kde je upravována pravděpodobnost mutace nebo metoda zakázaného prohledávání (tabu search). Metoda zakázaného prohledávání vychází z horolezeckého algoritmu a využívá krátkodobou paměť inverzních transformací k těm, jimiž se k současnému bodu dostala. Při prohledávání potom nemůžou být tyto transformace použity při hledání nového okolí. Tato jednoduchá modifikace pomůže při prohledávání opustit některá lokální optima [17,19].

3.2 Evoluční algoritmy

Evoluční algoritmy jsou inspirovány Darwinovou teorií o vzniku druhů pomocí přirozeného výběru. Tato teorie předpokládá nadprodukci, a v konečném prostředí dochází k soupeření o zdroje. Zdatnější jedinci mají větší šanci přežít a mají více potomků. Odtud název funkce „fitness“ v evolučních algoritmech, která určuje, jak dobrý jedinec je vůči konkurenci. Evoluční algoritmy vychází z tzv. neodarwinismu, což je doplněná teorie o poznatky z genetiky a dalších oborů. Významný rozdíl oproti předchozí teorii je připuštění náhodné změny, mutace. Jde tedy o nedeterministický prohledávací algoritmus, který hledá řešení postupným generováním nových řešení pomocí mutace a křížení chromozomu (zakódovaného problému) a výběrem lepších jedinců do dalších generací. Moderní teorie vychází z teorie sobeckého genu [16,17,18].

Jako evoluční algoritmy se označují čtyři metody, které vznikly nezávisle na sobě, ale mají mnohé společné vlastnosti, které zmíním nejprve, a v jednotlivých podkapitolách uvedu hlavně specifické vlastnosti jednotlivých metod.

Evoluční algoritmy jsou asi nejpopulárnější optimalizační technika, použitelná na širokou škálu složitých úloh. Společnými vlastnostmi jsou využití multimnožiny kandidátních řešení (populace), funkce pro jejich ohodnocení (fitness) a operátory pro modifikaci stávajících řešení inspirované biologií. Nejčastějšími operátory jsou křížení a mutace. V prvním případě vzniká nové řešení kombinací dvou, v případě mutace jde o modifikaci jednoho řešení. Evoluční algoritmus začíná iniciací populace, obvykle náhodně či pomocí známého řešení. Následná činnost probíhá iteračně v tzv. generacích. V každé generaci jsou jednotlivá kandidátní řešení z populace ohodnocena fitness funkcí, která udává úspěšnost jedince. Následně jsou vybráni jedinci tvořící rodiče novým jedincům. Potomci jsou generováni pomocí evolučních operátorů, například křížením. Z potomků (a případně i rodičů) jsou nakonec vybráni ti jedinci, kteří postupují do další generace. Pokud je využit elitismus

při výběru, vždy je zachován nejlepší jedinec, v opačném případě může dojít k jeho ztrátě. V nové generaci jsou upřednostněni jedinci s lepší hodnotou fitness, a tak se v průběhu mnoha generací průměrná fitness populace může zlepšovat. Fitness může být různě transformována, například aby hodnota ležela vždy v intervalu [0,1]. Konec evolučního algoritmu bývá definován různě: často počtem generací nebo získáním dostatečně dobrého řešení. Vzhledem k nedeterminismu této metody je evoluční algoritmus obvykle spuštěn vícekrát a je vybráno nejlepší řešení ze všech běhů. Velkou výhodou optimalizace pomocí evolučních algoritmů je, že k řešení problému stačí umět zakódovat problém do chromozomu, ohodnotit řešení a definovat genetické operátory. Není tedy nutné znát detailně všechny aspekty řešeného problému, ale stačí vhodně nastavit podmínky evoluce [17].

3.2.1 Genetický algoritmus

Genetický algoritmus navrhl v sedmdesátých letech minulého století John Holland při zkoumání možnosti adaptace v umělých systémech, avšak populární optimalizační metodou se stal až koncem osmdesátých let díky Davidu Goldbergovi, který prokázal možnost využití v mnoha obtížných úlohách. Jedinec je v genetickém algoritmu nejčastěji zakódován jako řetězec bitů či čísel fixní délky. K produkci potomků se využívá mutace a křížení. Mutace je pro binární chromozom bitová změna, pro číselné zakódování jiná platná hodnota pro daný problém. Zatímco mutace z jednoho rodiče vyprodukuje jednoho potomka, křížení z dvou rodičů obvykle generuje dva potomky (ale může i jen jednoho). Křížení je definováno různě, nejčastěji však jako jednobodové, vícebodové a uniformní. První dvě metody dělí chromozom na dvě či více částí. Tyto části jsou poté kopírovány střídavě od obou rodičů. Uniformní křížení rozhoduje nad každým genem, od kterého rodiče má pocházet. Podle mnohých je křížení klíčovou částí genetického algoritmu, díky níž vůbec funguje. Pravděpodobnosti mutací bývají nízké, například 0,1 %, u křížení vyšší, okolo 70 % všech jedinců rodičovské populace. Výběr nové populace může být deterministický a vybrat ty nejlepší jedince, ovšem častěji je použita některá nedeterministická metoda výběru. Proporcionální selekce definuje pravděpodobnost výběru jedince jako podíl jeho fitness a sumy fitness celé populace. Kromě výběru opírající se o hodnotu fitness může pravděpodobnost výběru vycházet z pořadí v seřazené posloupnosti podle fitness. Velmi oblíbenou metodou je turnaj, kdy je náhodně vybráno s jedinců (s je obvykle 2-8) a z nich postupuje nejlepší [17].

3.2.2 Evoluční strategie

V Německu v šedesátých letech vznikla podobná metoda, dnes známá jako evoluční strategie. V počátcích byla využívána hlavně pro optimalizaci složitých inženýrských úloh v aerodynamice. Typické zakódování pro tuto metodu představuje vektor reálných proměnných. Z operátorů bývá využita obvykle pouze mutace, která k vektoru reálných proměnných přičte náhodně vygenerovaný vektor. Generování odchylky genu probíhá pomocí náhodného generátoru s Gaussovým rozložením s nulovou střední hodnotou a definovaným rozptylem σ. Rozptyl σ může být konstantní, upravovaný podle úspěšnosti potomků nebo může být součástí každého genu a podléhat úpravám při mutaci. Změny σ v průběhu evoluce umožňují samoadaptaci algoritmu [17].

Základní varianta je označovaná jako ES(μ+λ), kdy μ určuje velikost rodičovské populace, a λ počet potomků. Symbol mezi těmito čísly určuje systém výběru nové populace: plus pro výběr z množiny rodičů i potomků a čárka pro výběr pouze z množiny potomků. Výběr další generace je deterministický.

3.2.3 Evoluční programování

Evoluční programování je téměř totožné s evoluční strategií, ale obě metody vznikly nezávisle na sobě. Předchozí metoda pochází z Německa, tuto vymyslel v USA v téže době Lawrence Fogel. Touto metodou vytvářel prediktory implementované pomocí konečných automatů [17].

3.2.4 Genetické programování

Genetické programování vzniklo koncem osmdesátých let a nejvíce se o jeho rozvoj a praktické využití zasloužil John Koza. Ve své knize „Genetic Programming“ [18] popisuje detailně teoretické pozadí i ukázky konkrétní aplikace v jazyku LISP, který je vhodný pro práci se stromovými strukturami. Genetické programování názorně předvádí například na problému symbolické regrese, tvorbě herních strategií či klasifikátorů, návrhu jednoduchých číslicových obvodů a mnohých dalších úlohách [17,18].

Cílem této metody není jen optimalizace hodnot proměnných, ale generování struktur, například matematických výrazů, programů nebo elektronických obvodů. Touto metodou lze vytvářet i patentovatelná řešení, kterých má John Koza momentálně uznaných několik [19].

Podobně jako v genetickém algoritmu jsou zde dvě varianty obnovy populace: generační (následující generace je tvořena jen z potomků) a překrývaní generací (s výběrem z potomků i rodičů). Liší se však zásadně v kódování řešení, které mívá variabilní délku a obsahuje spustitelnou strukturu, nejčastěji stromově reprezentované programy. Kromě mutace a křížení bývají využity i pokročilejší operátory, například pro tvorbu podprogramů. Prvky zakódované struktury mohou být terminály (konstanty, proměnné a funkce bez argumentů) a funkce s argumenty. Funkce by měly být uzavřené, tedy výstup musí být použitelný jako vstup dalších funkcí.

Nejdůležitějším operátorem bývá křížení, kterého je opět více druhů. Základní varianta náhodně vybere v obou rodičích uzel a vymění podstromy, jejichž kořeny tyto uzly tvoří. Mutace probíhá s malou pravděpodobností (pokud vůbec) a pracuje tak, že náhodně vybere uzel, odstraní jej i s podgrafem a nahradí novým vygenerovaným podgrafem.

Fitness často využívá trénovací množinu místo ohodnocení všemi možnými kombinacemi, kterých může být příliš mnoho. Hodnotou funkce fitness pak bývá suma odchylek od požadovaného chování, kterou se evoluční proces snaží minimalizovat [17].

Stromy jsou základní reprezentací programů, nikoli však jedinou. Existuje i varianta genetického programování s lineární či grafovou reprezentací. Lineární reprezentace se podobá zápisu programu v jazyku symbolických instrukcí. Obecné grafy využívá varianta kartézského genetického programování popisovaná v kapitole 3.3 [17].

3.3 Kartézské genetické programování

Kartézské genetické programování se poprvé objevilo v roce 1990, kdy Sushil Louis vytvořil chromozom reprezentující dvourozměrnou mřížku logických hradel [16,17]. Poprvé byla tato metoda uceleně představena v článku Millera a Thomsona roku 1999, kdy byl termín kartézské genetické programování poprvé použit. Této variantě genetického programování se říká kartézské dle dvourozměrné mřížky prvků, nad nimiž je vystavěn pomocí propojení orientovaný graf, obvykle acyklický.

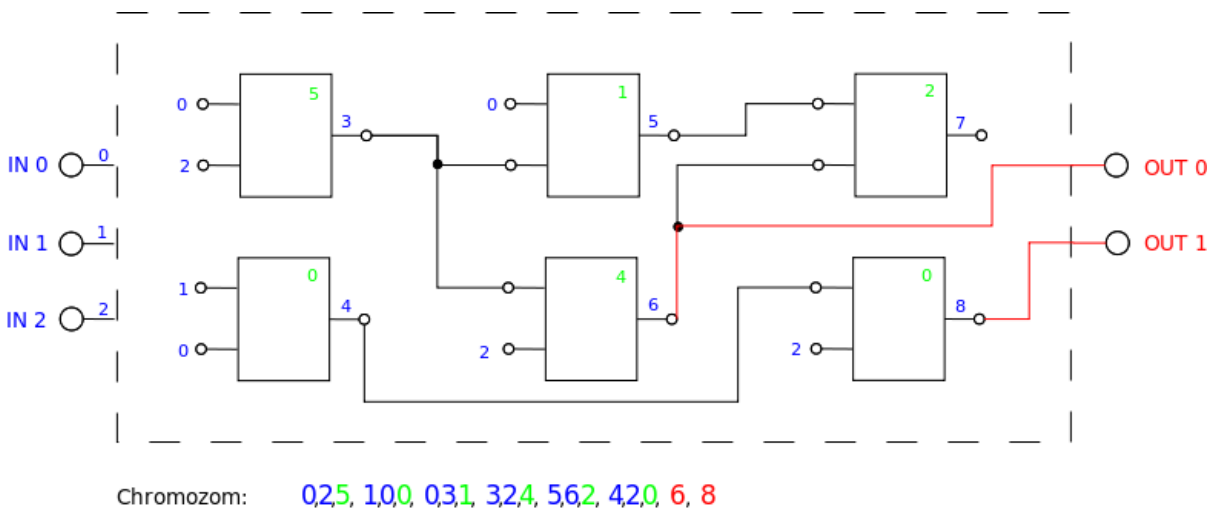
Značná část vlastností kartézského genetického programování je popsána již v předchozí kapitole 3.2 spolu s příbuznými evolučními algoritmy, proto se tato kapitola věnuje hlavně odlišnostem od příbuzných metod.

3.3.1 Zakódování úlohy

Kandidátní řešení úlohy je v CGP reprezentováno acyklickým grafem složeným ze vstupů, bloků a výstupů. Bloky mohou reprezentovat graf na různé úrovni abstrakce, například na úrovni tranzistorů, hradel, matematických operací nebo speciálních obvodů pro konkrétní účel, tzv. funkční bloky, kterými může být například komparátor, multiplexor či sčítačka. Použití reprezentace na vyšší úrovni abstrakce obvykle zmenšuje prostor možných řešení a urychluje tak prohledávání. Na nižší úrovni však může být optimalizace účinnější [16,17].

Každý graf kandidátního řešení musí splňovat některá omezení. Předně se musí vejít do mřížky bloků o definovaném počtu řádků a sloupců (viz obrázek 5). Každý blok může reprezentovat pouze funkci z předem definované množiny funkcí. Parametr L_back omezuje možná propojení grafu. Hodnota L_back udává počet předchozích sloupců, jejichž bloky mohou být napojeny na vstupy hradla. Pokud je L_back roven jedné, smí být propojeny jen sousední sloupce. Primární vstupy mohou být připojeny na vstup kteréhokoli bloku. Globální výstup může být napojen na libovolný blok, případně na globální vstup [16,17].

Kandidátní řešení (jedinec) je pro potřeby evoluce reprezentován chromozomem, který má obvykle podobu pole celočíselných hodnot. V chromozomu je mřížka bloků uložena lineárně, podle indexů



Obrázek 5: Ukázka kandidátního řešení v CGP a odpovídajícího chromozomu.

naznačených v obrázku 5 modře. Indexování bloků začíná primárními vstupy a pokračuje bloky po sloupcích. Každý blok je v chromozomu zastoupen indexy pro vstupy bloku (obvykle dvěma) a indexem funkce (zeleně). Na konci chromozomu jsou indexy bloků (nebo primárních vstupů), které jsou napojeny na globální výstupy (v obrázku červeně). Chromozom má fixní délku, ovšem výsledný obvod (fenotyp) má variabilní délku, neboť nemusí být v grafu všechny bloky zapojeny, jako například bloky s indexy 5 a 7 v obrázku 5 [16,17].

3.3.2 Ohodnocení kandidátních řešení

Předpokládejme, že je cílem vytvořit podle zadané specifikace kombinační obvod. Základní způsob ohodnocení kandidátního řešení je testování všech možných vstupních kombinací a porovnávání výstupů kandidátního řešení s požadovanými výstupy. Fitness je definována jako počet těchto kombinací a tedy i doba ohodnocení ovšem roste exponenciálně s počtem vstupů. Výpočet lze urychlit pomocí paralelizace na úrovni bitů. Moderní procesory používají pro data obvykle 64-bitová slova a s nimi mohou provádět logické operace, takže můžeme otestovat až 64 jedno-bitových logických operací najednou, v jediné instrukci. Přestože se jedná o 64-násobné urychlení, můžeme kvůli exponenciální složitosti použít pouze o několik vstupů více. Další možností urychlení nabízí paralelní architektury jako jsou FPGA čipy nebo procesory moderních grafických karet (GPU) [16] .

Lze ovšem evolučně optimalizovat i kombinační obvody s mnoha vstupy pomocí začlenění tzv SAT solveru. V tomto případě jsou referenční i porovnávaný obvod převedeny do podoby součinu klauzulí a jejich splnitelnost je rozhodnuta pomocí SAT solveru. Lze ještě více snížit složitost porovnáváním pouze změněné části obvodu [22,23].

Často jsou použity dvě verze výpočtu fitness jedince. Pokud obvod nepracuje správně, vychází fitness například z počtu správných výstupních bitů. Po nalezení správného řešení bere výpočet fitness v úvahu i počet hradel, respektive sumu jejich cen [16].

Další možnost, jak urychlit ohodnocování jedinců, spočívá ve zmenšení počtu testovaných hodnot. Místo všech kombinací testujeme jen jejich podmnožinu, tzv. trénovací množinu. Tato metoda sice urychlí výpočet, ale výsledný obvod nemusí správně reagovat na kombinace, na které nebyl trénován. Takovýto obvod tedy špatně zobecňuje na ostatní kombinace, proto je tato metoda často nepřípustná [16].

Kromě počtu hradel či ceny nás mohou při ohodnocování kandidátního řešení zajímat i další vlastnosti, jako je například zpoždění či příkon. Tyto vlastnosti však často působí proti sobě. V tomto případě mluvíme o multikriteriální optimalizaci a existují dvě možnosti. První možností je váhovaný součet dílčích funkcí, odpovídající požadovaným parametrům, tedy každá dílčí funkce má svůj koeficient, určující jakou měrou se ve fitness projeví. Problém může být s vhodným nastavením koeficientů. Jde o snadnou a rychlou metodu, ale ne vždy podává dobré výsledky. Druhou, lepší, ale výpočetně náročnější variantou je hledat paretovsky optimální kandidáty. Těch může být více a jsou to takoví, že neexistuje jiné řešení, které by bylo v některém kritériu lepší a zároveň nebylo v žádném horší. Tento koncept je implementován například v algoritmu NSGA-II [17].

3.3.3 Výběr jedince

Populace v CGP bývá malá, často obsahuje jen jediný chromozom. Potomků bývá více, obvykle čtyři. Vybrán je z celé populace ten nejlepší jedinec, který se dále mutuje a vytváří potomky pro další generaci. Tento elitismus má ovšem podstatnou změnu: pokud je více jedinců se shodnou hodnotou fitness, pak se vybere jako další nejlepší jedinec ten, který se liší od předchozího. Tato modifikace zajišťuje lepší diverzitu populace [4,16,17].

3.3.4 Genetické operátory

Obvykle se v CGP používá pouze mutace. Během jedné operace mutace může být změněn jeden i více genů. Pokud je na jeden chromozom více mutací, bývá počet předem určen nebo je odvozen z délky chromozomu. Bodová mutace ovšem musí zachovat graf konzistentní, tedy legální funkci a propojení podle L_back parametru zmíněného výše [16].

Přestože křížení u CGP není obvyklé, protože příliš rozbíjí kandidátní jedince, v některých situacích se ukázalo jako vhodné. Jde například o práci Clegg a kolektivu. Použili větší populaci a turnajovou selekci. Slibné výsledky získali například při regresi nebo zpracování obrazu. Při evoluci číslicového obvodu úspěšně použili křížení pří více chromozomech na jedince [16].

3.3.5 Využití CGP a jeho modifikace

Tato podkapitola stručně popisuje využití CGP v různých oblastech a jeho zajímavé modifikace.

Kartézské genetické programování bylo původně použito pro syntézu a optimalizaci číslicových obvodů, které dokáže zmenšit oproti obvodu optimalizovaném pomocí ABC v průměru o deset až dvacet procent [21,22]. Využitím nekonvenčních hradel (například polymorfních) je možné evolučně vytvořit ještě menší obvody na úrovni tranzistorů [20,24]. Tato hradla, provádějící jednu ze dvou funkcí na základě hodnoty řídicího signálu, však obvykle konvenční metody nejsou schopny využít. Z elektronických obvodů jsou zajímavé ještě například násobičky mnoha koeficienty. Jde o obvody s jedním vstupem a mnoha výstupy, každé reprezentující vstup vynásobený nějakým koeficientem. Jako bloky jsou použity posuny čísel (shift, <<), součet, rozdíl. Takovéto obvody se v praxi dají využít například při návrhu FIR filtrů (finite-impulse-response). Další elektronické obvody optimalizovatelné pomocí CGP jsou aplikačně specifické cache paměti. Velmi podobnou úlohou je vytváření matematických výrazů aproximující neznámou funkci nebo zpracování obrazu. Při zpracování obrazu lze CGP využít například pro evoluci filtrů (vstupy tvoří sousední pixely a výstupy hodnoty vypočítaného nového pixelu) nebo pro detekci hran. Pokud vstupy reprezentují souřadnice pixelu, bloky jsou tvořeny matematickými operacemi a výstupy definují hodnotu pixelu, může evoluce produkovat líbivé abstraktní obrazce. Podobně je možné automaticky tvořit i hudbu. Velmi zajímavou oblastí, v níž CGP pomáhá, je i medicína. CGP bylo testováno pro diagnostiku chorob jako je rakovina, Alzheimerova nebo Parkinsonova choroba. CGP je s úspěchem využito v mnoha dalších pracích a oblastech, namátkou řízení helikoptéry či robotů, rozpoznávání objektů nebo některých úlohách z bioinformatiky [16].

Z předchozího odstavce se může CGP jevit jako optimalizační metoda vhodná na všechny problémy, ovšem není tomu tak. Například evoluce FIR a IIR filtrů na úrovni hradel nedosahuje požadovaných výsledků [26]. Problém je také v reprezentaci obvodu pomocí chromozomu, neboť některá kandidátní řešení jsou častá, zatímco jiná poměrně vzácná, a bohužel často potřebujeme navrhovat právě ty méně často se vyskytující [27].

Z kartézského genetického programování se vyvinulo mnoho variant, z nichž několik zmíním. Pro některé úlohy je vhodné použít více chromozomů a každý odpovídá jiné části úlohy, například nějaký subsystému nebo výstupnímu signálu. Tato modifikace popisovaná zkratkou MC-CGP (multi-chromosome CGP) se přibližuje evoluci živých systémů, které také používají více chromozomů. Embedded CGP umožňuje evoluci vytvořit a znovu použít moduly (podobné funkcím v programování) a modulární CGP dokonce umožňuje i použití modulů v modulech, jak je běžné u programovacích jazyků. Mezi další odnože CGP patří vývojové sítě (CGPDN), umožnující

navrhovat strukturu a parametry umělé neuronové sítě. Sebemodifikující se CGP (SMCGP) používá kromě běžných funkcí bloku i tzv. sebemodifikující funkce, které reprezentují nějakou změnu genotypu (chromozomu). Tyto změny se v průběhu dekódování ukládají do seznamu změn, které se provedou mezi iteracemi CGP. Zajímavé je také použití relativního indexování propojení CGP a variabilní počet vstupů a výstupů v průběhu evoluce. SMCGP může nalézt obecné řešení problému, které pak zobecňuje [16].

3.3.6 Škálovatelnost

Velká nevýhoda CGP oproti konvenčním metodám je škálovatelnost. Pomocí CGP lze evolučně navrhovat jen relativně malé obvody, přibližně do dvanácti vstupů a několika desítek hradel. Text [25] popisuje a porovnává limity při evoluci obvodů pomocí různých metod. Tento problém lze částečně řešit různými metodami. Tyto metody můžeme rozdělit do tří skupin: evoluce na úrovni funkčních bloků, inkrementální evoluce a vývin (netriviální mapování mezi genotypem a fenotypem) [17].

Pokud použijeme k evoluci místo hradel funkční bloky, reprezentující složitější prvky, můžeme zmenšit prostor řešení, který prohledáváme. Takovéto bloky mohou být například násobičky, posuvy a podobně. Při této metodě se k ohodnocení často používá trénovací množina vstupních vektorů [17].

Inkrementální evoluce řeší problém škálovatelnosti pomocí dekompozice. Obecně má průběh tři fáze: dekompozice na jednodušší podproblémy, evoluce dílčích úloh a spojení výsledků do celku. Klasicky probíhá dekompozice z pohledu vstupů, případně výstupů. V diplomové práci se pokusím také o inkrementální evoluci, ovšem trochu jiným způsobem a nikoli pro syntézu ale jen pro optimalizaci [17].

Vývin (development) umožňuje v relativně malém genotypu kódovat velké fenotypy. Další výhodou je efektivní práce s opakujícími se strukturami, nevýhodou náročnost návrhu zakódování [17].

4 Navržená optimalizační metoda

Tato kapitola popisuje navrhovanou metodu pro optimalizaci kombinačních logických obvodů pomocí kartézského genetického programování. Oproti klasickému CGP však nebude obvod optimalizován vcelku, ale po částech o definované velikosti.

Velkým omezením CGP je exponenciální složitost ohodnocení kandidátního řešení vzhledem k počtu vstupů, neboť je zapotřebí otestovat všechny možné vstupní kombinace. Kvůli této vlastnosti je možné pomocí CGP klasicky optimalizovat pouze obvody o relativně malém počtu vstupů. Tento problém je možné řešit například ohodnocováním pomocí SAT solveru, který pracuje s CNF reprezentací obvodu [21, 22, 23]. V této metodě bude problém časové složitosti řešen pomocí snižování počtu vstupů, čehož bude dosaženo optimalizací obvodu po částech. Metoda nebude používat SAT solver, ale klasické otestování všech vstupních kombinací, kterých však bude výrazně méně než u celého obvodu.

Navrhovaná metoda by měla snižovat cenu (plochu na čipu) kombinačních podobvodů, které byly syntetizovány a optimalizovány pomocí deterministického systému ABC z univerzity v Berkeley. Evoluce tedy nebude obvod přímo vytvářet, ale pouze optimalizovat již syntetizovaný. Protože se jedná o novou metodu, bude v první fázi pro zjednodušení implementace zanedbáno zpoždění podobvodu i logický zisk jednotlivých hradel.

Podkapitola 4.1 popisuje nejdříve návrh systému na vysoké úrovni abstrakce a následně detailněji jednotlivé části. Podkapitola 4.2 popisuje podstatné implementační detaily, například rozhraní klíčových částí či popis optimalizačního skriptu pro ABC. Nakonec budou v podkapitole 4.3 prezentovány výsledky experimentů s navrženou metodou.

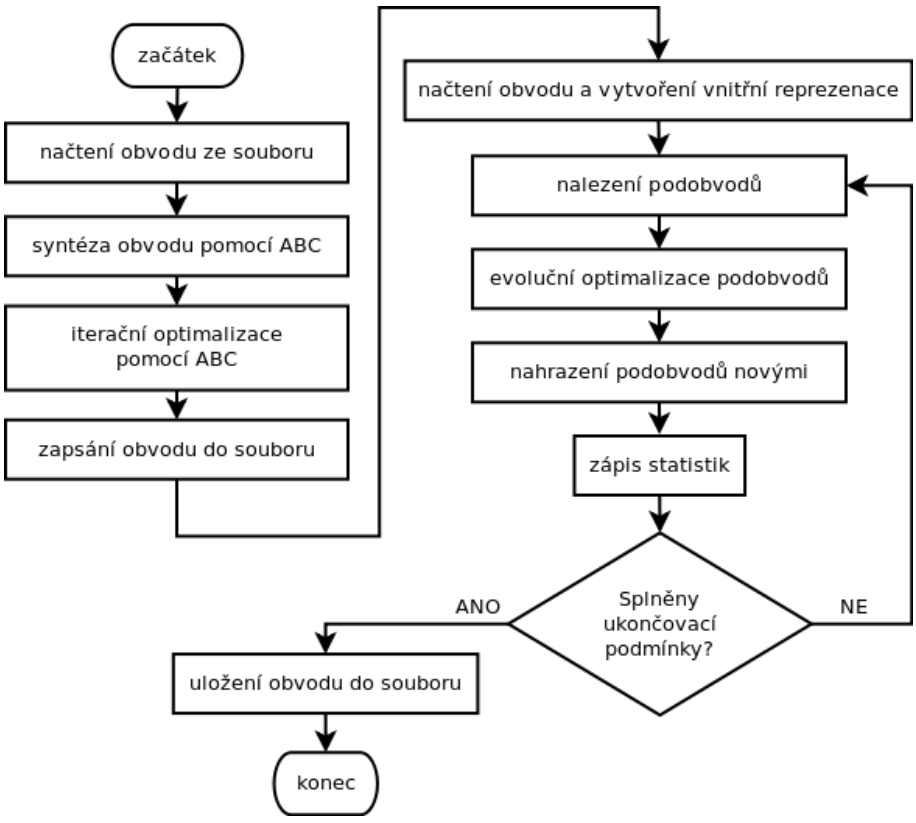
4.1 Návrh

Tato kapitola prezentuje návrh optimalizační metody. Nejprve je představen celkový pohled na navrhovaná optimalizační systém a jeho členění a v následujících podkapitolách jsou detailněji prezentovány klíčové součásti.

4.1.1 Optimalizační systém

Činnost celého systému znázorňuje diagram v obrázku 6. V levé části diagramu se nachází syntéza a optimalizace obvodu pomocí ABC a vpravo iterační optimalizace po částech pomocí CGP. Obě tyto části budou implementovány jako samostatné programy.

Vstupem metody bude definice kombinačního obvodu ve formátu pla nebo blif. Takto definovaný obvod bude nejprve syntetizován a optimalizován pomocí ABC a uložen v blif formátu. Takto vzniklý soubor bude následně optimalizován po částech pomocí CGP a uložen opět jako blif soubor.



Obrázek 6: Činnost navrhované metody.

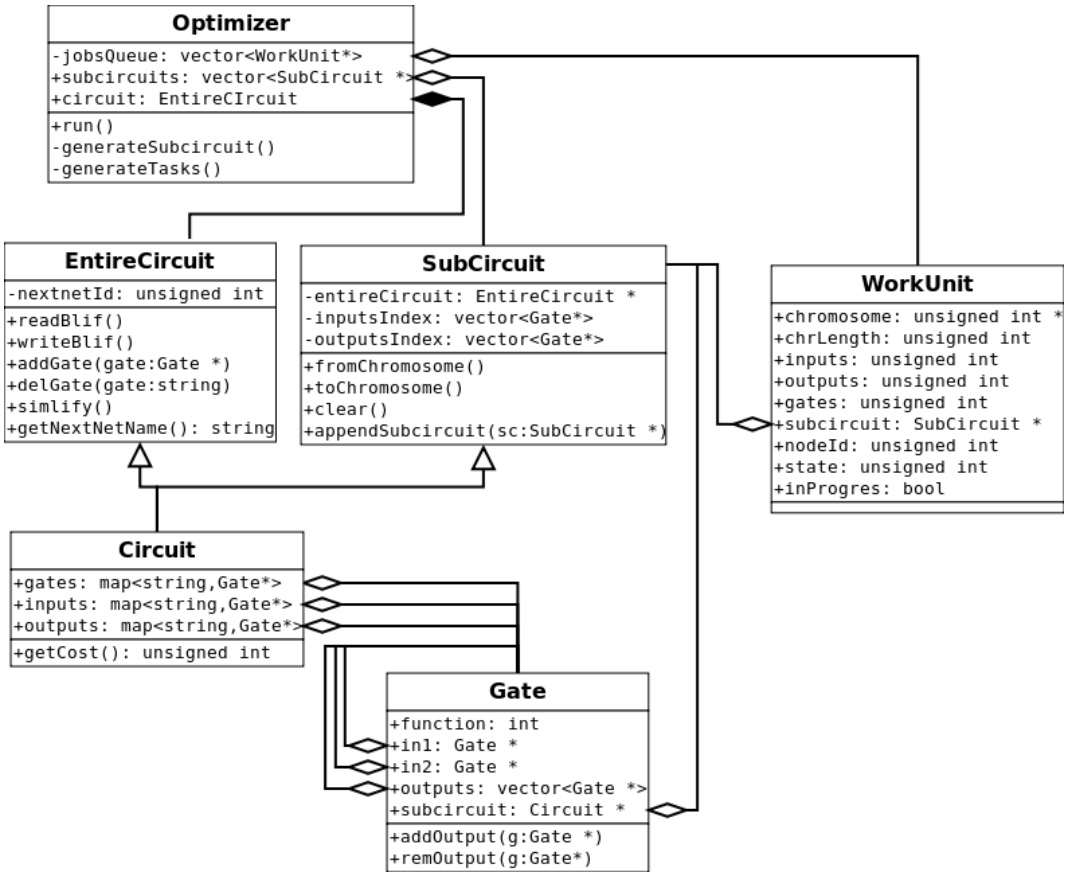
4.1.2 Syntéza a optimalizace pomocí ABC

Navrhovaná metoda bude obvod pomocí CGP pouze optimalizovat, případná syntéza z pravdivostní tabulky (pla) bude provedena spolu s počáteční optimalizací pomocí deterministického systému ABC. Takto optimalizovaný obvod bude následně použit jako vstupní obvod pro evoluční optimalizaci.
Optimalizace pomocí ABC i pomocí CGP bude pracovat s následující množinou hradel s maximálně dvěma vstupy: {ZERO, ONE, INV, BUF, AND, OR, NAND, NOR, XOR, XNOR}, kde ZERO a ONE jsou konstanty a BUF reprezentuje hradlo s funkcí Y=A a s nulovou cenou.

4.1.3 Vnitřní reprezentace obvodu a podobvodů

Vstupem programu pro evoluční optimalizaci bude model obvodu ve formátu blif namapovaný na danou množinu hradel. Při načítání tohoto souboru bude program vytvářet vlastní model obvodu. Tento model musí umožnit definici podobvodů v rámci celého obvodu a obousměrné procházení acyklickým grafem reprezentující obvod či podobvod. Také bude nutné podobvod nahrazovat novým podobvodem, který vyprodukuje evoluční algoritmus.

Diagram tříd na obrázku 7 zobrazuje navrženou strukturu pro uchování modelu obvodu a práci s ním. Hlavní třída (Optimizer) obsahuje jednen celý obvod (EntireCircuit) a seznam podobvodů (SubCircuit). Třídy EntireCircuit a SubCircuit jsou odvozeny z třídy Circuit, která definuje hradla,



Obrázek 7: Zjednodušený diagram tříd potřebných pro model obvodu a manipulaci s ním.

vstupy a výstupy (pod)obvodu. Hradlo (třída Gate) obsahuje 2 ukazatele na další hradla jako vstupy a seznam ukazatelů na hradla napojená na výstup tohoto hradla.

Rozhraní celého obvodu i jednotlivých podobvodů bude tvořeno pomocí abstraktních vstupně-výstupních hradel, které mají stejnou funkci jako hradlo BUF, tedy na výstup propagují první vstup. Tyto hradla budu dále označovat jako I/O hradla.

Hradla jsou vzájemně provázaná, lze tedy z každého hradla přistupovat k hradlům připojeným na jeho vstupy či výstup. Struktura také umožňuje přístup z hradla k podobvodu, v kterém se nachází.

4.1.4 Podobvody

Podobvod je definován množinou vzájemně propojených hradel, vstupy a výstupy. Pro potřeby kartézského genetického programování je nejdůležitější vlastností počet vstupů kvůli exponenciální složitosti ohodnocení správného chování. Velmi podstatný je i počet hradel v podobvodu. S rostoucím počtem hradel trvá ohodnocení déle, ale také je více možností jak podobvod upravit. Počet výstupů ovlivňuje činnost CGP méně než předchozí dvě vlastnosti.

Pro snížení počtu vstupů budou vybírány jednotlivé podobvody s definovanými minimálními a maximálními rozměry. Každé hradlo obvodu bude maximálně v jednom podobvodu, proto bude možné takto vytvořené podobvody optimalizovat samostatně.

Třída podobvodu (obrázek 7) obsahuje metodu pro vytvoření chromozomu z podobvodu (toChromosome()) a metodu pro nahrazení podobvodu novým podobvodem z chromozomu (fromChromosome()). Při vytváření chromozomu jsou vytvořeny indexy vstupů a výstupů, které jsou později potřebné k dekódování optimalizovaného chromozomu.

Dále v textu je uvedený polo-formální popis navrženého algoritmu 1 pro dělení obvodu na podobvody. Vstupem tohoto algoritmu je celý obvod a konstanty určující povolené rozměry podobvodů: minimální/maximální počet vstupů/hradel. Výstupem je upravený obvod (doplněný o I/O hradla) a množina podobvodů. Vstupy, respektive výstupy, podobvodu jsou tvořeny hradly, která jsou napojena na vstupy, respektive výstup, hradel obsažených v daném podobvodu, ale nenachází se v něm. Za sousední podobvod je považován takový, který má alespoň jedno hradlo mezi vstupy či výstupy tohoto podobvodu.

Při vytváření podobvodu jsou nejprve vytvořeny jedno-hradlové podobvody pro každé hradlo z celého obvodu (C). Z těchto počátečních podobvodů jsou poté náhodně vybírány podobvody k expanzi (s). Dokud to je možné, snaží se algoritmus k podobvodu připojit některý ze sousedních podobvodů. Pokud po vyčerpání možností expanze má obvod příliš málo vstupů nebo hradel, je rozdělen zpět na jedno-blokové podobvody, které mohou být později navázány na jiný rozšiřující se podobvod. Při rozšiřování podobvodu je nejprve vytvořena množina možných rozšíření (O). Tato množina je dále redukována: nejprve jsou vyřazeny příliš velké podobvody, poté podobvody s větším počtem vstupů než je minimální počet v celé množině, a nakonec podobvody, které mají více výstupů než je minimum v množině. Touto redukcí algoritmus optimalizuje rozměry podobvodu nejprve podle počtu vstupů a poté podle počtu výstupů. Pokud je množina možností neprázdná, vybere se libovolná zbylá možnost expanze, ta se provede a algoritmus pokračuje prozkoumáním okolí podobvodu pro další rozšiřování. Pokud je množina možností prázdná, pokračuje algoritmus vybráním dalšího jedno-blokového podobvodu k expanzi. V případě, že dále nerozšiřitelný podobvod nemá dostatečné rozměry, je před vybráním dalšího počátku rozdělen zpět na jedno-hradlové podobvody. Po vyprázdnění množiny možných počátečních podobvodů (X) je třeba ještě vložit I/O

hradla mezi podobvody, které budou plnit funkci pevného rozhraní, které evoluce nemůže změnit. Tímto je možné optimalizovat sousední části obvodu nezávisle.

Algoritmus 1: Dělení obvodu na podobvody.

Vstup: obvod (množina hradel) C,
konstanty:

- MIN_IN (minimální počet vstupů podobvodu),
- MAX_IN (maximální počet vstupů podobvodu),
- MIN_G (minimální počet hradel podobvodu),
- MAX_G (maximální počet hradel podobvodu)

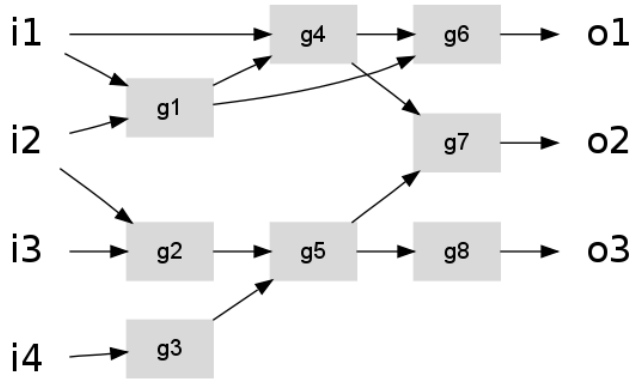
Výstup: obvod C, množina podobvodů (množin hradel) S

Postup:

1. Dále budou použity následující funkce:
   1. inputs(M) vytvoří množinu hradel připojených na vstupy podobvodu M
   2. outputs(M) vytvoří množinu hradel připojených na výstupy podobvodu M
   3. neighbours(M) vytvoří množinu sousedních podobvodů k podobvodu M
2. Nechť $S := \emptyset$ je množina podobvodů a $X := \emptyset$ množina jedno-hradlových podobvodů, z které budou náhodně vybírány počáteční podobvody k expanzi.
3. $\forall g \in C : S := S \cup \{\{g\}\}, X := X \cup \{\{g\}\}$ (jedno-hradlové podobvody)
4. $If\ |X| = \emptyset \Rightarrow goto\ 18.$
5. Nechť *s* je náhodně vybraný podobvod z X; $X := X \setminus s$
6. $N := neighbours(s)$
7. $O := \emptyset ; \forall n \in N : O := O \cup \left[(n, \{s \cup n\})\right]$
8. $\forall (n, no) \in O : (inputs(no) > \text{MAX\_IN} \vee |no| > \text{MAX\_G}) \Rightarrow O := O \setminus \{(n, no)\}$
9. $\text{min\_in} := \infty ; \forall (n, no) \in O : (inputs(no) < \text{min\_in}) \Rightarrow \text{min\_in} := inputs(no)$
10. $\forall (n, no) \in O : (inputs(no) > \text{min\_in}) \Rightarrow O := O \setminus \{(n, no)\}$
11. $\text{min\_out} := \infty ;$
    $\forall (n, no) \in O : (outputs(no) < \text{min\_out}) \Rightarrow \text{min\_out} := outputs(no)$
12. $\forall (n, no) \in O : (outputs(no) > \text{min\_out}) \Rightarrow O := O \setminus \{(n, no)\}$
13. $(O = \emptyset \wedge (|s| < \text{MIN\_G} \vee inputs(s) < \text{MIN\_IN})) \Rightarrow$
    $\Rightarrow (\forall g \in s : S := S \cup \{g\} ; S := S \setminus s ; goto\ 4.)$
14. $If\ O = \emptyset \Rightarrow goto\ 4.$
15. $Nechť\ (n, ns)\ je\ libovolný\ prvek\ z\ O$
16. $S := (S \cup ns) \setminus \{n, s\} ; X := X \setminus n$
17. goto 6.
18. Vložení I/O hradel mezi podobvody v obvodu C.

Příklad použití algoritmu

Pro větší názornost uvádím příklad možného průběhu algoritmu 1. Obvod (C) k rozdělení na podobvody je znázorněn na obrázku 8 a povolené rozměry podobvodů jsou definovány takto: MIN_IN = 1, MAX_IN = 3, MIN_G = 3, MAX_G = 5. V dále uvedeném postupu je uveden odpovídající bod algoritmu 1, změněné proměnné či množiny a případný komentář. Zřejmé či opakující se kroky postupu jsou vynechány.



Obrázek 8: Vstupní obvod pro ukázku agoritmu 1.

- (3) $S=X=\{\{g1\},\{g2\},\{g3\},\{g4\},\{g5\},\{g6\},\{g7\},\{g8\}\}$ Rozdělení obvodu C na jedno-hradlové podobvody.
- (5) $s=\{g5\}, X=\{\{g1\},\{g2\},\{g3\},\{g4\},\{g6\},\{g7\},\{g8\}\}$ Výběr počátečního podobvodu k expanzi.
- (6) $N=\{\{g2\},\{g3\},\{g7\},\{g8\}\}$ Sousední podobvody k podobvodu *s*.
- (7) $O=\{(\{g2\},\{g2,g5\}),(\{g3\},\{g3,g5\}),(\{g7\},\{g5,g7\}),(\{g8\},\{g5,g8\})\}$
- (9) $\text{min\_in}=2$
- (10) $O=\{(\{g3\},\{g3,g5\}),(\{g8\},\{g5,g8\})\}$
- (11) $\text{min\_out}=2$
- (16) $S=\{\{g1\},\{g2\},\{g3,g5\},\{g4\},\{g6\},\{g7\},\{g8\}\}$ $X=\{\{g1\},\{g2\},\{g4\},\{g6\},\{g7\},\{g8\}\}$ Vybrán podobvod obsahující g3 ke sloučení.
- (6) $N=\{\{g2\},\{g7\},\{g8\}\}$ Po napojení g3 pokračuje znova zvážení možností expanze.
- (7) $O=\{(\{g2\},\{g2,g3,g5\}),(\{g7\},\{g3,g5,g7\}),(\{g8\},\{g3,g5,g8\})\}$
- (…) Obdobně je k podobvodu postupně přidán podobvod obsahující hradlo g8 a podobvod obsahující hradlo g2.
- (6) $S=\{\{g1\},\{g2,g3,g5,g8\},\{g4\},\{g6\},\{g7\}\}$ $X=\{\{g1\},\{g4\},\{g6\},\{g7\}\}, N=\{\{g7\}\}$
- (…) Jediná možnost expanze je připojit podobvod s hradlem g7, což by ovšem překročilo maximální počet vstupů (bod 8). Po kroku 14 pokračuje algoritmus krokem 4.
- (5) $s=\{g6\}, X=\{\{g1\},\{g4\},\{g7\}\}$
- (…) K zvolenému podobvodu je připojen postupně podobvod obsahující g1 a podobvod s g4.

- (6) $S=\left|\{g1,g4,g6\},\{g2,g3,g5,g8\},\{g7\}\right|$
  $X=N=\left|\{g7\}\right|$
- (7) $O=\left|(\{g7\},\{g1,g4,g6,g7\})\right|$
- (8) $O=\emptyset$ Jediná možnost expanze zmíněná v předchozí odrážce výše má 4 vstupy, což je pro dané nastavení nepřijatelné a podobvod tedy nelze dále rozšířit.
- (…) Z bodu 14 pokračuje algoritmus bodem 4.
- (5) $s=\{g7\}, X=\emptyset$
- (6) $N=\left|\{g1,g4,g6\},\{g2,g3,g5,g8\}\right|$
- (7) $O=\left|(\{g1,g4,g6\},\{g1,g4,g6,g7\}),(\{g2,g3,g5,g8\},\{g2,g3,g5,g7,g8\})\right|$
- (8) $O=\emptyset$ Obě varianty byly eliminovány kvůli překročení maximálních rozměrů podobvodu.
- (13) $S=\left|\{g1,g4,g6\},\{g2,g3,g5,g8\}\right|$
  $X=\emptyset$
- (…) Nyní algoritmus v bodě 4 detekuje prázdnou množinu počátečních podobvodů X a přeskakuje na bod 18, kde jsou mezi podobvody vloženy I/O hradla, která budou v tomto případě vložena mezi hradla g4 a g7 a také mezi hradla g5 a g7. Výstupní množina podobvodů obsahuje dva podobvody: $S=\left|\{g1,g4,g6\},\{g2,g3,g5,g8\}\right|$ a hradlo g7 není přiřazeno do žádného podobvodu.

4.1.5 Zpracování úloh

Podobvody nebudou ihned optimalizovány, ale po rozdělení obvodu bude vytvořen objekt pracovní úlohy (WorkUnit, obrázek 7) pro každý podobvod a přidán do fronty úloh. Při generování úlohy je převeden obvod do chromozomu a uložen spolu s rozměry CGP. Počet bloků CGP je určen jako nahoru zaokrouhlený součin počtu hradel původního podobvodu a koeficientu redundance. Z takto nachystané fronty úloh bude možné jak sekvenční tak i paralelní zpracování jednotlivých podobvodů.

4.1.6 Evoluce

Pro evoluci bude použit algoritmus CGP s těmito vlastnostmi:

- strategie výběru 1+λ
- počet mutací na chromozom fixní, nastavitelný
- mutace bodová (změna funkce nebo propojení)
- nastavitelný počet opakování
- nastavitelný počet generací

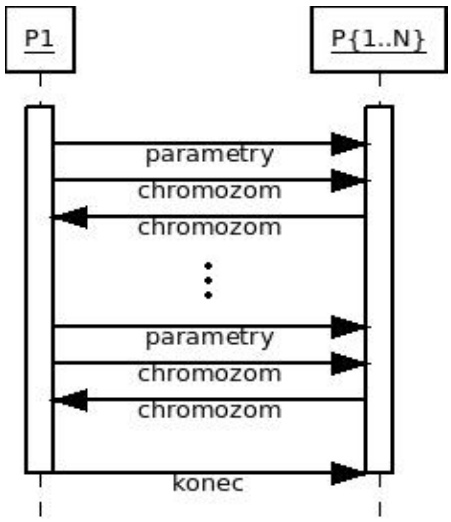
Tyto parametry budou otestovány a v závěru doporučeny jejich hodnoty.

Před začátkem evoluce vybraného podobvodu je vygenerována pravdivostní tabulka popisující jeho chování. Operátor mutace zkouší změnit chromozom definovaným počtem bodových mutací, poté zkontroluje korektnost chování podle vygenerované pravdivostní tabulky a pokud se chování změnilo, zkouší znovu měnit původní chromozom. Tento postup se opakuje, dokud není chromozom po mutaci korektní, tedy dokud se jeho chování neshoduje s požadovaným. Funkce fitness má vždy k dispozici obvod s požadovaným chováním a výpočet hodnoty fitness spočívá v sečtení cen hradel využitých v obvodu.

4.1.7 Paralelizace

Pro urychlení výpočtu bude použita paralelizace na dvou úrovních. První úroveň využívá paralelního zpracování bitových operací dnešními procesory. Předpokládám použití 64-bitových procesorů, tedy v jedné operaci může být zpracováno až 64 bitových operací. Takto bude paralelně simulováno chování logického obvodu při evoluci vybraného podobvodu otestováním všech možných vstupních kombinací a porovnáním odezvy na výstupech podobvodu s požadovanými, které jsou uloženy v pravdivostní tabulce. Díky této paralelizaci je možné urychlit kontrolu chování podobvodu až 64 krát a najednou otestovat všechny vstupní kombinace obvodu s šesti vstupy.

Druhá úroveň bude volitelná a bude využívat více výpočetních jader v rámci jednoho počítače nebo počítačové sítě pomocí OpenMPI knihovny [31]. Jeden výpočetní uzel se bude starat o celý obvod, jeho načítání, rozdělování a ukládání do souboru. Všechny ostatní výpočetní uzly budou provádět pouze CGP algoritmus. Takto bude možné paralelně optimalizovat jednotlivé podobvody na různých výpočetních uzlech. Komunikaci hlavního uzlu (P0) a CGP uzlů (P1..PN) znázorňuje diagram na obrázku 9. Výpočetní uzel vždy obdrží parametry evoluce následované chromozomem, obvod zoptimalizuje a pošle jako chromozom zpět. Tuto sekvenci akcí CGP uzly provádějí dokud neobdrží zprávu o konci optimalizace. Parametry evoluce jsou tyto: počet vstupů, výstupů, potomků, mutací na chromozom, generací, běhů CGP a velikost chromozomu.



Obrázek 9: MPI komunikace.

4.1.8 Zjednodušení obvodu

Po nahrazení podobvodů optimalizovanými podobvody bude celý obvod zjednodušen a poté teprve znovu rozdělen na podobvody. Zjednodušovací funkce bude provádět následující úpravy podobvodu:

- odstranění I/O hradel mezi podobvody,
- odstranění hradel BUF, pokud nejsou mezi globálním vstupem, respektive výstupem, a globálním výstupem,
- odstranění hradel, na která není napojeno žádné hradlo (ani globální výstup),
- úprava hradel se dvěma totožnými vstupy,
- propagace negace,
- propagace konstant ZERO, ONE.

4.2 Implementace

V této kapitole popisuji důležité implementační detaily optimalizačního systému. Nejprve stručně popíši jednotlivé části systému. Dále uvedu způsob určení ceny obvodu, optimalizaci pomocí ABC, problematiku vzniku cyklů v acyklickém grafu a způsob zpracování úloh.

Program může volitelně generovat grafy obvodu a podobvodů v průběhu optimalizace. Vygenerování grafické podoby obvodu z vnitřní reprezentace probíhá pomocí volně dostupného nástroje Graphwiz [33], který z textové definice grafu vygenerované optimalizační metodou vytvoří grafickou podobu.

4.2.1 Součásti systému a pomocné nástroje

*.cpp, *.h

Zdrojové kódy k programu pro evoluční optimalizaci. Pomocí *Makefile* se překládají na sekvenční verzi programu (*optimizer*) a paralelní verzi využívající OpenMpi knihovnu (*optimizer_mpi*).

Makefile

Umožňuje pohodlně překládat program v sekvenční i paralelní verzi, různě jej testovat a generovat grafy obvodů a podobvodů z .dot souborů.

abc_optimizer.sh

Skript pro syntézu a optimalizaci podobvodů pomocí ABC, viz podkapitola 4.2.3.

blifreorder.py

Tento skript přeuspořádá blif soubor namapovaný na hradla z *gates.lib* tak, aby každé hradlo v místě definice předcházela definice jeho vstupů. Program pro evoluční optimalizaci takovéto pořadí předpokládá, ale výstup z ABC jej ne vždy splňuje.

blifdiff.sh

Tento skript za pomoci systému ABC a jeho příkazu *cec* porovnává funkční ekvivalenci zadaných obvodů ve formátu blif.

evaluateBlif.sh

Pomocí tohoto skriptu lze určit cenu obvodu jako sumu cen všech hradel. Cena je definovaná přímo ve skriptu.

4.2.2 Ohodnocení (pod)obvodu

Cenu obvodu či podobvodu určuji jako sumu relativních ploch jednotlivých hradel. Vycházím ze stejných relativních ploch jako použili Vašíček a Sekanina v práci [23], jen je násobím třemi pro odstranění desetinných čísel. Výslednou cenu tedy počítá ABC, CGP i skript *evaluateBlif.sh* podle hodnot v tabulce 3:

Hradlo	Cena
INV	2
NAND, NOR	3
AND, OR	4
XNOR	5
XOR	6
BUF	0

Tabulka 3: Relativní plocha hradel.

Hradlo BUF považuji v této práci za propojku a proto má nulovou cenu. Přílišnému množení hradel BUF obvodu však zabrání zjednodušovací funkce volaná před každým dalším dělením obvodu na podobvody.

4.2.3 Syntéza a optimalizace pomocí ABC

Syntézu a optimalizaci obvodu pomocí systému ABC zprostředkovává skript *abc_optimizer.sh*. Tento skript načte soubor ve formátu pla nebo blif, a po optimalizaci uloží obvod namapovaný na množinu hradel definovanou v knihovně *gates.lib*. Podle této knihovny ABC určuje cenu obvodu. Optimalizace obvodu probíhá iterací dále uvedeného optimalizačního skriptu s nastavitelným počtem opakování. Stejný skript byl použit i v publikacích [21,28]. Ve skriptu je nastavena hodnota 100 iterací, což je pro většinu obvodů naprosto dostatečné. Optimalizace pomocí tohoto skriptu pro ABC má rychlou konvergenci, jak je uvedeno v publikaci [21].

Optimalizační skript

fraig_store; resyn
fraig_store; resyn2
fraig_store; resyn2rs
fraig_store; share
fraig_store; fraig_restore
map

4.2.4 Vznik smyček

Během ladění optimalizačního programu se v obvodu objevily smyčky, kvůli kterým se program zacyklil, neboť předpokládá acyklický graf, kterým kombinační obvod musí být. Smyčka v grafu

reprezentující obvod může vzniknout po nahrazení podobvodu optimalizovaným podobvodem, přestože původní obvod i optimalizovaný podobvod žádnou smyčku neobsahují.

Pokus o zabránění vzniku smyček při dělení na podobvody se nezdařil a navíc byl výpočetně náročný a omezoval možnosti tvorby podobvodů. Problém jsem nakonec vyřešil uchováním původního chromozomu reprezentujícího podobvod, z kterého je možné jej obnovit do původního stavu, pokud by nový podobvod (samostatně acyklický) způsobil cyklus v obvodu. Po optimalizaci každého podobvodu je z chromozomu načteno nové řešení a celý obvod je zkontrolován na výskyt smyček. Pokud je detekován cyklus v grafu, je podobvod nahrazen původní variantou, která smyčku netvořila.

4.2.5 Zpracování úloh

Podobvod k optimalizaci je zapouzdřen objektem pracovní úlohy, v kterém je uchován stav zpracování, původní chromozom pro optimalizaci a případnou obnovu, optimalizovaný chromozom, dimenze podobvodu a odkaz na objekt podobvodu.

Optimalizační algoritmus iteračně provádí následující sekvenci akcí: rozdělí obvod na podobvody a poté ke každému podobvodu vytvoří pracovní úlohu. Dále jsou pracovní úlohy zpracovány a podobvody nahrazeny novými. Nakonec, před dalším dělením na podobvody, je obvod deterministicky upraven zjednodušovací funkcí.

Vlastní zpracování úloh probíhá různě pro sekvenční a pro paralelní variantu programu. V sekvenční variantě je optimalizován jeden podobvod po druhém, v případě paralelního zpracování úloh je situace složitější. Najednou může být zpracováváno až N-1 podobvodů, kde N je počet MPI uzlů. Kromě fronty úloh je v paralelní verzi ještě pole ukazatelů na pracovní úlohu, v kterém má každý CGP uzel odkaz na právě optimalizovanou úlohu kvůli identifikaci při příjmu optimalizovaného chromozomu. Při přiřazení úlohy CGP uzlu je ve frontě označena jako probíhající a další uzly ji při vyhledávání volné práce přeskočí. Pro zlepšení účinnosti paralelní architektury je fronta úloh pro paralelní zpracování přeuspořádána primárně podle počtu vstupů a sekundárně podle velikosti chromozomu, aby byly časově náročnější úlohy zpracovávány dříve.

4.3 Experimenty

Tato podkapitola prezentuje provedené experimenty s implementovanou metodou a jejich výsledky. V podkapitole 4.3.1 je graficky znázorněn průběh optimalizace konkrétního obvodu pomocí této metody. Další podkapitoly jsou věnované testování důležitých parametrů metody a poslední podkapitola obsahuje průběh a výsledky optimalizace několika vybraných obvodů s parametry nastavenými na základě výsledků z předchozích testů. Pro účely testování jsem použil kombinační obvody ze sady LGSynth93. Vstupní obvody pro finální test (podkapitola 4.3.7) byly optimalizovány pomocí 100 iterací skriptu pro ABC. Dřívější experimenty nepracovaly vždy s takto důkladně optimalizovanými počátečními obvody, proto se v grafech vyskytují vyšší hodnoty zlepšení ceny oproti počátečnímu obvodu.

Všechny experimenty jsem prováděl na těchto počítačích:

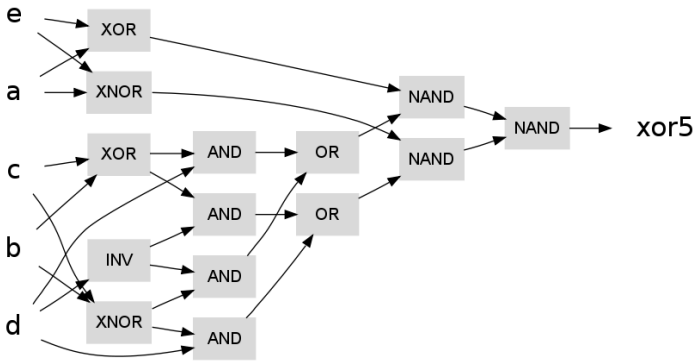
- pc1: AMD FX-8120 @ 3.8GHz 8 jader
- edesign1: Intel Xeon X5650 @ 2.67 6 jader (+ HT)
- edesign2: Intel Xeon X5353 @ 2.66 4 jádra (+ HT)
- edesign3: Intel Xeon E5-2630 @ 2.30 6 jader (+ HT)

Pro většinu experimentů jsem používal počítač pc1, pokud test obsahoval více obvodů, využíval jsem i edesign1 a edesign3 a to tak, že každý počítač prováděl experimenty s jiným obvodem nebo úplně jiné experimenty. Počítač edesign2 jsem využíval jen na pomocné testy, například na prvotní otestování vhodnosti parametrů. Experimenty jsem prováděl se sekvenční verzí, kterou jsem spustil vícekrát (pro každé jádro jeden běh). Tímto jsem mohl oproti verzi s MPI zaručit 100% efektivitu paralelního zpracování a vyloučit komunikační režii. Hyper-Threading (HT) na Intel procesorech se na těchto úlohách výkonově neosvědčil, proto jsem uvažoval pouze počet fyzických, nikoli logických jader. Naproti tomu u AMD FX obsahuje modul bulldozer dvě samostatná jádra, která sdílí některé komponenty, například FP jednotku, ale FX jednotku má každý samostatnou (celočíselnou aritmetiku využívá program velmi často) a i při využití všech 8 jader výkon na jádro neklesal.

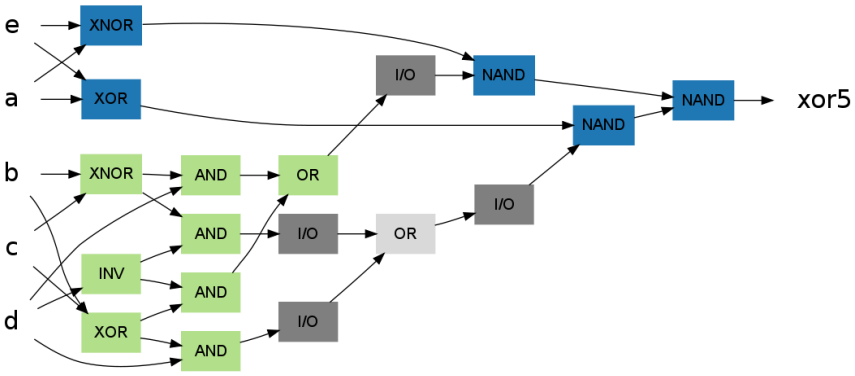
4.3.1 Ukázka průběhu optimalizace

V této podkapitole prezentuji pomocí obrázků 10 až 15 průběh optimalizační metody na malém souboru xor5 ze sady LGSynth93. Jde o velmi jednoduchý obvod, pro jehož optimalizaci není tato metoda potřeba, neboť je lepší jej optimalizovat vcelku, nikoli po částech. Pro tento příklad je povolený počet vstupů podobvodu 1-4 a počet hradel 3-8. Reálně není vhodné optimalizovat tak malé podobvody, jde pouze o demonstraci principu. Dělení obvodu na podobvody není deterministické, což dokládá obrázek 16, na kterém je několik variant rozdělení obvodu xor5 na podobvody o dříve definované velikosti.

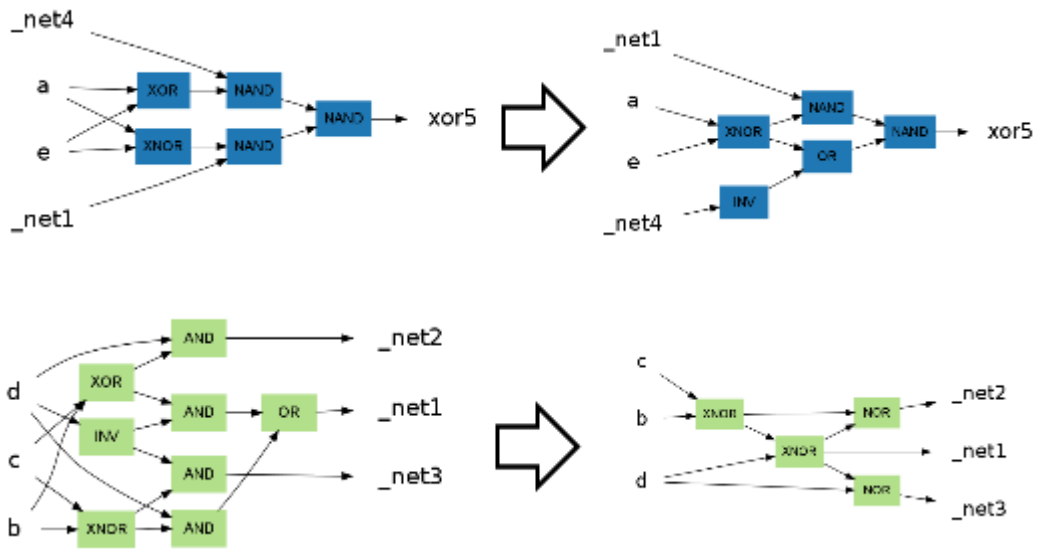
V grafech jsou hradla znázorněna barevně, pokud jsou přiřazena do některého podobvodu. Dále světle šedě, pokud nejsou přiřazena do žádného podobvodu a tmavě šedě jsou označeny speciální vstupně-výstupní hradla mezi podobvody.



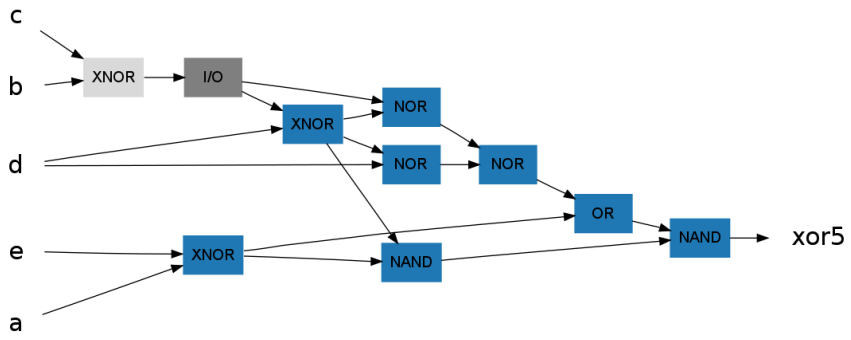
Obrázek 10: Originální obvod.



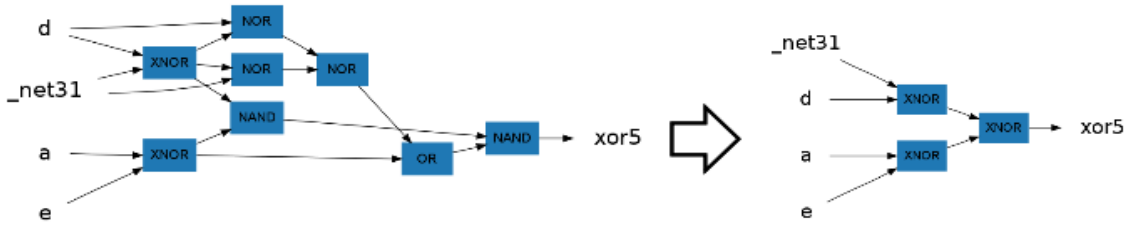
Obrázek 11: Originální obvod rozdělený na podobvody.



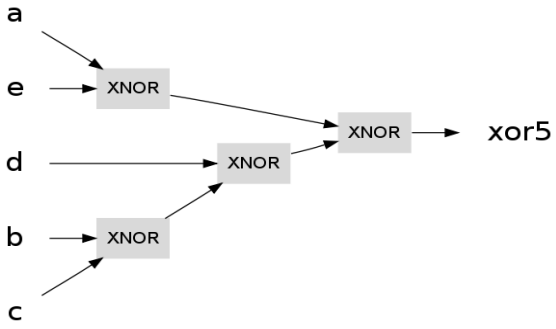
Obrázek 12: Optimalizace podobvodů první epizody.



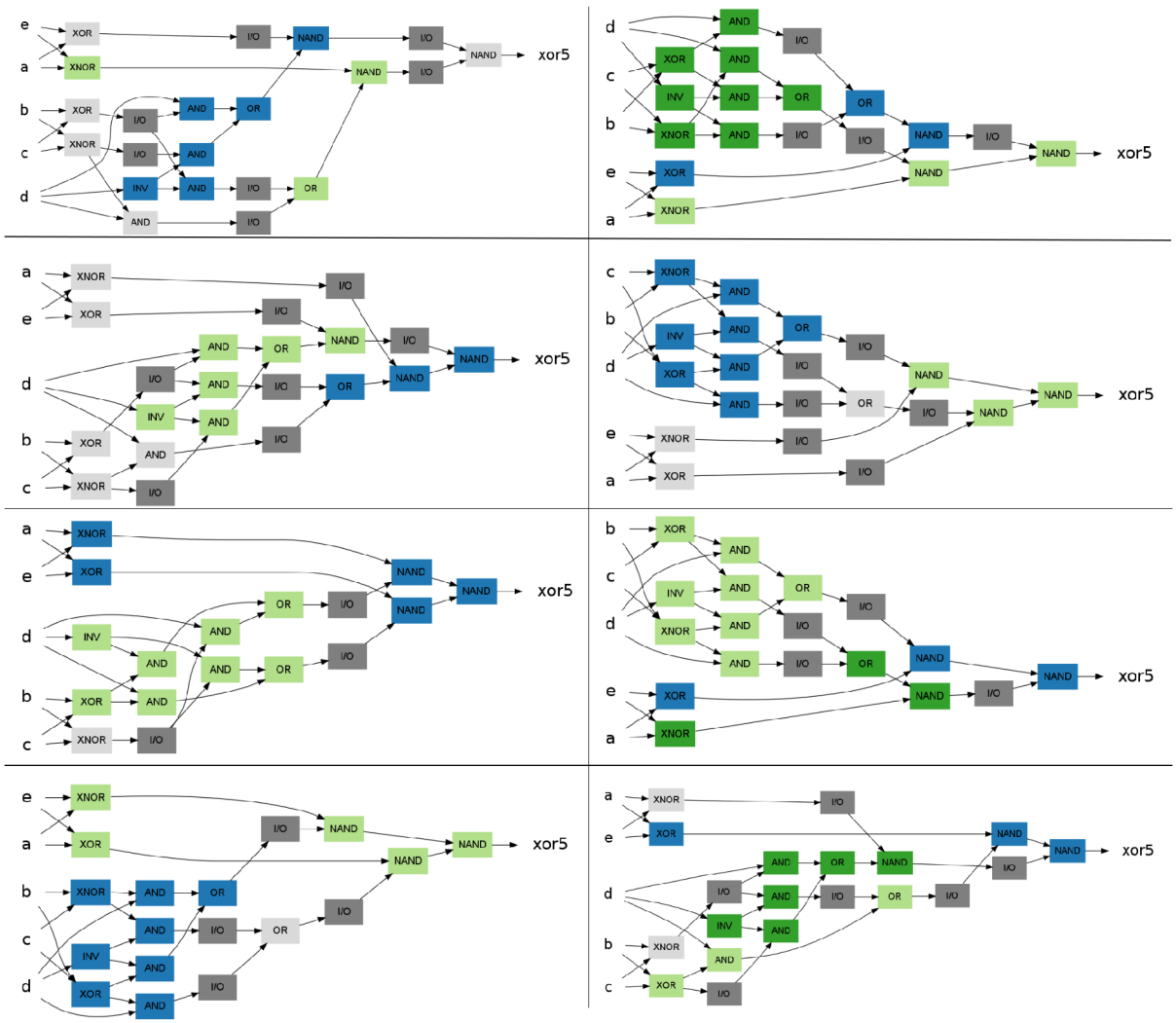
Obrázek 13: Obvod po prvním epizodě rozdělený na podobvody.



Obrázek 14: Optimalizace podobvodu z druhé epizody.



Obrázek 15: Výsledný optimalizovaný obvod.



Obrázek 16: Ukázky několika různých způsobů rozdělení na podobvody.

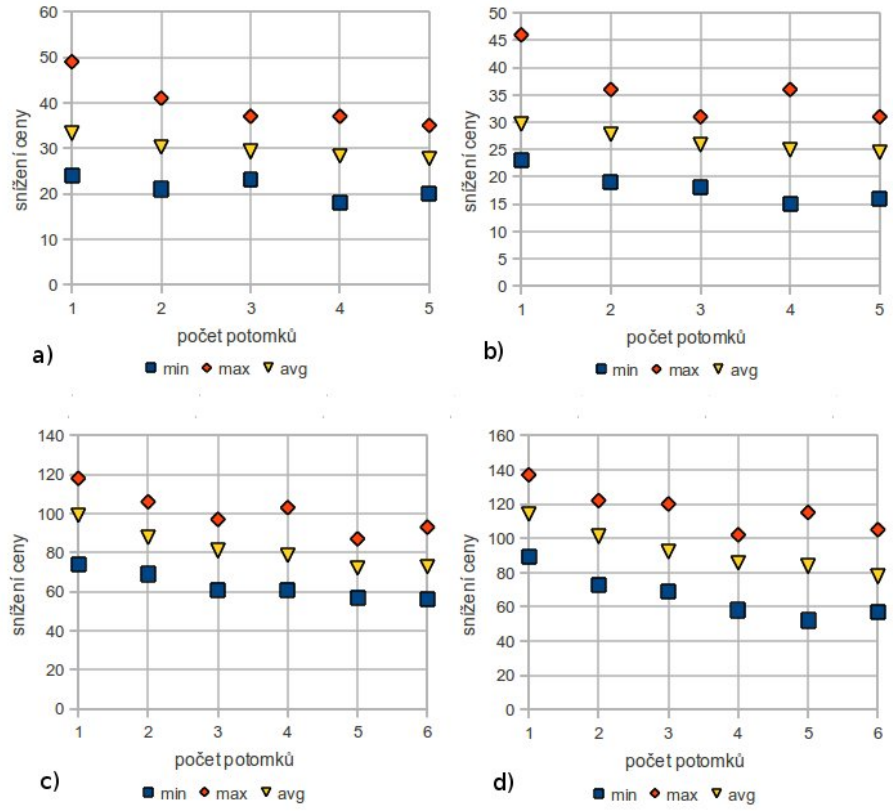
4.3.2 Počet potomků

Jako první parametr jsem otestoval počet potomků, který je najednou vytvářen z rodičovského chromozomu. Test probíhal na obvodech apex2 (varianta a,c) a apex5 (b,d). U každého obvodu jsem test provedl pro malé podobvody a pro větší podobvody, u nichž je navíc zvýšen počet generací kvůli větším chromozomům. Z naměřených dat (obrázek 17) vyplývá nadbytečnost využití více potomků, proto v dalších testech bude vždy jen jeden potomek.

V grafech v této podkapitole i několika následujících podkapitolách je na osu *y* vynesena změna relativní plochy obvodu mezi vstupním a výstupním obvodem optimalizační metody. Pro dané nastavení experimentu jsou v grafech vyjádřeny hodnoty průměrného (avg), maximálního (max) a minimálního (min) snížení ceny ze všech nezávislých běhů.

Parametry

- počet generací 200000 (obr. 16: a, c); 500000 (obr. 16: b,d)
- počet běhů CGP 1
- počet mutací na chromozom 1
- koeficient redundance 2
- doba běhu 30 min
- počet vstupů podobvodu 1 – 10 (a,c); 6 – 12 (b,d)
- počet hradel podobvodu 8 – 20 (a,c); 20 – 100 (b); 30 – 100 (d)
- počet nezávislých běhů 50



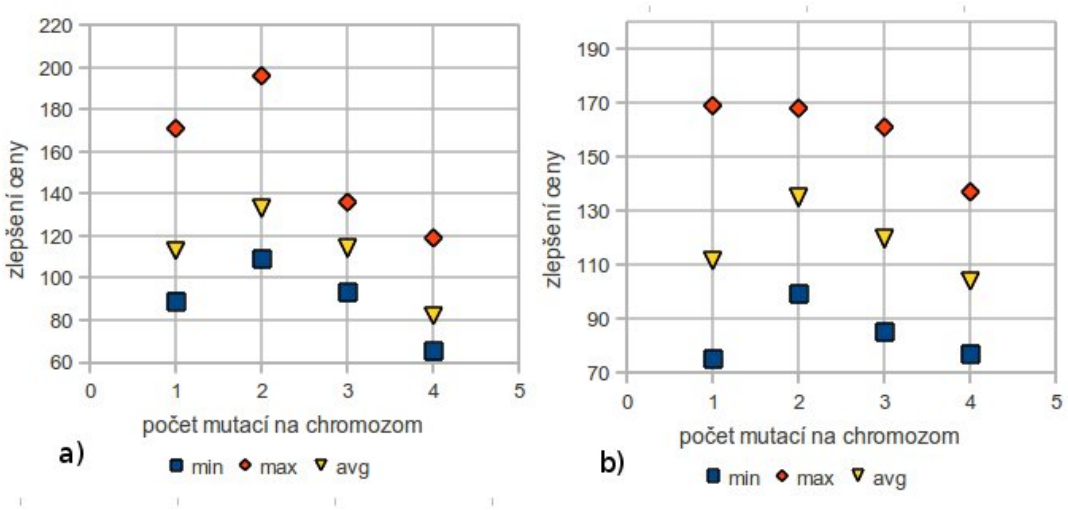
Obrázek 17: Vliv počtu potomků.

4.3.3 Počet mutací na chromozom

V evoluci používám místo pravděpodobnosti mutace počet mutací na chromozom. Test tohoto parametru jsem provedl na obvodu apex2, opět ve dvou variantách lišících se velikostí podobvodů a počtem generací. Pro větší i pro menší podobvody vychází nejlépe varianta s dvěma mutacemi na chromozom, jak ilustruje obrázek 18.

Parametry

- počet generací 30000 (obr. 17: a); 50000 (obr. 17: b)
- počet potomků 1
- počet běhů CGP 5
- koeficient redundance 1,5
- doba běhu 30 min
- počet vstupů podobvodu 3 – 9 (a); 6 – 12 (b)
- počet nezávislých běhů 50
- počet hradel podobvodu 8 – 20 (a); 20 – 100 (b)



Obrázek 18: Vliv počtu mutací na chromozom.

4.3.4 Počet běhů CGP oproti počtu generací

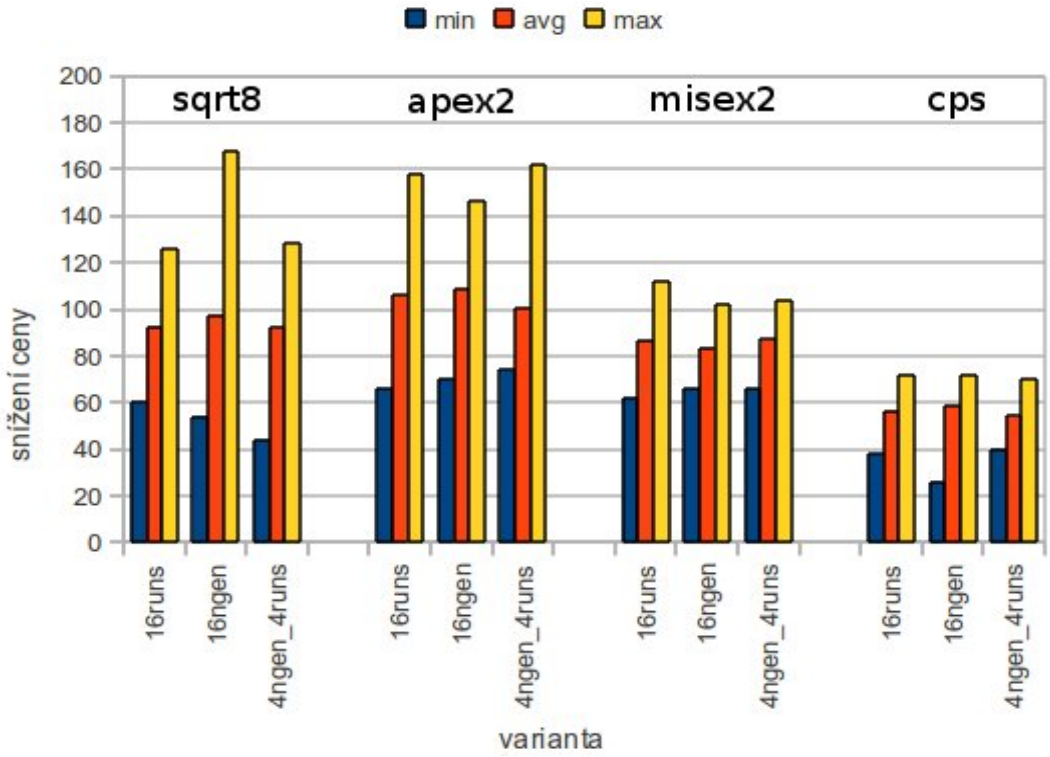
Tímto testem jsem se snažil zjistit, zda je při stejném (podobném) množství vykonané práce lepší při evoluci podobvodu upřednostnit počet generací nebo počet opakování evoluce podobvodu. Zvolil jsem tři varianty rozdělení práce:

- 16runs G=1 R=16 upřednostnění počtu pokusů o optimalizaci podobovdu
- 16ngen G=16 R=1 upřednostnění počtu generací
- 4ngen_4runs G=4 R=4 kompromis mezi předchozími variantami

Proměnná R určuje počet běhů CGP nad optimalizovaným podobvodem a proměnná G počet generací ve statisících. Tyto varianty jsem testoval na čtyřech různých obvodech a výsledky jsou vykresleny v grafu na obrázku 19. Všechny tři varianty produkují srovnatelné výsledky.

Parametry

- počet generací 100000*G $G \in \{1, 4, 16\}$
- počet potomků 1
- počet běhů CGP R $R \in \{1, 4, 16\}$
- počet mutací na chromozom 1
- koeficient redundance 4
- doba běhu 60 min
- počet vstupů podobvodu 3 – 10
- počet hradel podobvodu 8 – 40
- počet nezávislých běhů 20



Obrázek 19: Porovnání variant rozdělení práce.

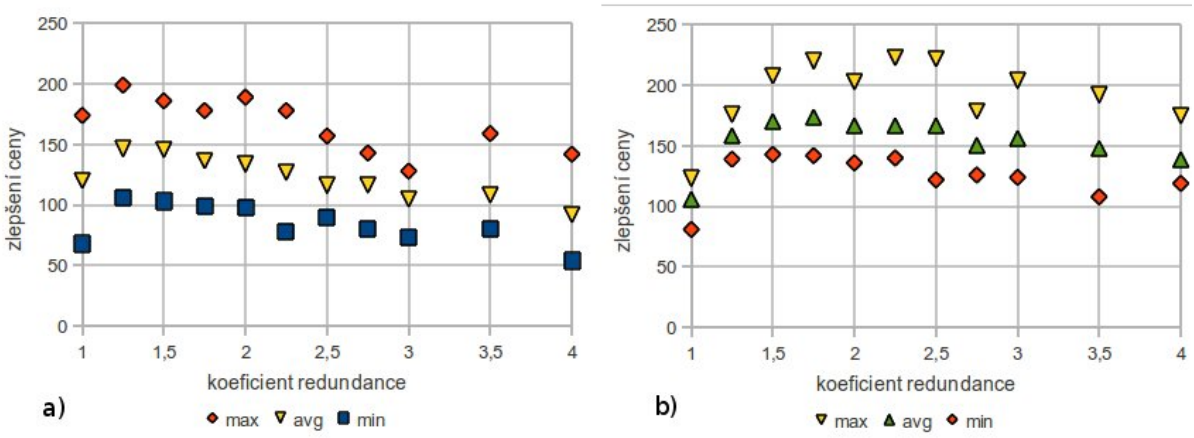
4.3.5 Redundance bloků CGP

Při optimalizaci obvodu pomocí kartézského genetického programování je důležitým parametrem redundance bloků. Pokud je chromozom bez redundance, tj. pokud pole hradel CGP obsahuje právě tolik hradel, jako optimalizovaný podobvod (koeficient 1), může evoluce těžko obvod změnit, pokud je bloků více, může evoluce část obvodu náhodně poskládat v jiné části chromozomu a pak ji při mutaci přepojit za původní. S rostoucí redundancí však roste i doba ohodnocení chromozomu.

Test jsem provedl na obvodu apex2 pro dvě různá nastavení, obdobně jako v předchozí podkapitole. Z dat prezentovaných v grafech na obrázku 20 vyplývá, že vhodnost koeficientu redundance závisí do jisté míry na velikosti podobvodů. Za rozumné nastavení tohoto koeficientu považuji hodnoty 1,25-2,5.

Parametry

- počet generací 100000 (obr. 19: b); 300000 (obr. 19: a)
- počet potomků 1
- počet běhů CGP 5
- počet mutací na chromozom 2
- doba běhu 30 min
- počet vstupů podobvodu 3 – 10 (b); 6 – 12 (a)
- počet hradel podobvodu 8 – 20 (b); 20 – 100 (a)
- počet nezávislých běhů 25



Obrázek 20: Vliv redundance bloků CGP.

4.3.6 Velikost podobvodů

Velikost povolených podobvodů má velký vliv na průběh optimalizace obvodu a lze ji vymezit těmito čtyřmi parametry:

- minimální počet vstupů (min_in),
- maximální počet vstupů (max_in),
- minimální počet hradel (min_g),
- maximální počet hradel (max_g).

Počet výstupů podobvodů není nastavitelný, ale algoritmus pro rozdělení obvodu na podobvody se jej snaží minimalizovat.

Jelikož lze uvažovat řádově tisíce kombinací těchto parametrů, je testování velikosti podobvodů rozděleno do tří fází, které dále popisuji.

První fáze

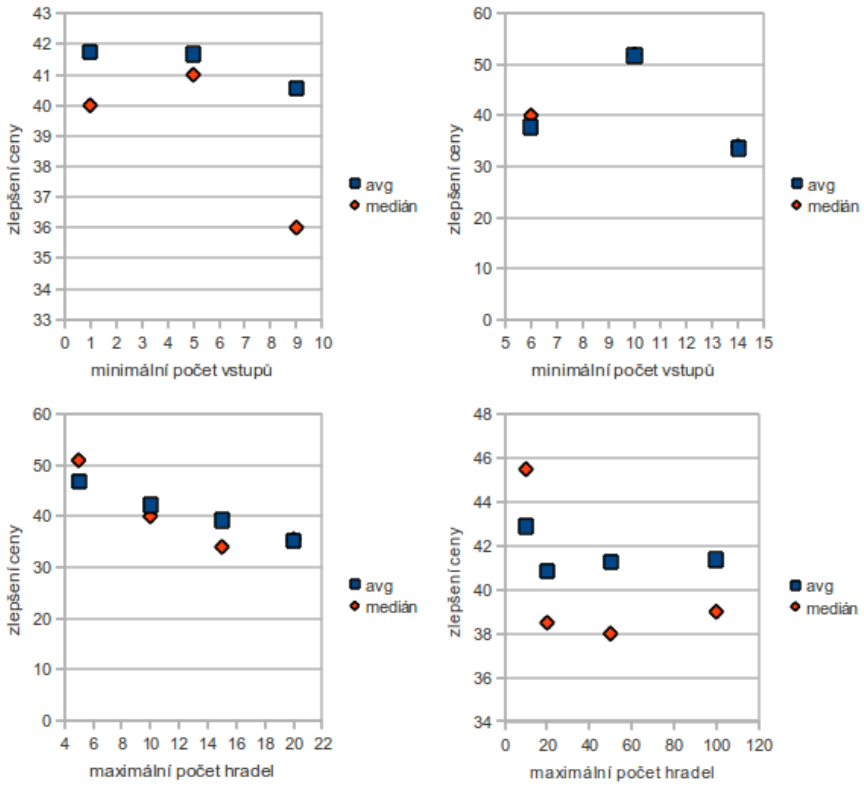
Po předběžném otestování mezních hodnot rozměrů podobvodů jsem provedl test na obvodu apex2 se všemi kombinacemi dále uvedených hodnot kromě těch, kdy je minimální počet vstupů respektive hradel větší než maximální. U každé takovéto velikosti podobvodu jsou dvě varianty lišící se počtem generací, celkem 224 testovaných kombinací parametrů.

- minimální počet vstupů: 1, 5, 9
- maximální počet vstupů: 6, 10, 14
- minimální počet hradel: 5, 10, 15, 20
- maximální počet hradel: 10, 20, 50, 100
- počet generací: 200000,1000000

Vycházel jsem z průměrné hodnoty z 10 běhů pro každou hodnotu a z nich jsem vyjádřil průměr a medián. Z výsledných dat souhrnně zobrazených na obrázku 21 jsem usoudil, že maximální počet vstupů a minimální počet hradel jsou důležitější parametry než minimální počet vstupů a maximální počet hradel. Proto v další fázi budu testovat důkladněji tyto dva parametry.

Parametry první fáze

- počet potomků 1
- počet běhů CGP 1
- počet mutací na chromozom 2
- koeficient redundance 1,5
- doba běhu 20 min
- počet nezávislých běhů 10



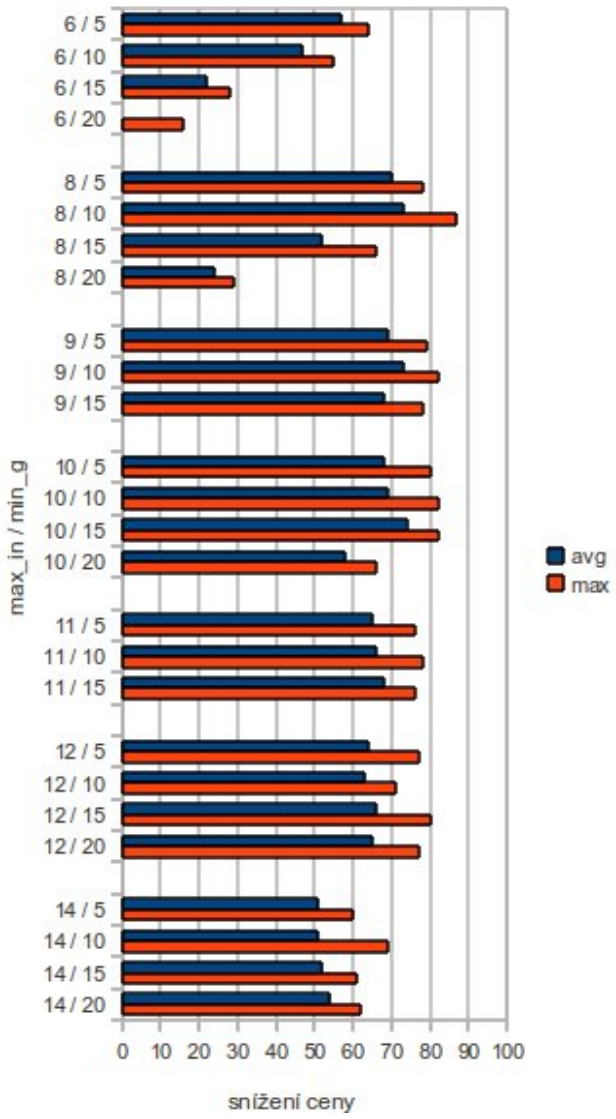
Obrázek 21: Vliv jednotlivých parametrů velikosti obvodů na cenu.

Druhá fáze

Na základě předchozích experimentů jsem zvolil konstantní hodnoty minimálního počtu vstupů a maximálního počtu hradel a testoval zbylé dva parametry určující možnou velikost podobvodu. Testoval jsem méně hodnot, zato důkladněji. Z výsledných dat prezentovaných grafem na obrázku 22 jsem pro další test vybral varianty se stejnou hodnotou maximálního počtu vstupů a minimálního počtu hradel.

Parametry druhé fáze

- počet generací 500000
- počet potomků 1
- počet běhů CGP 1
- počet mutací na chromozom 2
- koeficient redundance 1,5
- doba běhu 60 min
- min. počet vstupů podobvodu 1
- max. počet hradel podobvodu 50
- počet nezávislých běhů 25



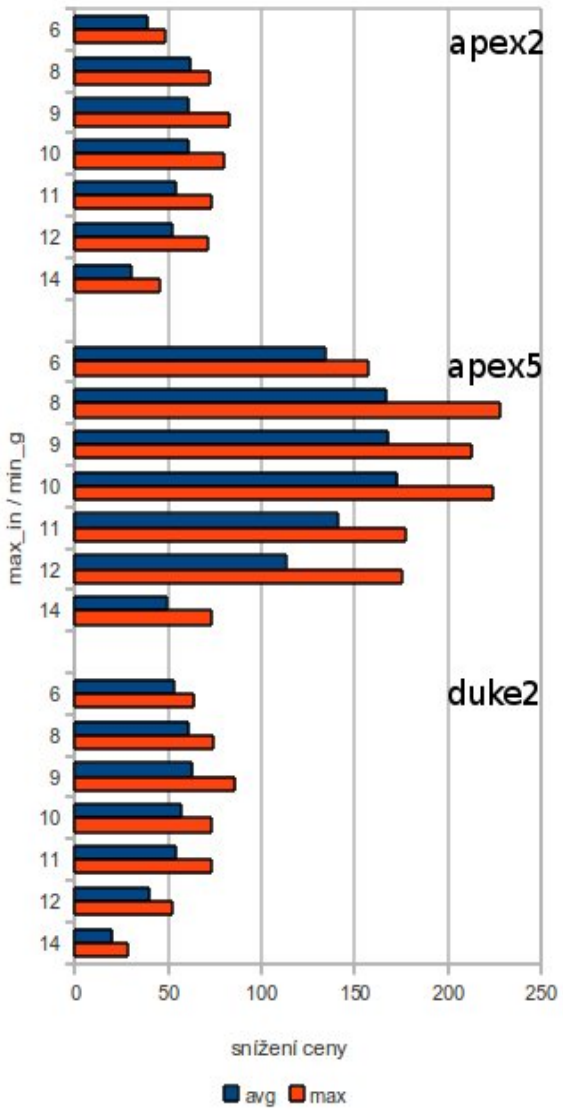
Obrázek 22: Test velikosti podobvodů - druhá fáze.

Třetí fáze

Pro poslední fázi testování velikosti podobvodů jsem zvolil hodnoty parametrů podle výsledků předchozích fází, nastavil delší čas optimalizace a testoval na třech různých obvodech. V této fázi jsem zvolil na základě předchozích výsledků shodné hodnoty maximálního počtu vstupů a minimálního počtu hradel. Každý ze tří obvodů byl optimalizován na jiném počítači a výsledky zobrazuje obrázek 23. U všech tří obvodů se osvědčily hodnoty maximálního počtu vstupů (a shodně i minimálního počtu hradel) 8-10.

Parametry třetí fáze

- počet generací 1000000
- počet potomků 1
- počet běhů CGP 1
- počet mutací na chromozom 2
- koeficient redundance 2,5
- doba běhu 6 hod
- min. počet vstupů podobvodu 1
- max. počet hradel podobvodu 50
- počet nezávislých běhů 25



Obrázek 23: Test velikosti podobvodů - třetí fáze.

4.3.7 Finální test

Pro otestování účinnosti metody jsem zvolil devět různých obvodů ze sady LGSynth93, které jsou uvedeny v tabulce 4. Po předchozích zkušenostech s konvergencí ceny obvodu jsem nastavil dvanáctihodinový běh a větší redundanci. Maximální počet vstupů podobvodů jsem zvolil deset, což je horní hranice doporučených hodnot z předchozí podkapitoly. Více vstupů jsem umožnil v naději ve větší úsporu výsledné ceny obvodu, což ovšem prodlužuje potřebnou dobu evoluce.

Srovnání ceny obvodu je uvedeno oproti původnímu obvodu, který byl optimalizován jako celek iteračně pomocí ABC. Optimalizační skript uvedený v podkapitole 4.2.3 byl aplikován 100x na obvod syntetizovaný též pomocí ABC z pravdivostní tabulky (pla formát).

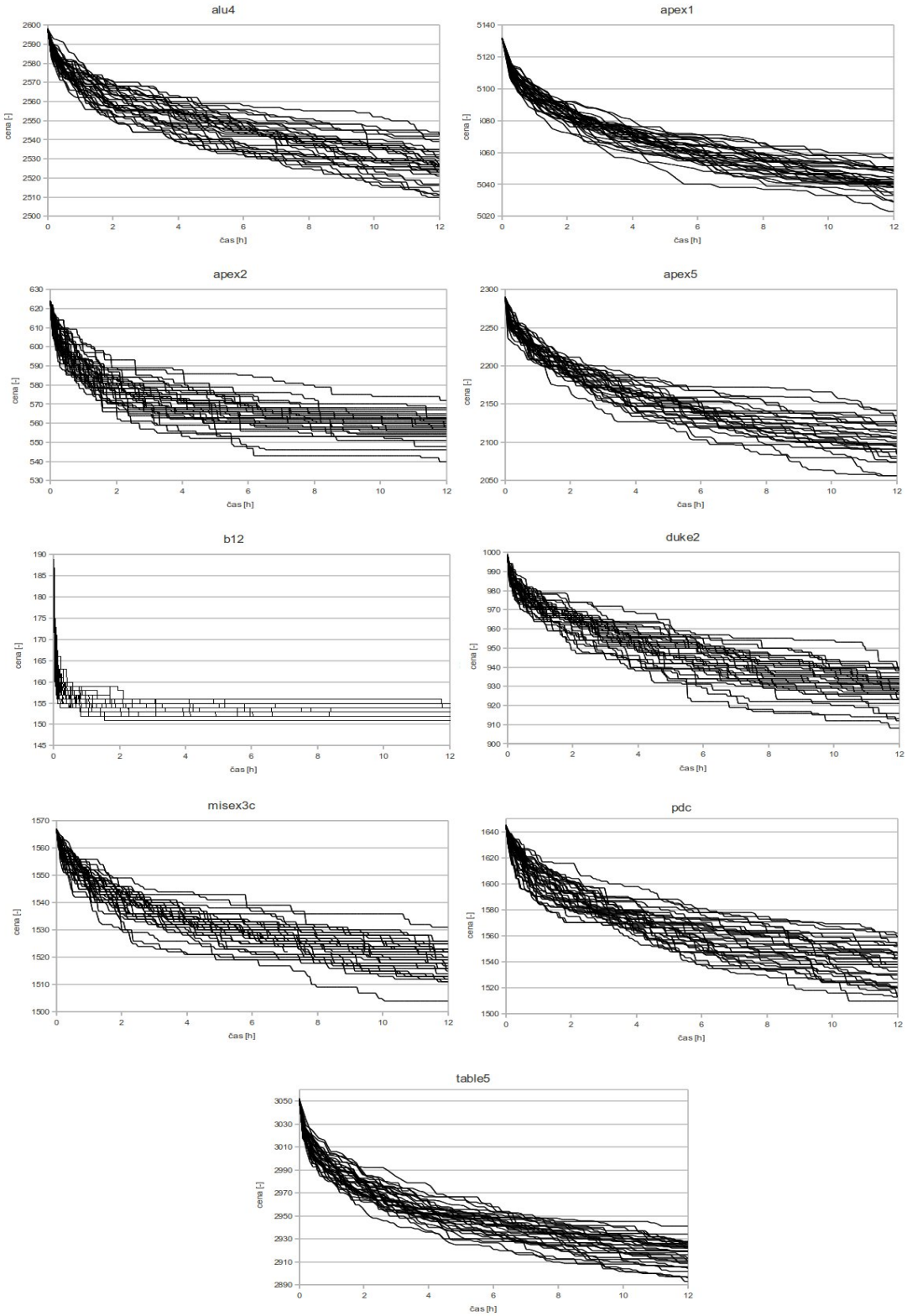
Zlepšení relativní ceny obvodů optimalizovaných pomocí ABC bylo 2-20 % (viz tabulka 3). Největšího snížení ceny metoda dosáhla u obvodu b12 a apex2, což jsou dva nejmenší obvody z devíti testovaných. Naopak největší obvod (apex1) byl nejméně vylepšen. Grafy na obrázku 24 ukazují průběh ceny v čase a dokládají vliv velikosti obvodů. Zatímco se ceny obvodů s méně hradly ke konci běhu metody příliš neměnily, u větších z testovaných obvodů by cena jistě dále klesala, pokud by optimalizace trvala déle. Počet vstupů obvodu ovšem neměl až tak velký vliv na potřebnou dobu optimalizace jako počet hradel. Exponenciální složitost CGP algoritmu se podařilo obejít rozdělením na menší podobvody, jak dokládá výsledek optimalizace obvodu apex5 se 117 vstupy, a 760, u kterého bylo snížení ceny o více než 10 %.

Parametry

- počet generací 1000000
- počet potomků 1
- počet běhů CGP 1
- počet mutací na chromozom 2
- koeficient redundance 2,5
- doba běhu 12 hod
- velikost vstupů podobvodu 1 – 10
- počet hradel podobvodu 10 – 50
- počet nezávislých běhů 35

Obvod	Počet vstupů	Počet výstupů	Počet hradel	Počáteční cena obvodu	Zlepšení ceny [%]	
					avg	max
alu4	14	8	848	2598	2,7	3,4
apex1	45	45	1692	5132	1,7	2,1
apex2	39	3	211	624	10,4	13,5
apex5	117	88	760	2290	8,3	10,2
b12	15	9	61	189	19	20,1
duke2	22	29	329	999	7	9,1
misex3c	14	14	511	1567	3,1	4
pdc	16	40	546	1645	6,6	8,2
table5	17	151	1005	3052	4,4	5,2
průměr	33	28	663	2011	7	8,4

Tabulka 4: Finální test vybraných obvodů.



Obrázek 24: Průběh ceny obvodů během evoluce (35 nezávislých běhů).

4.3.8 Doporučené hodnoty

V této podkapitole navrhuji vhodné hodnoty parametrů optimalizační metody, které však nemusí být nutně optimální a jejich vhodnost se může měnit podle typu a velikosti obvodu. Kombinací hodnot parametrů je mnoho a doba evoluce dlouhá, proto nebylo možné řádně otestovat všechna možná nastavení optimalizace.

Doporučuji jen jednoho potomka a dvě mutace na chromozom při vytváření nové populace. Experimenty dopadly poměrně dobře i pro chromozomy bez redundance, ale za vhodné nastavení považuji hodnoty koeficientu redundance 1,25-2,5. Počet nezávislých běhů CGP na jednom podobvodu bych nevolil větší než 8 a raději bych zvýšil počet generací. Při větší redundanci bloků a dostatečném počtu generací stačí jeden běh CGP. Počet generací potřebný k optimalizaci podobvodů závisí na velikosti podobvodů, koeficientu redundance a počtu potomků a osvědčil se v rozmezí 100000-5000000. Potřebná doba pro optimalizaci značně závisí na velikosti obvodu. Pro obvody o stovkách až tisících hradlech je zapotřebí mnoha hodin až několika dnů. Počet vstupů obvodu není nikterak důležitý díky dělení na podobvody o zadaném maximálním počtu vstupů.

Velikost podobvodů je pro tuto metodu zásadní, proto jsem ji testoval nejdůkladněji. Málokdy se vyskytne podobvod o menším počtu vstupů než je maximální povolený, proto je vhodné minimální počet vstupů neomezovat a nastavit na 1. Naopak maximální počet vstupů je nejdůležitějším parametrem velmi ovlivňující velikost i dobu ohodnocení podobvodu. V testech které jsem prováděl měla metoda nejlepší výsledky pro hodnoty maximálního počtu vstupů 8-10. Minimální počet hradel jsem volil stejný jako maximální počet vstupů. Je málo pravděpodobné, že evoluce vylepší obvod o méně než šesti hradlech. Příliš velká hodnota minimálního počtu hradel vede k omezení počtu podobvodů a menšímu počtu hradel přidělených do nějakého podobvodu. Kvůli době evoluce jsem také omezil maximální počet hradel podobvodu na 50, většina podobvodů je menší a má 10-20 hradel.

5 Závěr

V rámci diplomové práce jsem prostudoval a zpracoval tématiku číslicových obvodů, metody jejich optimalizace a kartézské genetické programování.

Na základě prostudovaných teoretických informací jsem vytvořil metodu pro evoluční optimalizaci kombinačních logických obvodů pomocí kartézského genetického programování po částech. Diplomová práce popisuje návrh této metody nejprve jako celku a poté jejích detailů, z nichž je nejpodstatnější algoritmus pro dělení obvodu na podobvody a vnitřní model obvodu. Práce dále obsahuje popis implementace některých částí, například způsob ohodnocení (pod)obvodu či syntézu a optimalizaci obvodů pomocí ABC. S vytvořenou evoluční metodou jsem provedl řadu experimentů.

Postupně jsem testoval vliv jednotlivých parametrů metody a na konci podkapitoly 4.3 doporučuji vhodné parametry metody na základě předchozích experimentů. Navržená metoda uspěla při optimalizaci obvodu apex5 se 117 vstupy a 760 hradly, u kterého za 12 hodin snížila cenu o více než 10%. Optimalizační metoda dobře zvládá obvody s mnoha vstupy, neboť optimalizované podobvody mají jen tolik vstupů, aby ohodnocení chromozomu trvalo přijatelnou dobu. Velké obvody, například apex1 s 1692 hradly, však stále potřebují dlouhou dobu optimalizace, řádově dny. Na zvolené sadě devíti testovacích obvodů byla dosažena v průměru úspora plochy 7 % oproti počátečním obvodům optimalizovaných pomocí ABC.

Jako rozšíření této metody je možné zahrnout do optimalizačních kritérií další vlastnosti, zejména zpoždění kombinačního obvodu, logický zisk hradel či příkon obvodu. Také je možné použít jinou množinu hradel, například nestandardní hradla prezentovaná v práci [24]. Dalším testováním je možné nalézt parametry, při kterých bude metoda pracovat efektivněji. V neposlední řadě se nabízí možnost střídavého použití této metody a deterministického systému ABC v průběhu optimalizace. V takovém případě nemusí být nastavení velikosti podobvodů vždy stejné, ale může se v průběhu optimalizace měnit.

Literatura

[1] Wikipedia contributors. *Moore's law: Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 8 Nov. 2012. [online]. c2012 [cit. 2012-11-22]. Dostupný z: <en.wikipedia.org/w/index.php?title=Moore%27s_law&oldid=521966242>.

[2] Český statistický úřad. *Informační společnost v číslech: 2012 Česká republika a EU.* Praha, 2012. 76 s. Dostupné z: <www.czso.cz/csu/2012edicniplan.nsf/p/9705-12>. ISBN 978-80-250-2171-2.

[3] WAKERLY, J. F. *Digital design: principles and practices.* Fourth Edition. New Jersey, 2006. ISBN 0-13-186389-4.

[4] GAJDA, Z.; SEKANINA, L. *An Efficient Selection Strategy for Digital Circuit Evolution.* In: Evolvable Systems: From Biology to Hardware, Berlin, DE, Springer, 2010, s. 13-24. ISBN 978-3-642-15322-8.

[5] MEINEL, C.; THEOBALD, T. *Algorithms and Data Structures in VLSI Design. OBDD – Foundations and Applications.* Berlin: Springer, 1998. ISBN 3-540-64486-5.

[6] *Berkeley Logic Interchange Format (BLIF).* Berkeley: University of California, 2005. Dostupné z: <www.cs.uic.edu/~jlillis/courses/cs594/spring05/blif.pdf>.

[7] ABC: A System for Sequential Synthesis and Verification. [online]. [cit 2012-12-03]. Dostupné z: <www.eecs.berkeley.edu/~alanmi/abc/>.

[8] Elgris / EDIF Implementation. [online]. [cit. 2012-12-03]. Dostupné z: <www.elgris.com/content/edif_overview.html>.

[9] KUEHLMANN, A.; PARUTHI, V.; KROHM, F.; GANAI, M. *Robust boolean reasoning for equivalence checking and functional property verification*. IEEE Trans. on CAD, 21(12), 2002, s. 1377-1394.

[10] BIERE, A. *The AIGER And-Invert Graph (AIG) Version 20070427.* Johannes Kepler University, 2007. Dostupné z: <fmv.jku.at/aiger/format-20070427.pdf>.

[11] SENTOVICH, E. M. et al. *SIS: A System for Sequential Circuit Synthesis.* Electronics Research Labratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992. Dostupné z: <www.eecs.berkeley.edu/~alanmi/courses/2008_290A/papers/sis_paper.pdf>.

[12] *The VIS Home Page.* [online]. Poslední aktualizace 2012-02-05. [cit. 2012-12-09]. Dostupné z <vlsi.colorado.edu/~vis/>.

[13] MVSIS: Logic Synthesis and Verification. [online]. Poslední aktualizace 2005-12-10. [cit. 2012-12-09]. Dostupné z <embedded.eecs.berkeley.edu/mvsis/>.

[14] Petr Fišer. BOOM – The BOOlean Minimizer. [online]. Poslední aktualizace 2006-08-02. [cit. 2012-12-09]. Dostupné z <service.felk.cvut.cz/vlsi/prj/BOOM/>.

[15] Espresso. [cit 2012-12-09]. Dostupné z: <embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm>.

[16] MILLER, J. F. *Cartesian Genetic Programming.* Natural Computing Series. Springer, Berlin 2011. ISBN 978-3-642-17309-7.

[17] SEKANINA, L. et al. *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům.* Praha: Akademia, 2009, 328 s. ISBN 978-80-200-1729-1.

[18] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, USA: MIT, 1992. ISBN 0-262-11170-5.

[19] KOZA, J. R. *25 Patents—John R. Koza.* [online]. [cit. 2012-12-13]. Dostupné z <www.genetic-programming.com/patents.html>.

[20] GAJDA, Z.; SEKANINA, L. Recent Advances in Evolutionary Synthesis and *Optimization of Ordinary and Polymorphic Circuits*. Brno, FIT VUT, 2011, 111s. ISBN 978-80-214-4417-1.

[21] FIŠER, P; SCHMIDT, J.; VAŠÍČEK, Z.; SEKANINA, L. *On Logic Synthesis of Conventionally Hard to Synthesize Circuits Using Genetic Programming.* Proc. of 13th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS'10), Vienna (Austria), 14.-16.4.2010, s. 346-351.

[22] VAŠÍČEK, Z.; SEKANINA, L. *A Glogal Postsynthesis Optimization Method for Combinational Circuits.* In: Proc. of the Design, Automation and Test in Europe DATE 2011. Grenoble, FR: EDAA, 2011, s. 1525-1528. ISBN 978-3-9810801-7-9.

[23] VAŠÍČEK, Z.; SEKANINA, L.: *On Area Minimization of Complex Combinational Circuits Using Cartesian Genetic Programming.* In: 2012 IEEE World Congress on Computational Intelligence. CA, US: IEEE, 2012, s. 2379-2386. ISBN 978-1-4673-1508-1.

[24] GAJDA, Z.; SEKANINA, L. *Reducing the Number of Transistors in Digital Curcuits Using Gate-Level Evulutionary Design*. In: Genetic and Evolutionary Computation Conference. Association for Computing Machinery, New York, 2007. s. 245-252. ISBN 9781595936974.

[25] SEKANINA, L. *Evolutionary Design of Digital Circuits: Where Are Current Limits?*. In: Proc. of the 1st NASA/ESA Conference on Adaptive Hardware and Systems. Piscataway, US: IEEE CS, 2006, s. 171-178. ISBN 0-7695-2614-4.

[26] SEKANINA, L.; VAŠÍČEK, Z.. *On the Practical Limits of the Evolutionary Digital Filter Design at the Gate Level.* In: Applications of Evolutionary Computing. Berlin, DE: Springer, 2006, s. 344-355. ISBN 978-3-540-33237-4.

[27] PAYNE, A. J.; STEPNEY, S. *Representation and structural biases in CGP.* In: IEEE Congress on Evolutionary Computation. Piscataway, NY, USA: IEEE, 2009, s. 1064-1071. ISBN 978-1-4244-2958-5.

[28] FIŠER, P.; SCHMIDT, J. *It Is Better to Run Iteratire Resynthesis on Parts of the Circuit.* In: Proceedings of the 19th International Workshop on Logic and Synthesis. Irvine, California, USA: University of California, 2010, s. 17-24.

[29] FIŠER, P.; SCHMIDT, J. *Improving the Iterative Power of Resynthesis.* In: Proc. of 15th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS'12). Tallinn, Estonia, 2012, s. 30-33.

[30] ŠLAPAL, J. *Matematické struktury v informatice.* MAT – Studijní opora. Fakulta informačních technologií, Vysoké učení technické v Brně. Brno, 2006.

[31] *Open MPI: Open Source High Performance Computing*. [online]. [Cit. 12.4.2013]. Dostupné z <www.open-mpi.org>.

[32] Graphviz - Graph Visualization Software. [online]. [Cit. 12.5.2013]. Dostupné z <http://www.graphviz.org>.