VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

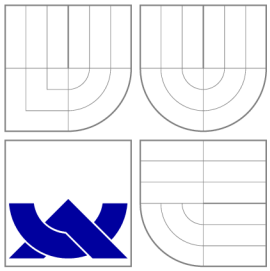
KARTÉZSKÉ GENETICKÉ PROGRAMOVÁNÍ S DYNAMICKOU MODIFIKACÍ PARAMETRŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

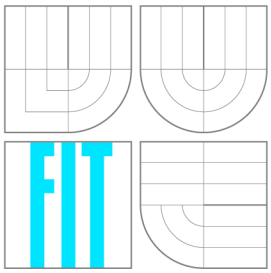
AUTOR PRÁCE
AUTHOR

KAROLÍNA HAJNÁ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KARTÉZSKÉ GENETICKÉ PROGRAMOVÁNÍ S DYNAMICKOU MODIFIKACÍ PARAMETRŮ

CARTESIAN GENETIC PROGRAMMING WITH DYNAMIC PARAMETERS MODIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAROLÍNA HAJNÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá návrhem kombinačních obvodů pomocí kartézského genetického programování. Zkoumá možnosti použití třívstupových look-up tabulek v kombinaci s dynamickou redukcí množiny funkcí. Jsou prezentovány výsledky experimentů, které porovnávají CGP s dvouvstupovými hradly, CGP s třívstupovými look-up tabulkami a CGP s třívstupovými look-up tabulkami s použitím dynamické redukce množiny funkcí. Jednotlivé metody pro dynamickou redukci jsou vysvětleny a vzájemně porovnány.

Abstract

This bachelor's thesis deals with design of combinational circuits through the use of cartesian genetic programming. It examines possibilities of using 3-input look-up tables in combination with dynamic function set reduction. Results of experiments which compare CGP with 2-input gates, CGP with 3-input look-up tables and CGP with 3-input look-up tables and dynamic reduction of function set are presented. Each method of dynamic reduction is explained and these are compared to each other.

Klíčová slova

kartézské genetické programování, evoluční design, look-up tabulka, číslicový obvod

Keywords

cartesian genetic programming, evolutionary design, look-up table, digital circuit

Citace

Karolína Hajná: Kartézské genetické programování
s dynamickou modifikací parametrů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Kartézské genetické programování s dynamickou modifikací parametrů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Prof. Ing. Lukáše Sekaniny, Ph.D

.....
Karolína Hajná
18. května 2015

Poděkování

Tímto bych chtěla poděkovat svému vedoucímu Prof. Ing. Lukáši Sekaninovi, Ph.D za rady při psaní bakalářské práce.

Obsah

Kapitola 1

Úvod

Pro celou řadu problémů dosud neexistují specializované algoritmy, které by poskytovaly dobrý výsledek v přiměřeném čase. Patří sem zejména různé optimalizační úlohy, které jsou časově náročné a běžné metody při jejich řešení mohou uváznout v lokálním extrému.

Úlohou spadající do této oblasti je i návrh kombinačních obvodů. Kromě klasických metod pro jejich návrh je využíván také evoluční návrh, který je stále vyvíjen a zdokonalován. Princip evolučních algoritmů (EA) je vysvětlen v kapitole 2. Jedním z EA, které se k návrhu kombinačních obvodů používají, je kartézské genetické programování (CGP), kterému se tato práce věnuje a je popsáno v kapitole 3.

V tomto textu jsou prezentovány výsledky hledání nových možností, jak snížit časovou složitost CGP a umožnit tak návrh komplexnějších obvodů. Důraz je kladen na obvody sestavené ze třívstupových look-up tabulek (LUT). Evoluční návrh s vícevstupovými LUT je obtížný, protože vyžaduje rychlou paralelní simulaci kandidátních řešení, jejíž akcelerace je netriviální, avšak nutná pro smysluplnou evoluci..

Byly provedeny experimenty (kapitola 5) s nejpoužívanější variantou CGP, která používá dvouvstupová hradla. Jejich výsledky poté byly porovnány s výsledky experimentů s CGP s třívstupovými LUT a plnou množinou funkcí. Pro obvody, které se při experimentech navrhovaly, byla manuálně sestavena množina funkcí umožňující nalézt řešení během menšího počtu evaluací. Posledním krokem bylo navržení několika úprav algoritmu, které by umožnily redukovat množinu funkcí za běhu tak, aby došlo ke zmenšení množství možných kandidátních řešení a pro algoritmus bylo snažší najít plně funkční řešení. Jejich výsledky byly opět srovnány oproti základní variantě, aby se zjistilo, jestli některá z metod přináší zlepšení efektivity algoritmu. Všechny navržené metody jsou popsány v kapitole 4.

V závěru (kapitola 6) jsou shrnuty získané poznatky a navrženy možnosti, jak v práci pokračovat.

Kapitola 2

Evoluční algoritmy

Termínem evoluční algoritmy (EA) [8] [2] se označují různé stochastické algoritmy prohledávání stavového prostoru, které používají množinu kandidátních řešení k vytváření nových kandidátních řešení pomocí biologií inspirovaných operátorů. Stejně jako v přírodě mají nejvyšší šanci přežít a reprodukovat se nejsilnější jedinci (nejlepší řešení úlohy). Prohledávací strategie je založena na populacích.

Cílem EA je nalezení optimálního řešení, popř. co nejlepšího řešení. Jsou vhodné pro složité problémy, při jejichž řešení se prochází velký stavový prostor a hrozí uváznutí v lokálních extrémech. Evoluční algoritmy často poskytují velmi dobré výsledky i u problémů, kde nejsme schopni stanovit hodnotu nejlepšího možného řešení, které se snažíme nalézt. Tyto algoritmy vždy poskytují alespoň nějaké řešení.

2.1 Předpoklady použití

Abychom mohli evoluční algoritmus použít, musíme být schopni potenciální řešení problému zakódovat jako řetězec. Způsob kódování řešení závisí na řešeném problému. Zatímco u některých problémů nám může stačit binární kódování, jinde použijeme celočíselné nebo řetězec znaků. Nalezení vhodného kódování je důležitým předpokladem pro efektivní činnost algoritmu. [4]

Dále musíme umět potenciální řešení ohodnotit. K tomuto účelu se používají rozmanité fitness funkce. Stejně jako v případě reprezentace platí, že způsob, jakým se kandidátní řešení hodnotí, záleží na zadaném problému. Špatně navržená funkce může vést k neefektivnímu nebo dokonce nepoužitelnému algoritmu, protože dojde k degradaci kandidátních řešení a EA nebude schopen nalézt vyhovující řešení ani ve velmi dlouhém časovém úseku.

2.2 Pojmy

EA adoptují některé pojmy z biologie. Nejdůležitější jsou:

Gen
- základní stavební jednotka chromozomu
- jeho hodnota (alela) patří do definované abecedy

Chromozom
- složen z genů

- může mít proměnnou velikost
- také označován jako genotyp, jedinec, individuum

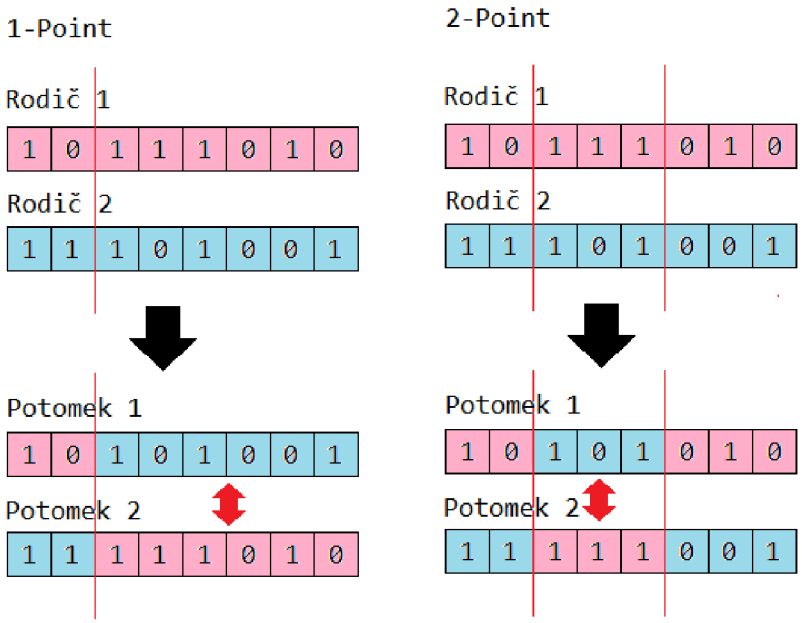
Fenotyp
- sestaven podle obsahu chromozomu
- potenciální řešení

Populace
- složena z jedinců
- obvykle pevná velikost

2.3 Genetické operátory

Abychom mohli v průběhu evoluce vytvářet nové jedince, potřebujeme genetické operátory. Tyto operátory se aplikují na jednoho nebo více jedinců že stávající populace. Rozlišujeme čtyři základní operátory - křížení, mutace, selekce a nahrazení.

K provedení křížení potřebujeme vybrat z populace dva rodiče. Náhodně si v nich zvolíme jeden (jednobodové křížení) nebo více (vícebodové křížení) bodů křížení a takto vzniklé části chromozomů se prohodí. Další varianty křížení jsou uvedeny v [10] . Křížení je aplikováno s určitou pravděpodobností. Jeho princip je na obrázku 2.1.

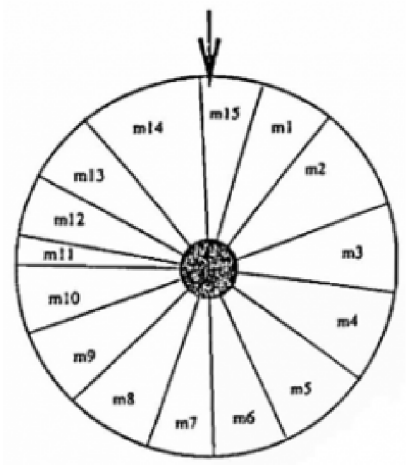


Obrázek 2.1: Křížení v evolučních algoritmech, převzato z [2]

Mutace spočívá v náhodné změně jednoho nebo více genů jedince. Tento operátor bývá použit s malou pravděpodobností.

Pro výběr rodičů ke křížení a jedinců do nové populace slouží selekce. Je několik strategií, jak taková kandidátní řešení vybrat. Jedním ze způsobů je ruleta (viz obrázek 2.2), kdy je vypočtena relativní fitness jedince vůči celé populaci. K výběru dochází pomocí pravděpodobnosti odpovídající relativní hodnotě fitness. Jedinci v vyšší hodnotou fitness mají vyšší šanci být vybrání. Další možností je výběr podle pořadí, kdy se jedince seřadí

podle fitness. K výběru dochází na rozdíl od předchozího případu podle pořadí. Dalším ze způsobů je turnajová metoda. Z populace je náhodně vybráno k jedinců a z nich je vybrán nejlepší jedinec. Tento proces se opakuje dokud potřebujeme jedince vybírat. Jinou možností je deterministický výběr na základě fitness.



Obrázek 2.2: Selekce pomocí rulety, převzato z [2]

Nahrazení se provádí v okamžiku, kdy potřebujeme sestavit novou populaci z populace původní a nově vygenerovaných potomků. Existuje několik základních postupů. Nová populace může být tvořena n jedinci z původní populace a m potomky. Druhým postupem je sestavení nové populace výhradně z potomků. Lze také aplikovat elitismus, kdy se do nové populace vždy dostane nejlepší jedinec ze staré populace.

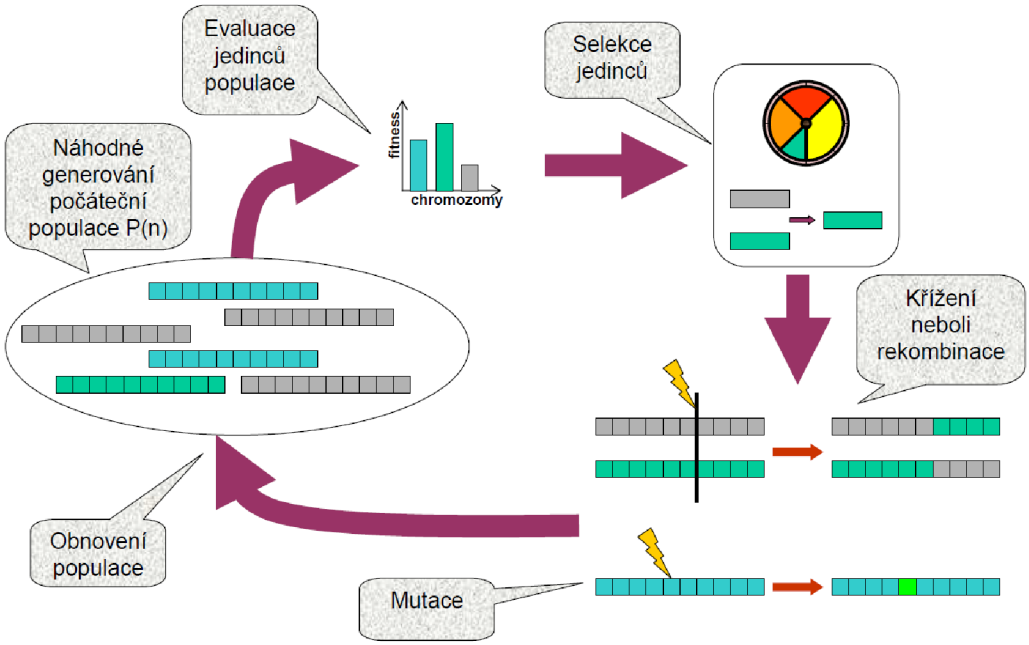
Opět platí, že vybíráme genetické operátory vhodné pro daný problém. Špatnou volbou bychom nemuseli vůbec dosáhnout řešení. Důležitou roli hraje také nutnost oprav chromozomu po aplikaci operátorů. Chromozomy mohou mít pouze omezenou množinu povolených kombinací jednotlivých genů. Aplikace operátorů může mít za následek, že chromozom obsahuje jednu z nepovolených kombinací. Zde přichází na řadu opravy, které mohou být výpočetně náročné. Z toho důvodu, pokud je to možné, se jim snažíme vyhýbat nebo je alespoň minimalizovat.

2.4 Algoritmus

Princip evolučních algoritmů je jednoduchý. Pracují s následujícím algoritmem:

1. Vygenerování počáteční populace
2. Ohodnocení každého kandidátního řešení pomocí fitness funkce
3. Výběr vhodných jedinců ze stávající populace
4. Vytvoření nové populace z vybraných jedinců pomocí zvolených operátorů
5. Ohodnocení každého kandidátního řešení nové populace
6. Pokud je splněna ukončující podmínka, kandidátní řešení s nejvyšší hodnotou fitness je výsledkem evolučního algoritmu. Pokud není splněna, pokračuje se bodem 3

Grafické znázornění tohoto postupu je na obrázku 2.3.



Obrázek 2.3: Evoluční algoritmus, převzato z [8]

2.5 Varianty evolučních algoritmů

V oblasti evolučních algoritmů můžeme nalézt několik přístupů k problematice. Liší se hlavně v používaných genetických operátorech a reprezentaci řešení. Dále si uvedeme hlavní známky nejčastějších variant EA.

Genetické algoritmy používají binární nebo celočíselné chromozomy pevné délky. Ke generování potomků využívají křížení a mutaci.

U genetického programování se jedná o evoluci spustitelných struktur, které jsou nejčastěji reprezentovány jako stromy nebo jiné grafy. Jako genetické operátoy používá křížení a mutaci. Typická je velká populace s malým počtem generací.

V případě evoluční strategie se chromozom kóduje reálnými čísly a zároveň jsou v něm zakódovány strategické parametry (míra mutace ap.). Ze všech variant evolučních algoritmů má nejpropracovanější teoretické základy. Často používá pouze mutaci.

Poslední variantou je evoluční programování, které bylo původně navrženo k návrhu automatů. Používá pouze mutaci.

Všechny tyto přístupy se často vzájemně kombinují podle toho, jak jsou vhodné pro zadaný problém. Z tohoto důvodu lze zřídkakdy EA jednoznačně zařadit do jedné z popsaných kategorií.

2.6 Hodnocení účinnosti algoritmu

Zásadní vliv na dobu nalezení řešení má především počet generací G a velikost populace P. Mluvíme o počtu evaluací, což je součin $G \cdot P$. Podaří-li se nám snížit počet evaluací, klesne i časová složitost algoritmu. Čím vyšší je časová složitost operací ohodnocení jedince T_a a vytvoření nové populace T_p, tím se zvyšuje doba potřebná úspěšnému ukončení. Celkově nás tedy zajímá doba evoluce T_e: [8].

$$T_e = G \cdot (P \cdot T_a + T_p)$$

.

Snížení času potřebného na ohodnocení jedinců (což je nejnáročnější operace) a vytvoření nové populace lze dosáhnout vhodnou volbou metod a optimalizací implementace.

Při hodnocení evolučních algoritmů nám však záleží především na dosažení požadovaného řešení. Zajímá nás např. průměrný počet evaluací nutný k nalezení řešení a kolikrát z n spuštění algoritmus výsledek nalezl. Můžeme počítat i různé statistické údaje jako je medián, minimum, maximum, směrodatná odchylka apod.

Kapitola 3

Kartézské genetické programování

Jedná se o evoluční algoritmus, který kóduje zadaný problém do acyklické orientované grafové struktury. Obecně nachází uplatnění při návrhu kombinačních obvodů [9] [7], v umělé inteligenci a strojovém učení, při řešení matematických problémů, v evolučním umění a mnoha dalších oblastech [6]. Přestože algoritmus nese v názvu "genetické programování", využívá i principu evoluční strategie [1]. Jako genetický operátor vůbec nevyužívá křížení a pracuje s velkým počtem generací a malou populací. CGP zde budeme používat k návrhu obvodů na úrovni hradel a LUT, proto se následující text bude zabývat výhradně touto problematikou.

3.1 Reprezentace obvodu

Kandidátní obvod je popsán jako maticové uspořádání jednotlivých bloků. Velikost pole je $r \times c$, kde r je počet řádků a c počet sloupců matice. Velikost matice se zadává před začátkem výpočtu a v jeho průběhu se již nemění. Každá buňka představuje jeden element (hradlo). Každý element má obecně n vstupů a jeden výstup. Tato práce se zabývá návrhem obvodů na úrovni třívstupových LUT, který je porovnán oproti variantě s dvouvstupovými hradly.

Elementy mohou realizovat libovolnou funkci zadanou pravdivostní tabulkou. Zadán je počet primárních vstupů i a primárních výstupů o navrhovaného obvodu. Nad takto definovanou maticí se hledá vhodné propojení elementů, které realizuje požadovaný obvod popsaný pomocí pravdivostní tabulky. Pro propojování elementů platí, že vstupy elementu jednoho sloupce mohou být připojeny pouze na výstupy elementů v předcházejících sloupcích nebo na primární vstupy. Zabrání se tak zpětné vazbě. Důležitým parametrem řídícím vnitřní konektivitu je L-back, který udává počet bezprostředně předcházejících sloupců, jejichž výstupy je možno připojit na vstup elementu. Tento parametr může nabývat hodnot z intervalu <1, c>. Hodnota 1 znamená, že elementy lze propojovat pouze s bezprostředně předcházejícím sloupcem. Hodnota c říká, že na vstup lze připojit výstupy libovolného ze všech předchozích sloupců. Žádné zpětné vazby nejsou povoleny. Primární výstupy lze připojit na výstup kteréhokoli elementu.

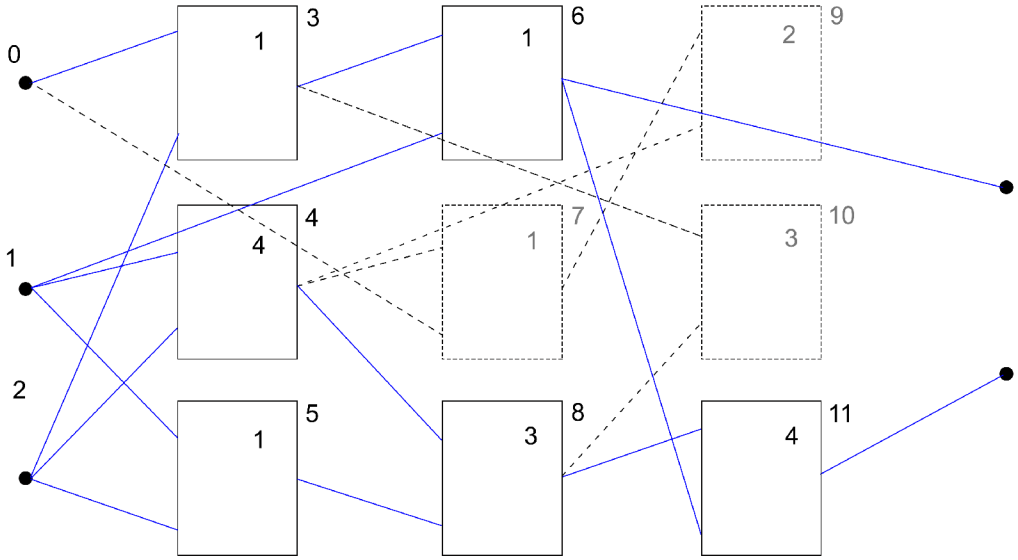
3.2 Kódování chromozomu

Evoluční algoritmy pracují se zakódovanými kandidátními řešeními (chromozomy) problému. Jelikož v CGP máme matici programovatelných bloků, je chromozom zakódován do

celočíselných řetězců délky $r \times c \times (n+1)+o$. Všechny elementy v matici a primární vstupy a výstupy mají přiřazeny unikátní indexy. Nejprve se číslují primární vstupy a poté elementy zleva doprava. Každý blok je reprezentován pomocí $n+1$ hodnot. Prvních n udává indexy výstupů, kam jsou připojeny vstupy bloku, poslední hodnota je rezervována pro reprezentovanou funkci. Poslední část tvoří k-tice s velikostí shodnou s počtem primárních výstupů. Jednotlivé hodnoty v ní představují výstupy elementů nebo primární vstupy, na které jsou primární výstupy připojeny. Chromozom pro CGP s parametry: $r = 3$, $c = 3$, $i = 3$, $o = 2$ může vypadat například následovně.

$$(0,2,1)(1,2,4)(1,2,1)(3,1,1)(4,0,1)(4,5,3)(7,4,2)(3,8,3)(8,6,4)(6,11)$$

Grafická reprezentace obvodu kódovaného tímto chromozomem je na obrázku 3.1.



Obrázek 3.1: Obvod v CGP

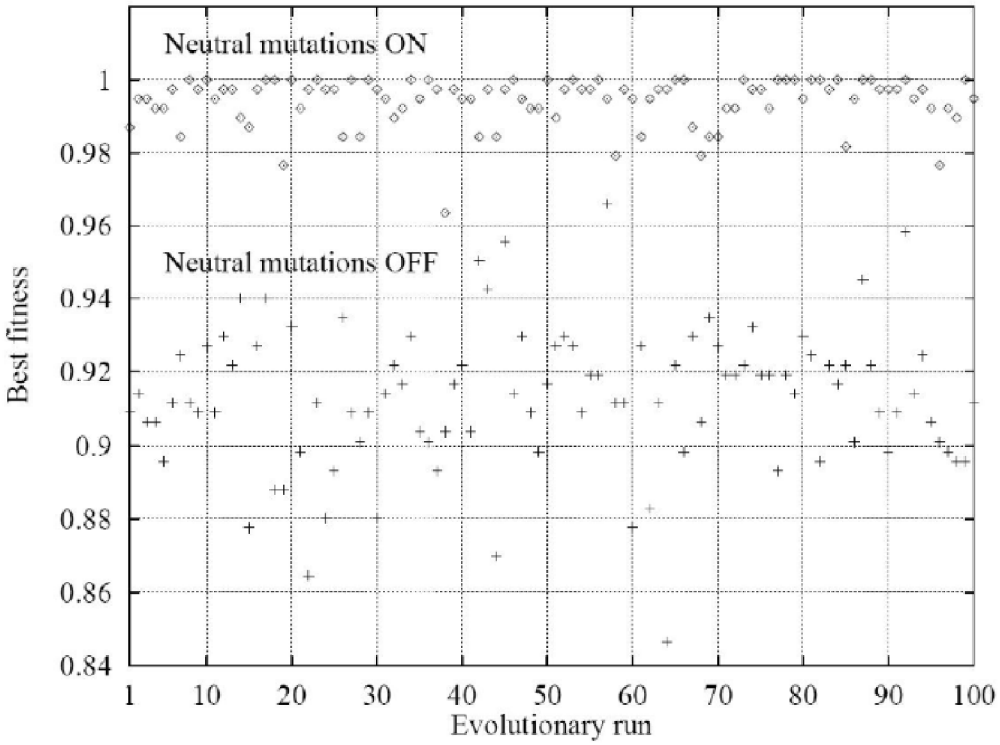
Čísla uvnitř bloků znázorňují realizovanou funkci. Vedle bloků je jejich index (zároveň je tak očíslován jejich výstup). Neaktivní bloky matice, které se nepodílejí na řešení, a jejich propojení jsou znázorněny šedou barvou.

3.3 Evoluční algoritmus

Prohledávací algoritmus CGP je velmi jednoduchý. Je založen na evoluční strategii (ES), konkrétně na variantě $(1+\lambda)$. Hodnota λ se obvykle pohybuje kolem 5. Selekce je postavena na principu elitismu.

V CGP se ke generování potomků nepoužívá křížení, ale pouze operátor mutace, který je definován jako náhodná změna genu. Mutace je řízena parametrem četnosti mutace. Ten udává počet genů, které se budou v jednom procesu mutace měnit. Ne všechny kombinace hodnot v chromozomu jsou přípustné, proto je třeba dbát na zajištění korektních hodnot. Vliv mutace může být pozitivní, negativní, nebo neutrální. V případě pozitivní mutace

dochází ke změně genu v zapojené části reprezentovaného obvodu, která zvyšuje ohodnocení jedince. Negativní vliv vzniká obdobně, ale dochází při něm k degradaci jedince. Významná je neutrální mutace [9] [7]. V tomto případě dochází k mutaci genu, který se přímo nepodílí na řešení (nemá vliv na žádný z primárních výstupů) nebo k takové mutaci aktivních hradel (genů), která vede na logicky identickou funkci. Ohodnocení jedince zůstává stejné a pokud v mutované podobě zůstane v populaci dostatečně dlouhou dobu, může dojít k násobnému počtu neutrálních mutací. Takto může vzniknou zcela nová strukturu, která do té doby v populaci neexistovala. Pokud pak proběhne mutace, která danou část chromozomu aktivuje, ohodnocení jedince se může rapidně zvýšit a velkou částí tak přispět k nalezení celkového řešení. Vliv neutrálních mutací na běh CGP je vidět na obrázku 3.2 Struktury vzniklé neutrálními mutacemi mohou být i destruktivní. V takovém případě se tímto způsobem vzniklý jedinec nedostane do nové populace a nepůsobí negativně na běh celé evoluce. Je také důležité zajistit, aby se v případě výběru z více možností nevolil jako nejlepší stále jeden jedinec.



Obrázek 3.2: Vliv neutrálních mutací na běh CGP, převzato z [9]

Celý algoritmus sestává z několika kroků:

1. Vygenerování $1 + \lambda$ náhodných jedinců pro inicializaci populace
2. Ohodnocení všech jedinců populace pomocí fitness funkce
3. Nalezení nejlépe ohodnoceného jedince (nejvyšší fitness hodnota)
4. Vygenerování λ potomků mutací nalezeného nejlepšího jedince
5. Nalezený nejlepší jedinec společně s jeho λ potomky tvoří novou populaci

6. Pokud není splněna podmínka pro ukončení, pokračuje se krokem 2

3.4 Fitness funkce

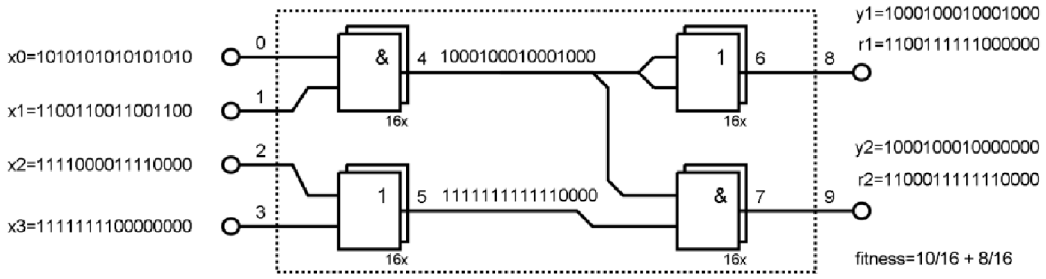
Kritickým krokem algoritmu je ohodnocení jedinců pomocí fitness funkce. Je potřeba najít takovou fitness funkci, která zohlední všechny požadované vlastnosti a nezapříčiní zpomalení evoluce a degradaci populace [4].

Hlavním kritériem hodnocení navrženého obvodu je míra funkčnosti. Výpočet probíhá přivedením všech možných kombinací vstupů na primární vstupy a porovnáním výstupů ohodnocovaného jedince s očekávanými výstupy. Fitness je potom definována jako počet bitů, které kandidátní obvod správně vyprodukoval. Obecně se tak musí provést porovnání 2^i výstupů při i vstupech. Při počtu výstupů o odpovídá maximum fitness funkce $2^i \cdot o$.

Dále můžeme obvody hodnotit na základě minimalizačních požadavků nebo zpoždění. V této práci je použita fitness funkce, která po nalezení plně funkčního řešení zohledňuje počet členů v navrženém obvodu.

3.5 Akcelerovaná simulace

Pro ohodnocení kandidátního obvodu s i vstupy je třeba vygenerovat a získat odezvu pro 2^i vektorů. Vyhodnocovat kombinace vstupů jednotlivě by bylo značně neefektivní, protože by vedlo na 2^i průchodů obvodem. Z tohoto důvodu se používá bitově-paralelní simulace. Namísto jednotlivých bitů se ke každému vstupu přivede vektor vstupů o délce x bitů, který kóduje příslušné bity pravdivostní tabulky, a celé ohodnocení se tak urychlí x-krát. Stupeň paralelizace je omezen architekturou procesoru, přičemž x udává počet bitů FX na dané architektuře. Na obrázku 3.3 je příklad pro $x = 16$.

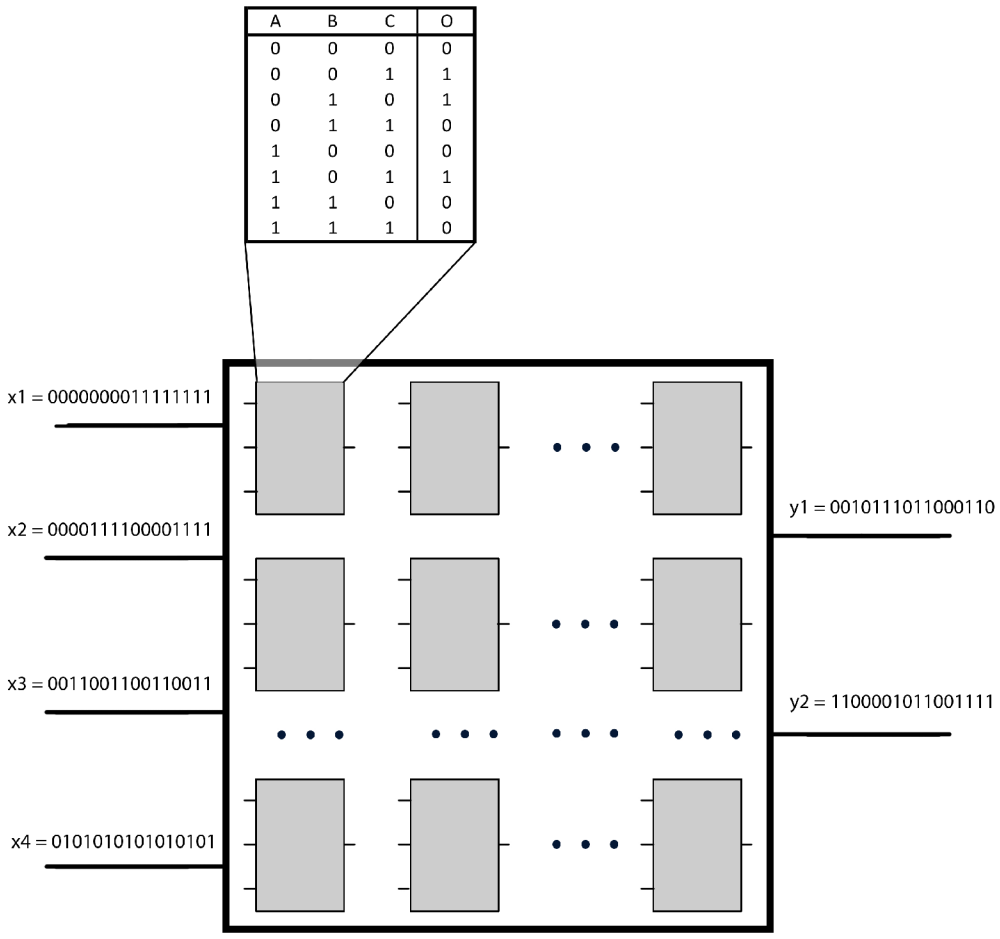


Obrázek 3.3: Paralelní simulace v CGP, převzato z [9]

3.6 CGP s LUT

CGP, které používat třívstupové LUT, funguje obdobně jako jeho standardní varianta. Jednotlivé elementy v mřížce realizují třívstupové logické funkce. Princip této varianty je zobrazen na obrázku 3.4.

Vyvstává zde však otázka, jak akceleraci popsanou v kapitole 3.5 funkcí implementovat. U dvouvstupových hradel se používá jednoduchý výčet všech používaných funkcí a výběr pomocí switch. Funkce jsou realizovány pomocí bitových operací nad datovým typem unsigned int (o x bitech). U vícevstupových LUT vzniká problém exponenciálního nárůstu počtu funkcí v závislosti na počtu vstupů do prvku. Dále je kvůli paralelní simulaci nutné



Obrázek 3.4: CGP s třívstupovými LUT

převést logickou funkci s několika operandy popsanou kódem v LUT na posloupnost elementárních logických funkcí pracujících se dvěma operandy. Pro paralelní simulaci je kritickou částí způsob vyhledání příslušné funkce a realizace bitové operace nad celým vektorem. U vyššího počtu vstupů je neefektivní výčtem popsat všechny funkce (pro $n = 3$ je to 256 funkcí, pro $n = 4$ je to 65536 funkcí atd.) tak, jak je to používáno u dvouvstupových hradel. Nicméně pro účely třívstupových LUT byl postup založený na výčtu vyhodnocen jako nejvhodnější. Pro všech 256 funkcí byla manuálně nalezena jejich realizace pomocí binárních logických operátorů. V příloze jsou uvedeny vytvořené implementace. Rozdíl oproti CGP s dvouvstupovými hradly je zejména v počtu bitových operací, které je nutné vykonat na jednu funkci. Namísto jedné operace pro dvouvstupová hradla může počet operací dosahovat až pěti. Velká část funkcí se třemi vstupy však tohoto maximálního počtu nedosahuje a průměrný počet binárních operací je 3,5. Pro vícevstupové LUT tento počet ještě více narůstá a časová složitost algoritmu značně stoupá.

Pro získání kompletního ohodnocení kandidátního obvodu bude počet binárních operací rovný počtu průchodů $(2^i/x)$ násobený průměrným počtem binárních operací na funkci. Ve srovnání s obvody, které používají dvouvstupová hradla, lze předpokládat, že obvody

tvořené třívstupovými LUT budou mít méně prvků, protože počítají s více vstupy na prvek. Z tohoto důvodu bude postačovat k nalezení řešení menší mřížka CGP.

Kapitola 4

Dynamická změna parametrů CGP

Navržené metody pro dynamickou změnu parametrů se zaměřují na množinu funkcí, kterou mohou jednotlivé elementy realizovat. Při evoluci je běžné, že část množiny funkcí se ve výsledném obvodů vůbec neobjeví, protože je pro něj nevhodná. Díky použití třívstupových LUT je tento jev ještě zřetelnější, protože funkcí je celkem 256. Za běhu algoritmu tak dochází k tomu, že při operaci mutace se v chromozomech nových jedinců mohou opakovaně objevovat funkce, které nepřispívají k nalezení řešení. Může tak dojít ke zpomalení evoluce.

Všechny navržené metody používají ke stanovení nové množiny funkcí pouze nejlepší kandidátní řešení v populaci. Aktivními prvky se v následujícícm textu rozumí elementy mřížky, které jsou součástí fenotypu.

4.1 Body redukce množiny funkcí

Aby bylo možné za běhu algoritmu množinu funkcí redukovat, je potřeba stanovit body jeho běhu, kde k tomu bude docházet. Při redukci množiny funkcí by se měly dát identifikovat funkce, které přispějí k nalezení řešení a budou výsledný obvod realizovat. S ohledem na tuto skutečnost byly navrženy dvě možnosti, kdy bude k eliminaci funkcí docházet. První je založena na hodnotě fitness funkce, druhá na počtu generací.

4.1.1 Redukce na základě fitness funkce

Protože nevhodné funkce se přestávají v nejlepších jedincích objevovat až ve chvíli, kdy se přibližují požadovanému řešení, nemělo by smysl provádět eliminaci na začátku evoluce. Čím kvalitnější řešení v danou dobu k dispozici, tím kvalitnější bude i redukce množiny funkcí. Zároveň však chceme, aby se s redukovanou množinou funkcí pracovalo co možná nejdéle a tím došlo k většímu zrychlení algoritmu. Při opakovaném spuštění standardního CGP bylo vypozorováno, že hodnota fitness roste tím pomaleji, čím je kvalitnější kandidátní řešení. Typicky v rozmezí 80 - 90% maxima hodnoty fitness nastává zlom, kdy je nalézání kvalitnějších kandidátních řešení již velmi pozvolné. Nejdelší dobu tak algoritmus prohledává kandidátní řešení, která by mohla vést ke zlepšení fitness o zbývajících 10-20% její maximální hodnoty. Z tohoto důvodu byl pro redukci zvolen bod evoluce, kdy fitness dosáhne 85% maximální hodnoty.

4.1.2 Redukce na základe počtu generací

Tato metoda provádí redukci množiny funkcí několikrát v průběhu evoluce. Děje se tak periodicky vždy po určitém počtu generací. Obecně by bylo možno počet redukcí provést n-krát, což znamená zavedení nového parametru. Body, kde dochází k redukci, byly zvoleny pomocí rozdělení konečného počtu generací na stejně velké intervaly. V případě konečného počtu generací 30 mil. a jedné redukce by tento proces nastal v polovině (po 15 mil. generací), při dvou redukcích po třetinách (každých 10 mil. generací) atd. Pro účely experimentů v tomto textu byla redukce prováděna dvakrát v průběhu celé evoluce.

4.2 Způsob redukce

Samotnou redukci množiny funkcí lze provádět několika různými způsoby. Dále uvedené postupy jsou použity v několika variantách společně s výše uvedenými možnostmi bodů, kdy k redukci dochází.

4.2.1 Redukce podle relativního zastoupení

Jednou z možností, jak identifikovat užitečné funkce pro řešenou úlohu, je zachovat všechny funkce, které se v průběžném řešení objevují dostatečně často vzhledem k předem určené hranici. Abychom mohli určit, které funkce se zachovají v množině, byl stanoven následující obecný práh:

$$p \cdot B/F$$

kde B je počet bloků, které při určování uvažujeme a F je počet doposud používaných funkcí. Parametr p je volen podle zkušeností s danou variantou redukce množiny funkcí. Pro různé další experimenty by bylo možno práh upravovat nebo stanovit zcela odlišný.

4.2.2 Půlení množiny

V této variantě dochází k redukci obecně n-krát za celou evoluci. Je ale zřejmě, že počet těchto redukcí má svá omezení plynoucí z velikosti počáteční množiny funkcí. Na rozdíl od předchozí metody je pevně daný počet funkcí, které po redukci zůstanou zachovány. Do redukované množiny budou vždy vybírány nejpoužívanější funkce v průběžném řešení. Pokud by nastala situace, kdy je stejně zastoupeno více používaných funkcí, vybere se potřebný počet náhodně. V případě, že není potřebný počet funkcí zastoupen v blocích, množina se náhodně doplní o zbylý počet funkcí. V našem případě, kdy jsou používány třívstupové LUT, bude velikost množiny po první redukci 128, po druhé 64 atd. Na rozdíl od předchozí varianty zde nehrozí, že redukce funkcí bude mít pouze minimální efekt nebo naopak k ní dojde příliš drasticky.

Použití této varianty nemá příliš velký význam v kombinaci s redukcí podle fitness funkce, protože by došlo ke zmenšení množiny na 128 funkcí, což je stále příliš vysoký počet. Z tohoto důvod nebyla taková kombinace variant v experimentech použita. V budoucnu by se však při redukci podle hodnoty fitness dalo experimentovat např. se zachováním pouze čtvrtiny nebo osminy funkcí.

4.2.3 Redukce podle aktivních prvků

Vliv na hodnotu fitness jedince mají pouze aktivní prvky. Nabízí se tak možnost uvažovat při redukci množiny funkcí právě pouze funkce aktivních prvků. Výběr funkcí nebude negativně ovlivněn možnými destruktivně působícími úseky genotypu.

4.2.4 Redukce podle všech prvků

Jak již bylo popsáno ve třetí kapitole, neaktivní části genotypu mohou díky neutrálním mutacím nést informaci s pozdějším výrazně pozitivním vlivem na ohodnocení jedince. Aby se tento vliv mohl projevit i při redukci množiny funkcí, byla navržena metoda, která uvažuje celou mřížku prvků, nikoli pouze aktivní prvky.

4.2.5 Implementace

Implementace všech metod je založena na použití dvou polí čísel typu integer. První z polí slouží pro počítání výskytů funkcí v poli CGP. Indexy v tomto poli odpovídají číslům funkcí (tak, jak jsou v pravdivostních tabulkách funkcí) a hodnoty, které jsou na těchto indexech uložené, odpovídají právě počtu výskytů. Druhé pole slouží pro uchování funkcí, které jsou použity v množině funkcí. Indexy v něm slouží jako pořadí funkcí v právě používané množině. Jednotlivé položky pak značí číslo funkce, aby bylo možno příslušnou funkci nalézt ve výčtu funkcí při ohodnocování.

Kapitola 5

Experimenty

Bylo provedeno několik sad experimentů, které porovnávají běžně používanou variantu CGP s dvouvstupovými hradly, CGP s třívstupovými LUT a plnou množinou funkcí a CGP s třívstupovými LUT a dynamickou redukcí množiny funkcí. Výsledky jsou porovnány taktéž s CGP, kde byla množina přizpůsobena pro jednotlivé navrhované obvody. Takto navržená množina umožnila algoritmu nalézt plně funkční řešení rychleji než s plnou množinou funkcí. V této kapitole jsou používány názvy základních logických funkcí s číslem 3 ve svém názvu, které značí jejich třívstupovou variantu. Např. AND3 je třívstupový AND. XOR3 je v experimentech realizován jako $a \oplus b \oplus c$, ačkoli se z logického hlediska nabízí ji realizovat způsobem, kdy by odpovídala hodnotě 1 ve chvíli, kdy je právě jeden ze vstupů v log. 1. Tento přístup byl zvolen z důvodu, že je při návrhu obvodů běžně používán.

Aby bylo možné lépe porovnat chování jednotlivých variant, byly vybrány různé rozměry mřížky, pro které byl algoritmus spouštěn. Parametr L-back byl vždy nastaven na maximální hodnotu konektivity, protože toto nastavení usnadňuje celou evoluci. Jako ukončující podmínka evoluce sloužil počet generací stanovený na 20 000 000. Velikost populace byla ve všech případech rovna pěti. Dostáváme tak tři různá nastavení parametrů, která jsou v tabulce 5.

	nastavení 1	nastavení 2	nastavení 3
počet řádků	8	10	12
počet sloupců	7	7	7
L-back	7	7	7
počet generací	20 000 000	20 000 000	20 000 000
velikost populace	5	5	5
počet mutovsaných genů	1	1	1

Tabulka 5.1: Různá nastavení experimentů

Varianty algoritmu, které počítají s prahem, vůči kterému je porovnáváno relativní zastoupení funkcí, jsou nastaveny následovně.

$$5 \cdot B/F$$

pro případy, kdy se uvažují pouze aktivní prvky kandidátního obvodu, a

$$0.5 \cdot B/F$$

v případech, kdy se množina funkcí sestavuje z celé mřížky. Hodnoty $p = 5$, resp. $p = 0.5$ byly stanoveny experimentálně na základě opakovaného spouštění algoritmu a pozorování

jeho průběhu s různými hodnotami parametru p. Pro redukci, která uvažuje pouze aktivní prvky, je tato hodnota vyšší z důvodu, že zapojená část mřížky je typicky pouze menší částí mřížky celé. U variant uvažujících funkce v celé mřížce je potřeba dát prostor ovlivnit evoluci většímu množství funkcí, aby nedošlo k vyloučení funkcí použitých ve fenotypu.

Pro každou variantu algoritmu a obvodu bylo provedeno padesát běhů. Jak již bylo popsáno v předchozí kapitole, v případě redukování množiny podle hodnoty fitness je stanovená hranice 85% její maximální hodnoty. Při experimentech byly navrhovány tři různé obvody:

- 9b parita,
- sčítačka 4b + 3b a
- násobička 4b × 3b.

Sady experimentů jsou rozděleny podle obvodů, které v nich byly navrhovány. Jejich výsledky jsou vzájemně porovnány a je popsán vliv velikosti mřížky na výsledná řešení. Primárním ukazatelem účinnosti metod je počet generací (obecně bychom mluvili o počtu evaluací, ale velikost populace byla ve všech případech nastavena stejně) a procentuální úspěšnost algoritmu v nalezení řešení ze všech padesáti běhů. Druhým ukazatelem je počet prvků, ze kterého navržené obvody sestávají. V úvahu je brán počet prvků prvního řešení a počet prvků nejlepšího nalezeného řešení před ukončením evoluce.

Počet generací je prezentován pomocí grafů, které zobrazují minimum, 1. kvartil, medián, 3. kvartil a maximum ze zaznamenaných hodnot. V případech, kdy byly u některých metod hodnoty příliš vysoké, nejsou tyto hodnoty v grafech již zobrazeny, aby byla nejdůležitější část výsledků čitelná. Písmena A, B, C atd. na ose y značí variantu algoritmu.

A) Standardní varianta CGP s dvouvstupovými hradly

B) CGP s třívstupovými LUT a plnou množinou funkcí

C) CGP s množinou redukovanou přímo pro navrhovaný obvod

D) CGP s redukcí na základě fitness podle relativního zastoupení funkcí v aktivních prvcích

E) CGP s redukcí na základě počtu generací podle relativního zastoupení funkcí v aktivních prvcích

F) CGP s redukcí na základě fitness podle relativního zastoupení funkcí ve všech prvcích

G) CGP s redukcí na základě počtu generací podle relativního zastoupení funkcí ve všech prvcích

H) CGP s půlením množiny na základě počtu generací a podle zastoupení funkcí v aktivních prvcích

I) CGP s půlením množiny na základě počtu generací a podle zastoupení funkcí ve všech prvcích

Kvalita nalezených řešení a úspěšnost algoritmu jsou zobrazeny v tabulkách. Varianty CGP jsou pro přehlednost řazeny stejně jako u grafů a jsou pojmenovány. Metody pro dynamickou redukci množiny funkcí jsou v tabulce pojmenovány heslovitě jako "způsob redukce, uvažované prvky, bod redukce".

5.1 Parita 9b

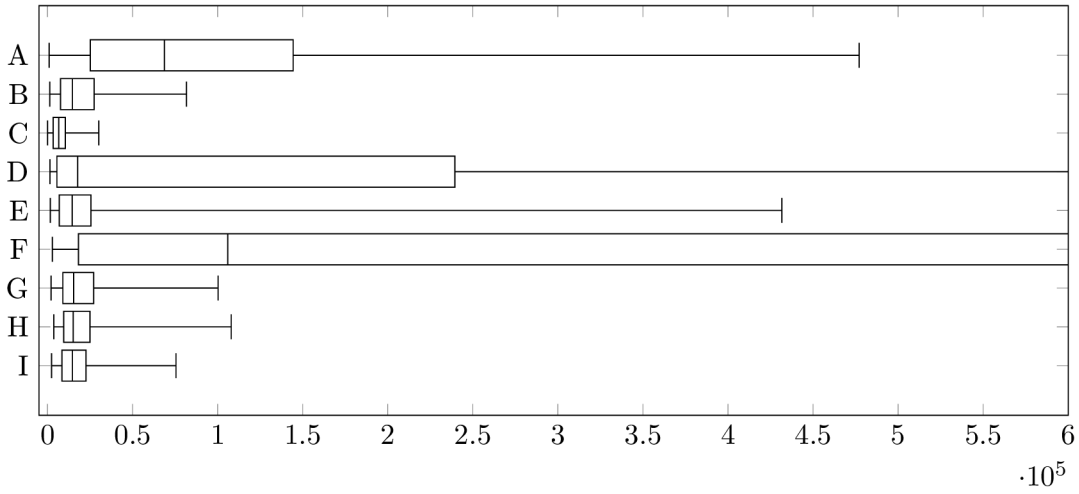
Nejjednodušším z navrhovaných obvodů je sudá 9b parita. Typicky tento obvod tvoří pouze funkce XOR, která se použije postupně na všechny bity. Na základě této znalosti byla navržena množina funkcí pro CGP s třívstupovými LUT. Tvoří ji pouze dvě funce: XOR a XOR3. Je tedy zřejmé, že v tomto případě se hledá spíše propojení prvků než funkce, které by se měly použít. Grafy 5.1, 5.2 a 5.3 potvrzují, že takto navrženou množinou došlo k výraznému urychlení oproti variantě CGP, která používá plnou množinu třívstupových funkcí. V těchto grafech si také lze všimnout, že při použití třívstupových LUT v CGP několikanásobně urychlilo hledání řešení oproti standardní variantě dvouvstupovými hradly. Podíváme-li se na výsledky metod, které se snažily přiblížit rychlosti CGP s předem redukovanou množinou, zjistíme, že metody využívající hodnotu fitness k určení bodu redukce (označeny D a F) jsou značně neefektivní. Celkové rozložení výsledků je horší než při použití plné množiny. Zřejmě je tento problém zapříčiněn tím, že kandidátní obvod při svých 85% funkčnosti nemusí obsahovat funkce, které řešení usnadní. Algoritmus je pak nucen řešení hledat mnohem složitějším způsobem přes funkce jiné.

Porovnáme-li varianty D a F i vzájemně mezi sebou, je vidět velký nárust počtu generací, který potřebuje k nalezení řešení varianta uvažující všechny prvky mřížky. V tomto případě byl mylný předpoklad, že by k řešení přispěly neutrální mutace přinášející pozitivní změny do genotypu. Nejsou dostatečně četné na to, aby dobu návrhu snížili. Ve výsledku tedy převažují destruktivní dopady náhodných genů, které se neprojevují v hodnocení fenotypu. Toto je zejména viditelné v tabulce 5.2, kde varianta F dosáhla úspěšnosti pouze 90%, zatímco ve všech ostatních případech bylo řešení nalezeno vždy. Na základě výsledků CGP s půlením množiny podle všech prvků by bylo možné namítnou, že tato varianta poskytuje srovnatelné výsledky s CGP s plnou množinou funkcí a problém tak nebude způsoben uvažovanými prvky. Je však nutné si uvědomit, jak často se půlení množiny provádí. Experimenty byly spuštěny na 20 000 000 generací a půlení se provádělo celkem dvakrát za evoluci. Dostáváme tak interval 6 666 6667 generací. Protože je návrh 9b parity snadná úloha, došlo k nalezení řešení ještě před samotnou reduckí množiny funkcí. Redukce tak měla vliv na minimalizaci obvodu, ale nehrála významnou roli při hledání funkčního řešení. Návrh tohoto obvodu vyžaduje řádově desetitsíce až statisíce generací.

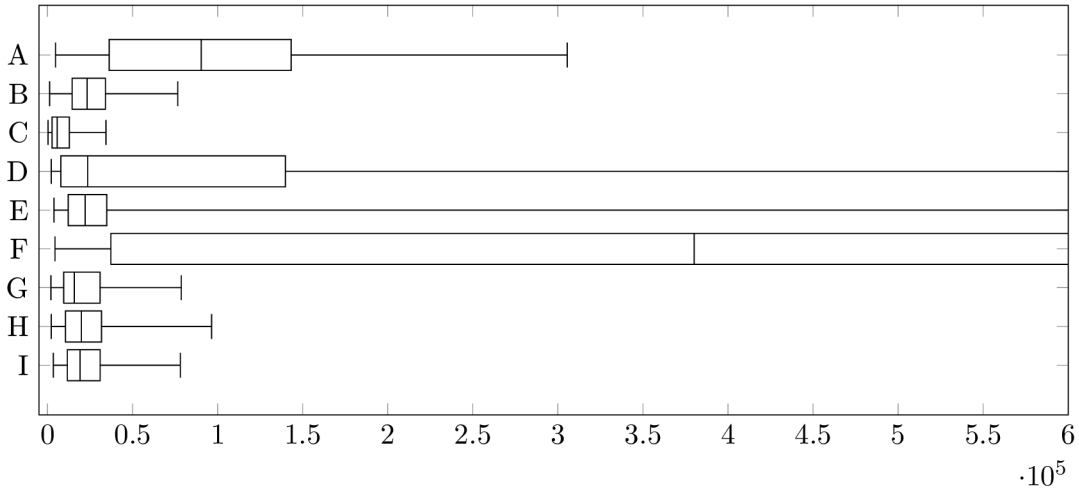
Protože i zbytek metod používá redukování množiny v intervalech daných počtem generací, lze konstatovat, že s daným nastavením nelze na návrhu parity sledovat chování variant v oblasti rychlosti nalezení prvního funkčního řešení. V příslušných grafech je vidět, že všechny tyto varianty poskytly výsledky v podobném rozsahu. Můžeme však porovnávat jejich účinnost v oblasti minimalizace a sledovat chování v závislosti na velikosti mřížky.

Při pohledu na tabulky 5.2, 5.3 a 5.4 vidíme, že standardní CGP používající hradla velmi často našla minimální řešení, které sestává z osmi hradel XOR. Při použití třívstupových LUT má odpovídající minimalizovaný obvod 4 prvky realizující funkcí XOR3. Srovnání, průměrného počtu prvků v řešeních tak zjistíme, že pouze CGP, které používalo množinu přizůsobenou návrhu parity, předčilo výsledky standardní varianty. Dobré výsledky poskytla ještě varinata používající redukci podle relativního zastoupení funkcí v aktivních prvcích na základě fitness a varianta používající půlení množiny podle všech prvků.V obou případech je průměrný počet prvků v obvodu nižší než při použití CGP s plnou množinou funkcí.

Zajímavostí při použití plné množiny funkcí je, že algoritmus často nacházel minimální řešení, která vůbec nepoužívala funkci XOR nebo XOR3. Takto naržené obvody používaly také pouze 4 prvky, i když funkce byly komplexnější. Toto řešení se objevovalo i u variant s dynamickou redukcí množiny.



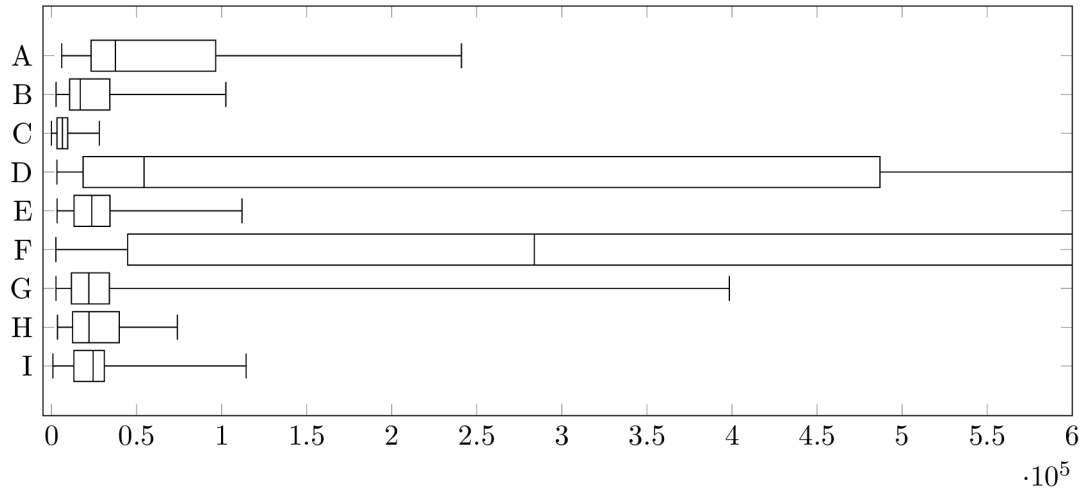
Obrázek 5.1: Počet generací potřebný k nalezení 9b parity v poli 8×7



Obrázek 5.2: Počet generací potřebný k nalezení 9b parity v poli 10×7

Porovnáním výsledků u jednotlivých velikostí mřížky zjistíme, že velikost řešení má tendenci růst společně s mřížkou. K tomuto jevu dochází jak u standardního CGP, tak u CGP používajícího třívstupové LUT. V tomto ohledu tedy jednotlivé metody dynamické redukce množiny funkcí nepřinášejí při návrhu parity změny.

Celkově můžeme říci, že naprosto nevhodná je pro návrh parity metoda používající redukci podle relativního zastupení funkcí v aktivních prvních na základě hodnoty fitness. Naopak zlepšení při minimalizaci nalezeného řešení přináší varianta používající redukci podle relativního zastoupení ve všech prvcích na základě fitness. Velmi podobné výsledky jako u CGP s plnou množinou pak má půlení množiny podle všech prvků. Bohužel se pomocí žádné z metod nepovedlo přiblížit výsledkům poskytovaným CGP, které používá množinu sestavenou přímo pro návrh vybraného obvodu.



Obrázek 5.3: Počet generací potřebný k nalezení 9b parity v poli 12×7

verze CGP	**Úspěšnost**	**Počet uzlů prvního řešení**			**Počet uzlů**		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	100	8	16	11.14	8	10	8.12
LUT3 s plnou množinou	100	6	24	11.74	4	9	6.26
LUT3 s redukovanou množinou	100	5	18	10.28	4	4	4
LUT3, relativní, aktivní, fitness	100	6	24	14	4	12	6.96
LUT3, relativní, aktivní, generace	100	7	23	12.98	4	11	5.26
LUT3, relativní, vše, fitness	90	6	29	17.36	4	19	10.47
LUT3, relativní, vše, generace	100	5	27	14.24	4	11	6.4
LUT3, půlení, aktivní, generace	100	5	25	13.06	4	9	6.26
LUT3, půlení, vše, generace	100	5	24	12.72	4	8	5.88

Tabulka 5.2: Kvalita řešení 9b parity v poli 8×7

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	100	8	18	11.86	8	10	8.16
LUT3 s plnou množinou	100	7	28	13.44	4	10	6.54
LUT3 s redukovanou množinou	100	5	24	10.98	4	5	4.06
LUT3, relativní, aktivní, fitness	98	5	26	14.76	4	19	8.73
LUT3, relativní, aktivní, generace	100	7	30	14.72	4	11	5.84
LUT3, relativní, vše, fitness	98	7	28	15.57	4	19	9.57
LUT3, relativní, vše, generace	100	5	27	15.16	4	9	6.5
LUT3, půlení, aktivní, generace	100	8	27	15.84	4	10	6.56
LUT3, půlení, vše, generace	100	4	29	13.8	4	12	6.64

Tabulka 5.3: Kvalita řešení 9b parity v poli 10×7

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	100	8	20	11.72	8	11	8.34
LUT3 s plnou množinou	100	5	29	14.46	4	12	6.68
LUT3 s redukovanou množinou	100	4	22	11.46	4	6	4.12
LUT3, relativní, aktivní, fitness	100	5	37	17.38	4	20	8.32
LUT3, relativní, aktivní, generace	100	7	29	15.88	4	9	6.08
LUT3, relativní, vše, fitness	100	8	39	18.68	4	29	10.12
LUT3, relativní, vše, generace	100	5	39	14.78	4	29	6.96
LUT3, půlení, aktivní, generace	100	5	31	16.12	4	11	6.64
LUT3, půlení, vše, generace	100	7	30	13.7	4	11	6.88

Tabulka 5.4: Kvalita řešení 9b parity v poli 12×7

5.2 Sčítačka 4b + 4b

Sčítačka 4b + 4b je druhým obvodem použitým pro experimentální účely ohodnocení účinnosti navržených metod. Ze zkušeností s návrhem sčítaček je známo, že řešení obvodu je tvoření celkem třemi funkcemi: XOR, AND, OR. [5] Tato znalost posloužila jako základ pro návrh množiny funkcí se třemi vstupy, která umožní CGP snažší nalezení plně funkčního řešení. Narozdíl od návrhu množiny pro paritu však hledání funkcí pro sčítačku není tak intuitivní. Větší počet dvouvstupových funkcí znamená značný počet možných třívstupových funkcí, které AND, XOR a OR kombinují. Návrh této množiny proto probíhal formou experimentů s jednotlivými množinami funkcí. Znalosti o návrhu sčítaček na úrovni dvouvstupových hradel tak byly doplněny o znalosti získané experimenty s množinou rozšířenou o různé kombinace funkcí. Právě různé možné kombinace třívstupových funkcí hledání množiny značně komplikují. Návrh nemusí jednoznačně usnadňovat jednotlivé funkce. Účinnosti algoritmu může být ovlivněna skupinami funkcí, které je obtížné v celkovém počtu 256 funkcí identifikovat. Výsledná množina pro návrh sčítačky 4b + 4b na úrovni třívstupových LUT je následující: {identita, AND, OR, XOR, AND3, OR3, XOR3, (a $\wedge$ b) $\vee$ c, (a $\oplus$ b) $\vee$ c}. V grafech 5.4, 5.5 a 5.6 je vidět, že tato množina přináší urychlení návrhu na počet generací. Pokud se však podíváme do tabulek 5.5, 5.6 a 5.7, zjistíme, že se algoritmus má s touto množinou větší problémy obvod minimalizovat. Důvodem je již popsané problematické hledání třívstupových funkcí, které urychlí hledání plně funkčního řešení a zároveň umožní snažší minimalizaci. V plné množině funkcí jsou zřejmě takové funkce, které kombinují základní dvouvstupovou množinu a zároveň realizují větší část obvodu než manuálně navržená třívstupová množina. Cílem dynamické redukce množiny funkcí bylo odstranit také tento nedostatek.

Při pohledu na grafy výsledků návrhu sčítačky nás zaujme varianta F, která poskytuje jednoznačně nejvyšší rozsah hodnot. Je tak z hlediska rychlosti návrhu horší než standardní varinta CGP s dvouvstupovými hradly. Důvody tohoto chování byly vysvětleny již v textu hodnotícím výsledky návrhu 9b parity. Druhý největší rozsah hodnot má varianta D. Medián hodnot se pohybuje ve stejné oblasti jako u standardního CGP. Vzhledem k faktu, že ohodnocení kandidátního obvodu s třívstupovými LUT je náročnější na počet bitových operací než ohodnocení obvodu s dvouvstupovými hradly, nepřináší tato metoda žádné zlepší, spíše naopak. Zejména je tento výsledek zřejmý v porovnání varianty D s CGP používajícím plnou množinu a nejznatelnější při porovnání s CGP s množinou přizpůsobenou pro návrh sčítačky.

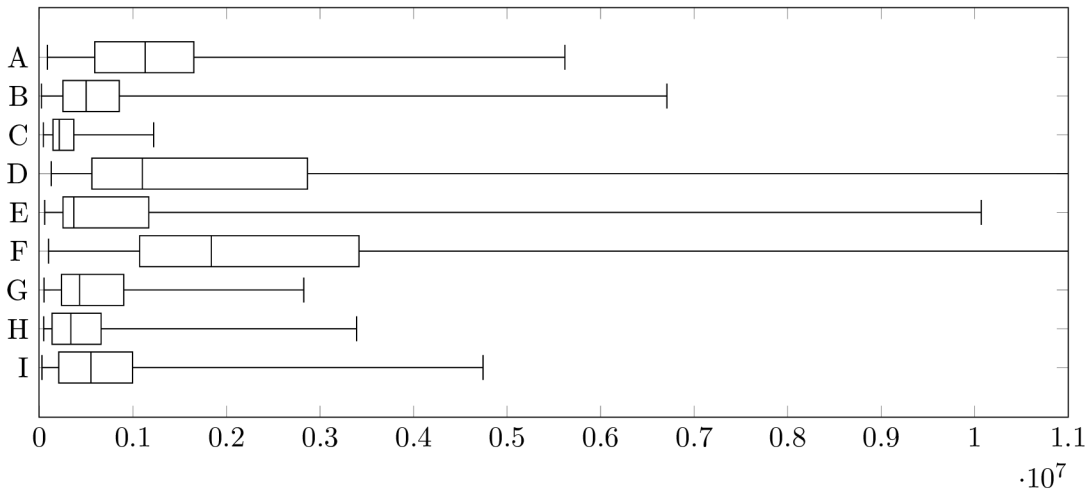
Metody G, H a I poskytují přibližně stejné výsledky. Rozdíly najdeme při porovnání jednotlivých velikostí mřížek, kdy je každá z těchto metod částečně lepší nebo horší u jiné velikosti pole. Pokud bychom provedli sadu experimentů s větším počtem běhů, dalo by se očekávat, že výsledky poskytované metodami ustálí v přibližně stejných rozsazích. Tento rozsah odpovídá výsledkům, které byly získány ze sady experimentů s CGP s plnou množinou funkcí. První řešení sčítačky byla naleze řádově ve statisících generací. Tyto metody používají právě počet generací k určení bodu, kdy dochází k redukci množiny. Stejně jako v případě parity tak mají metody vliv pouze na minimalizaci řešení. Když se zaměříme na tabulky s úspěšností algoritmu a počtem prvků řešení, tak zjistíme, že všechny tyto varianty poskytují funkční řešení ve 100 % případů. Jejich nevýhodou je však počet prvků minimalizovaného řešení. Obecně jsou řešení z tohoto hlediska horší než při použití plné množiny funkcí.

Zajímavá je pro nás metoda E, která ve všech případech poskytuje průměrně lepší hodnoty mediánu i obou kvartilů. Větší počet spuštění algoritmu by výsledky ujasnil. Vliv

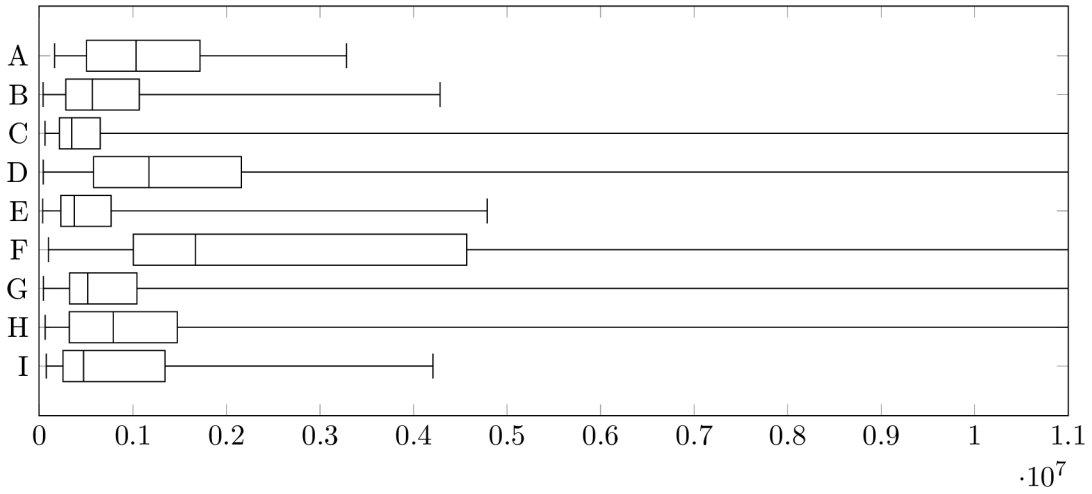
na minimalizaci je negativní stejně jako u metod G, H a I.

Při porovnání chování metod v závisloti na velikosti mřížky zjistíme, že s rostoucí velikostí mřížky se zvyšuje i počet generací potřebný k nalezení plně funkčního řešení. Tento trend platí pro všechny varianty algoritmu. Použití třívtupových LUT nebo dynamická redukce množiny nemají na toto chování vliv. Stejné výsledky poskytuje porovnání počtu prvků v jednotlivých řešeních. Čím větší mřížka, tím méně kvalitní řešení.

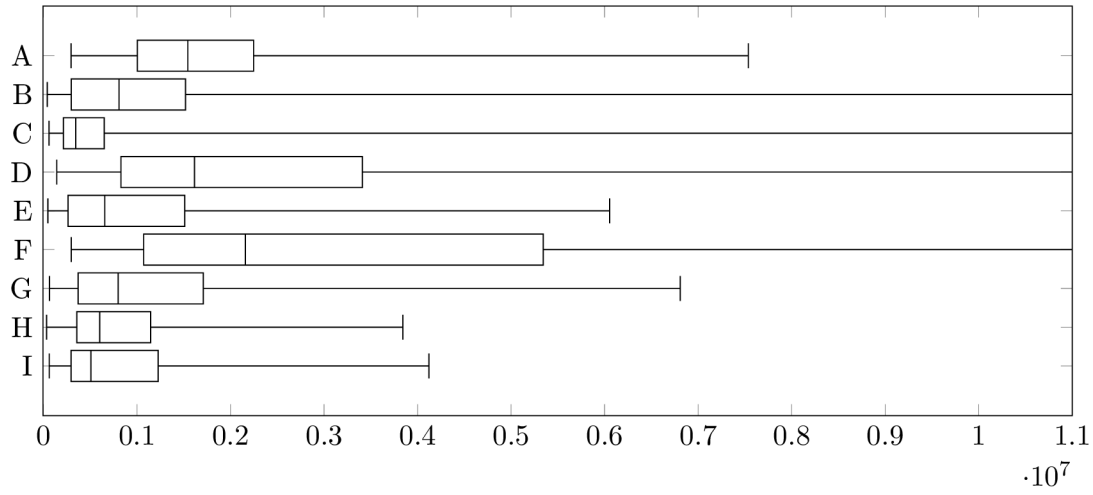
Celkově lze konstatovat, že metody založené na hodnotě fitness nejlepšího kandidátního řešení selhávají. Metody založené na počtu generací neumožňují obvodu minimalizaci jako plná množina. Překvapivě negativní vliv má i půlení množiny, které je z navržených metod nejšetrnější a množina u něj zůstává po redukci největší.



Obrázek 5.4: Počet generací potřebný k nalezení sčítačky 4b + 4b v poli 8×7



Obrázek 5.5: Počet generací potřebný k nalezení sčítačky 4b + 4b v poli 10×7



Obrázek 5.6: Počet generací potřebný k nalezení sčítačky 4b + 4b v poli 12×7

verze CGP	**Úspěšnost**	**Počet uzlů prvního řešení**			**Počet uzlů**		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	98	20	33	25.94	17	26	20.29
LUT3 s plnou množinou	100	14	33	23.68	10	20	13.56
LUT3 s redukovanou množinou	100	18	37	25.9	13	25	16.74
LUT3, relativní, aktivní, fitness	100	17	39	26.88	11	38	18.34
LUT3, relativní, aktivní, generace	100	16	36	25.3	10	24	14.72
LUT3, relativní, vše, fitness	94	23	40	30.51	14	32	21.89
LUT3, relativní, vše, generace	100	15	36	24.6	11	24	14.34
LUT3, půlení, aktivní, generace	100	13	36	24.62	9	30	14.14
LUT3, půlení, vše, generace	100	16	34	24.64	10	19	14.16

Tabulka 5.5: Kvalita řešení sčítačky 4b + 4b v poli 8×7

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	100	21	37	27.7	17	32	20.42
LUT3 s plnou množinou	100	17	43	26.5	10	18	13.92
LUT3 s redukovanou množinou	100	20	38	28.3	13	24	17.32
LUT3, relativní, aktivní, fitness	98	21	44	29.92	12	33	18.49
LUT3, relativní, aktivní, generace	98	19	45	26.76	9	19	14.06
LUT3, relativní, vše, fitness	96	21	49	32.54	11	37	22.31
LUT3, relativní, vše, generace	100	16	39	26.54	9	21	13.74
LUT3, půlení, aktivní, generace	100	20	45	26.74	10	41	15.48
LUT3, půlení, vše, generace	100	15	38	27.12	10	20	14.5

Tabulka 5.6: Kvalita řešení sčítačky 4b + 4b v poli 10×7

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	100	19	37	28.04	17	29	20.6
LUT3 s plnou množinou	100	20	42	29.48	10	24	15.32
LUT3 s redukovanou množinou	100	22	39	29.54	13	24	17.62
LUT3, relativní, aktivní, fitness	96	19	45	30.23	12	33	18.38
LUT3, relativní, aktivní, generace	100	15	41	28.9	11	21	14.98
LUT3, relativní, vše, fitness	98	22	48	34.88	14	37	23.96
LUT3, relativní, vše, generace	100	15	43	28.72	10	35	15.98
LUT3, půlení, aktivní, generace	100	16	38	27.46	10	24	15.06
LUT3, půlení, vše, generace	100	21	47	28.94	11	33	16.02

Tabulka 5.7: Kvalita řešení sčítačky 4b + 4b v poli 12×7

5.3 Násobička 4b × 3b

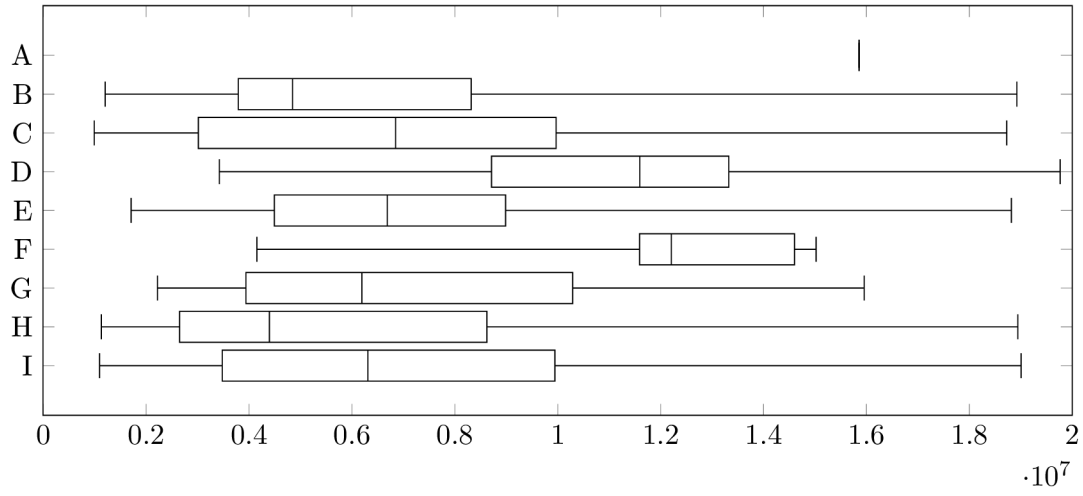
Nejsložitějším z obvodů použitých pro experimenty je násobička 4b × 3b. Jedná se o obvod, který je typicky tvořen funkcemi AND, OR a XOR. Pro návrh pomocí evoluce je však velmi důležitá také identita. Mnohé experimenty již dokázaly, že tato funkce významně urychluje celou evoluci [9], i když se ve výsledném navrženém obvodu neobjeví. Návrh redukované množiny třívstupových funkcí na základě těchto znalostí je obtížný stejně jako v případě sčítačky. Po řadě experimentů s různými množinami funkcí byla vybrána tato: {identita, AND, OR, XOR, AND3, OR3, XOR3, (a ∧ b) ∨ c, (a ∧ b) ⊕ c, (a ∨ b) ∧ c, (a ∨ b) ⊕ c, (a ⊕ b) ∧ c, (a ⊕ b) c}.

Výsledky sad experimentů s tímto obvodem jsou pro nás nejzajímavější. Když se podíváme na tabulky 5.8, 5.9 a 5.10, všimneme si, že standardní varianta CGP má problémy řešení této úlohy nalézt. Zejména viditelný je tento problém u velikosti mřížky 8×7, kde bylo nalezeno z padesáti běhů pouze jediné řešení. S rostoucí velikostí mřížky se úspěšnost algoritmu sice zvyšovala, ale stále nebyla úspěšná ani ve třetině behů. Úspěšnost celého návrhu pro nás tak při návrhu násobičky bude hlavním ukazatelem vhodnosti metod. Nejprve se podívejme na úspěšnost algoritmu používajícího předem redukovanou množinu a porovnejme ho s úspěšností evoluce používající plnou množinu. Při experimentech s mřížkou 8×7 a 12×7 jsou výsledky lepší s redukovanou množinou. U mřížky 10×7 vyšlo lépe použité plné množiny. V tomto případě se však jedná pouze o rozdíl jediného běhu a lze tak konstatovat, že redukovaná množina umožňuje CGP nalézt řešení s větší úspěšností. Výjma mřížky 8×7 také redukovaná množina poskytla výsledky během menšího počtu generací.

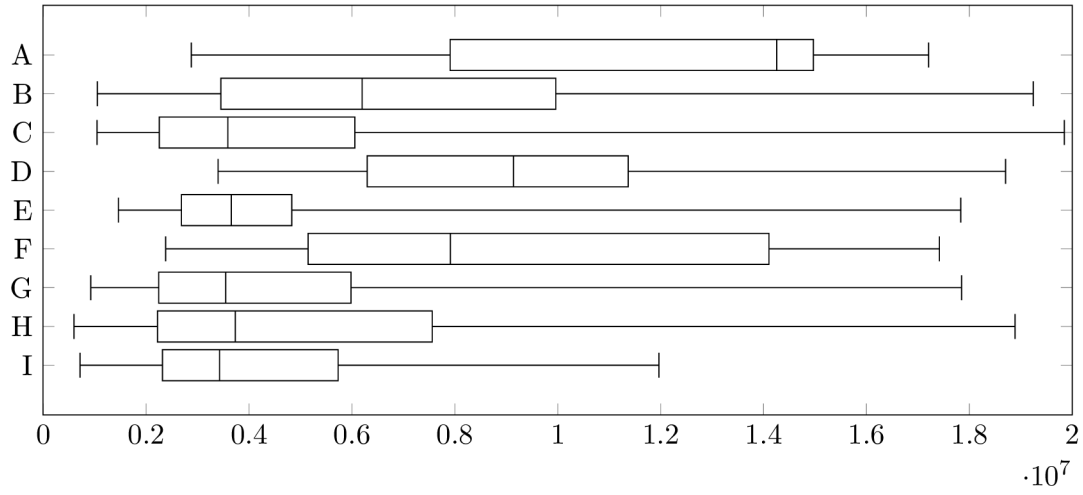
Z metod s dynamickou redukcí množiny se z hlediska rychlosti návrhu i úspěšnosti neosvědčily metody D a F, které jsou založeny na hodnoté fitness nejlepšího kandidátního řešení. V grafech 5.7, 5.8 a 5.9 mají ostatní metody různé výsledky při různých velikostech mřížky. Když porovnáme výsledky pro velikost pole 8x×7 (graf 5.7 a tabulka 5.8), Vyjde nám nejlépe metoda H, která používá půlení množiny podle aktivních prvků na základě počtu generací. V porovnání s algoritmem, který používá plnou množinu, má nižší minimum, 1. kvartil, medián a maximum. Zbylé metody při tomto nastevní parametrů poskytují obecně horší výsledky. Když vezmeme v úvahu ještě úspěšnost algoritmů, vychází G nejhůře. Naopak nejlepší je I a 6 % za ní H. Zajímavá je také informace, že předem zredukovaná množina v této mřížce nepřinesla zlepšení v počtu generací. Umožnila však celkově vyšší úspěšnost. Můžeme říci, že díky použití třívstupových LUT je CGP schopno nalézt řešení i v menších mřížkách, kde standardní CGP naprosto selhává. Rozdíl v úspěšnosti je obrovský. Malá mřížka má však negativní dopad na dynamickou redukcí funkcí.

U mřížky 10×7 (graf 5.8 a tabulka 5.9) má z hlediska počtu generací potřebných k nalezení řešení nejlepší výsledky metoda E, jejíž oba kvartily (zejména velký rozdíl je znatelný u 3. kvartilu), medián i maximum mají nižší hodnoty než CGP s plnou množinou. Medián má přibližně stejnou hodnotu jako u CGP s množinou přizpůsobenou pro návrh násobičky. Ani metody G, H a I však nemají špatné výsledky. Jejich minimum a 1. kvartil je dokonce o něco nižší než u metody E, ale kvůli vyšším hodnotám 3. kvartilu zůstávají horšími. Opačná je však situace při porovnání úspěšnosti v nalezení funkčního řešení. Redukce podle relativního zastoupení v aktivních blocích na základě počtu generací má úspěšnost o celých 10 % nižší než ostatní tři metody. Celkově tak vychází v mřížce 10×7 nejlépe půlení množiny.

Poslední sada experimentů byla provedena s nastavením velikost mřížky 12×7 (graf 5.9 a tabulka 5.10). Stejně jako v předchozích případech si můžeme všimnout, že nejlepší výsledky opět poskytují metody E a I. Podívejme se nejprve na metodu E, tedy redukci podle relativního zasotupení v aktivních prvních na základě počtu generací. Minimum je o



Obrázek 5.7: Počet generací potřebný k nalezení násobičky 4b × 4b v poli 8×7

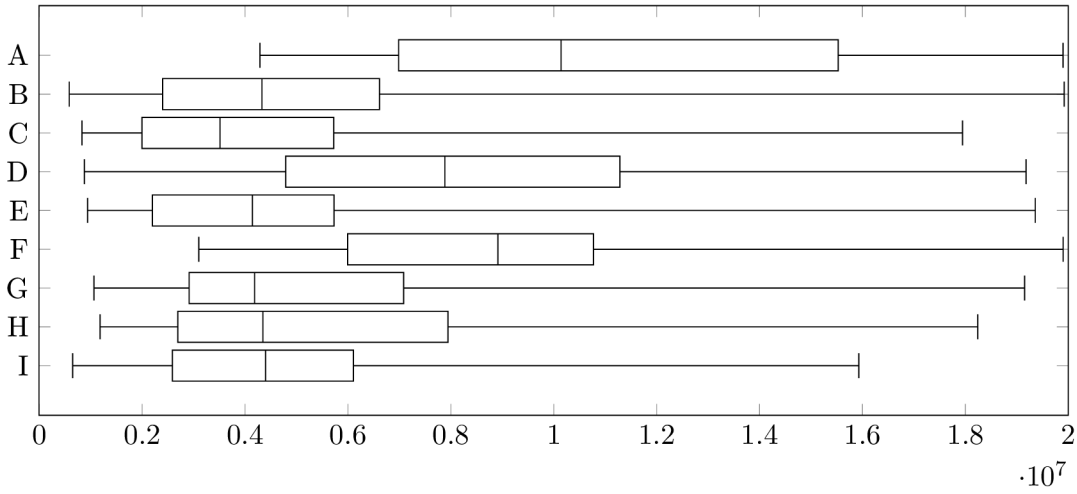


Obrázek 5.8: Počet generací potřebný k nalezení násobičky 4b × 4b v poli 10×7

něco vyšší než při použití plné množiny, ale má nižší kvartily, medián i maximum. Má také velmi vysokou úspěšnost – 98 %. Metoda I má srovnatelné minimum a medián. 1. kvartil je o málo vyšší než má použití plné množiny. Nižší hodnotu má však 3. kvartil a výrazně nižší je i maximum. Úspěšnost je 92 %, což je vysoká hodnota. Metody G a H dopadly hůře, i když je rozdíl poměrně malý.

Z hlediska minimalizace jsou nejlepší metody používající půlení množiny. Velmi důležité je si všimnout, že v tomto ohledu předčily i manuálně redukovanou množinu a to poměrně výrazně. Výsledky jsou sice horší než s plnou množinou funkcí, ale blíží se jim.

Vezmeme-li v úvahu všechny faktory – úspěšnost nalezení navrhovaného obvodu, počet generací potřebný k úspěšnému návrhu a kvalitu minimalizace, vyjde nám nejlépe půlení množiny, které uvažuje všechny prvky. Má srovnatelnou, ne-li vyšší úspěšnost oproti použití plné množiny, při minimalizaci poskytuje výsledky vyšší pouze o minimální hodnotu a řešení dokáže nalét rychleji. Zajímavá je však také informace, že metody založené na fitness hodnotě poskytovaly lepší výsledky s rostoucí velikostí mřížky a tím pádem i počtem prvků v kandidátních řešeních.



Obrázek 5.9: Počet generací potřebný k nalezení násobičky 4b × 4b v poli 12×7

5.4 Celkové srovnání

Při shrnutí všech výsledků je vhodné začít srovnáním standardního CGP a jeho varianty s třívstupovými LUT a plnou množinou funkcí. Ve všech případech použití třívstupových funkcí výrazně snížilo počet generací potřebný k návrhu plně funkčního řešení. Dále se s vyšší úspěšností dařilo takové řešení nalézt. Když vezmeme v úvahu část obvodu realizovanou dvouvstupovými hradly, kterou je možno realizovat pomocí třívstupových funkcí, vyjde nám, že z hlediska minimalizace obvodu použití třívsutpových LUT neposkytuje tak dobré výsledky jako použití hradel. Jedním z možných důvodů tohoto chování je vysoký počet funkcí.

Porovnáním CGP s LUT a plnou množinou funkcí s variantou, která používá množinu přizpůsobenou přímo pro návrh jendotlivých obvodů (tedy předem redukovanou) se potvrzuje to, co platí i u standardní varianty CGP. Menší počet správně zvolených funkcí ovlivňuje celý návrh výrazně pozitivním směrem. Vhodně vybrat třívstupové funkce je však značně složitější než v případě dvouvstupových funkcí. Funkce jsou komplexnější a je tedy těžší předvídat jejich přesný dopad na evoluci. Nedokonale zredukovaná množina má negativní dopad na minimalizaci obvodu, a to i v případě, že první řešení bylo nalezeno rychleji než s plnou množinou.

Při experimentech s metodami využívajícími k adaptaci fitness hodnotu nejlepšího kandidátního řešení se ukázalo, že je tato metoda nevhodná. Výsledky však naznačují závislost na počtu prvků, které se při redukci množiny uvažují. Jednou z možností, proč problém u této metody nastává, je poměr počtu funkcí a počtu prvků. Jestliže je prvků pouze zlomek z celkového počtu funkcí v množině, není možné přesně určit přínos jednotlivých funkcí pro evoluci. Ve velké většině případů se může stát, že každá funkce bude v uvažovaných prvcích zastoupena pouze jednou. Protože obvod není při redukci ještě kompletní, je tak velmi nepředvídatelné, které funkce se do nové množiny dostanou. Do budoucna by mohlo být přínosné prozkoumat chování těchto metod v mnohanásobně větších polích, než které byly použity v našich experimentech. Jako vhodná by se také mohla ukázat metoda, která detekuje přínosné funkce důmyslnějším způsobem. Bylo by možné sledovat shluky funkcí nebo vývoj nejlepších řešení a na základě zaznamenaných dat množinu redukovat.

Naopak dobré výsledky byly zaznamenány při použití půlení množiny. Zejména v kom-

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	2	48	48	48	48	48	48
LUT3 s plnou množinou	72	36	51	41.61	33	46	37.61
LUT3 s redukovanou množinou	78	37	51	43.56	33	48	40.18
LUT3, relativní, aktivní, fitness	44	39	51	45.5	36	50	43.5
LUT3, relativní, aktivní, generace	70	35	49	41.6	34	47	38.83
LUT3, relativní, vše, fitness	10	41	49	44.8	39	48	43.2
LUT3, relativní, vše, generace	52	33	48	41.35	31	48	38.35
LUT3, půlení, aktivní, generace	80	37	48	41.5	33	46	37.48
LUT3, půlení, vše, generace	86	33	50	41.37	31	49	38.49

Tabulka 5.8: Kvalita řešení násobičky 4b × 4b v poli 8×7

binaci s uvažováním všech prvků tato metoda poskytovala vysokou úspěšnost při nalézání plně funkčních řešení a neomezovala minimalizaci výrazným způsobem jako ostatní metody. Ve srovnání s CGP používajícím plnou množinu funkcí došlo i k částečnému urychlení návrhu.

V závislosti na velikosti mřížky CGP s třívstupovými LUT následuje trend, který je zřetelný u jeho standardní varianty. Větší mřížka umožňuje nacházet plně funkční obvody snáze, ale ze cenu míry minimalizace nalezeného řešení.

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	18	51	57	54.22	47	56	52,22
LUT3 s plnou množinou	100	33	52	44.86	32	48	39
LUT3 s redukovanou množinou	98	40	56	48.41	37	53	42.27
LUT3, relativní, aktivní, fitness	66	45	59	50.33	38	56	45.88
LUT3, relativní, aktivní, generace	82	38	52	44.76	33	47	39.24
LUT3, relativní, vše, fitness	28	48	59	53.07	43	55	49.57
LUT3, relativní, vše, generace	92	39	54	45.17	32	47	39.83
LUT3, půlení, aktivní, generace	90	38	55	46.09	33	46	39.27
LUT3, půlení, vše, generace	92	38	53	45.39	33	49	39.5

Tabulka 5.9: Kvalita řešení násobičky 4b × 4b v poli 10×7

verze CGP	Úspěšnost	Počet uzlů prvního řešení			Počet uzlů		
	[%]	MIN	MAX	AVG	MIN	MAX	AVG
Standardní	32	50	67	56	48	63	52.88
LUT3 s plnou množinou	90	37	58	47.78	32	51	39.31
LUT3 s redukovanou množinou	98	42	62	52.14	35	58	43.78
LUT3, relativní, aktivní, fitness	80	46	67	54.58	39	59	48.45
LUT3, relativní, aktivní, generace	98	41	62	48.43	34	61	41.73
LUT3, relativní, vše, fitness	52	47	69	57.12	42	68	51.69
LUT3, relativní, vše, generace	92	39	57	48.09	33	52	41.13
LUT3, půlení, aktivní, generace	90	38	58	48.11	33	51	40.71
LUT3, půlení, vše, generace	92	40	56	48.15	31	48	40.83

Tabulka 5.10: Kvalita řešení násobičky 4b × 4b v poli 12×7

Kapitola 6

Závěr

V této práci byl představen princip CGP s třívstupovými LUT, které bylo srovnáno se standardní variantou CGP. Zejména byla popsána problematika paralelní simulace, která se při použití vícevstupových LUT komplikuje. Na předem zredukovaných množinách funkcí bylo ukázáno, že redukce množiny má význam i u CGP s třívstupovými funkcemi a usnadňuje návrh. Bylo navrženo několik metod pro dynamickou redukci množiny, u kterých byl zkoumán vliv na efektivitu návrhu tří různých obvodů. Experimenty odhalily, že některé metody jsou nevhodné a bylo popsáno několik možných důvodů těchto výsledků. Rovněž byla navržena možná řešení. Experimenty také ukázaly pozitivní vliv půlení množiny funkcí na rychlost návrhu plně funkčního řešení zadaných úloh. Této vlastnosti by bylo možné dalo využít v kombinaci s běžně používanou variantou CGP k urychlení návrhu. Části výsledků byla publikována na studentské konferenci Excel@FIT [3].

Příloha A

Výčet třívstupových funkcí

```
0: 0;
1: (~(a | b | c));
2: (~a & ~b & c);
3: ~a & ~b;
4: (~a & b & ~c);
5: ~a & ~c;
6: (~a & (b ^ c));
7: ~a & ~(b & c);
8: ~a & b & c;
9: ~a & ~(b ^ c);
10: ~a & c;
11: ~a & ~(b & ~c);
12: ~a & b;
13: ~a & ~(~b & c);
14: ~a & (b | c);
15: ~a;
16: a & ~b & ~c;
17: ~b & ~c;
18: ~b & (a ^ c);
19: ~b & ~(a & c);
20: ~c & (a ^ b);
21: ~c & ~(a & b);
22: (~a & (b ^ c)) | (a & ~b & ~c);
23: (a & ~b & ~c) | (~a & ~(b & c));
24: (a ^ b) & (a ^ c);
25: (a ^ b) & (a ^ c) | (~a & ~b & ~c);
26: (a ^ b) & (a ^ c) | (~a & c);
27: (a & ~b & ~c) | (~a & ~(b & ~c));
28: (a & ~b & ~c) | (~a & b);
29: (a & ~b & ~c) | (~a & ~(~b & c));
30: (a & ~b & ~c) | (~a & (b | c));
31: (~a | (a & ~b & ~c));
32: a & ~b & c;
33: ~b & ~(a ^ c);
34: ~b & c;
35: ~b & (~a | c);
36: (a & ~b & c) | (~a & b & ~c);
37: (a & ~b & c) | (~a & ~c);
38: (a & ~b & c) | (~a & (b ^ c));
39: (a & ~b & c) | (~a & ~(b & c));
40: c & (a ^ b);
41: (a & ~b & c) | (~a & ~(b ^ c));
42: (a & ~b & c) | (~a & c);
43: (a & ~b & c) | (~a & ~(b & ~c));
44: (a & ~b & c) | (~a & b);
45: (a & ~b & c) | (~a & ~(~b & c));
46: (~a & (b | c)) | (a & ~b & c);
47: (~a | (a & ~b & c));
48: a & ~b;
49: (a & ~b) | (~a & ~b & ~c);
50: (a & ~b) | (~a & ~b & c);
51: (a & ~b) | (~a & ~b);
52: (a & ~b) | (~a & b & ~c);
53: (a & ~b) | (~a & ~c);
54: (a & ~b) | (~a & (b ^ c));
55: (a & ~b) | (~a & ~(b & c));
56: (a & ~b) | (~a & b & c);
57: (a & ~b) | (~a & ~(b ^ c));
58: (a & ~b) | (~a & c);
59: (a & ~b) | (~a & ~(b & ~c));
60: (a & ~b) | (~a & b);
61: (a & ~b) | (~a & (b | ~c));
62: (a & ~b) | (~a & (b | c));
63: (~a | (a & ~b));
64: a & b & ~c;
65: (~c & ~(a ^ b));
66: (~a & ~b & c) | (a & b & ~c);
67: ((~a & ~b) | (a & b & ~c));
68: (~a & b & ~c) | (a & b & ~c);
69: ((~a & ~c) | (a & b & ~c));
70: ((~a & (b ^ c)) | (a & b & ~c));
71: ((~a & (~b | ~c)) | (a & b & ~c));
72: ((~a & b & c) | (a & b & ~c));
73: ((~a & ~(b ^ c)) | (a & b & ~c));
74: ((~a & c) | (a & b & ~c));
75: ((~a & (~b | c)) | (a & b & ~c));
76: (~a & b) | (a & b & ~c);
```

```
77: ((~a & (b | ~c)) | (a & b & ~c));
78: (~a & (b | c)) | (a & b & ~c);
79: (~a | (a & b & ~c));
80: a & ~c;
81: (a & ~c) | (~a & ~b & ~c);
82: (a & ~c) | (~a & ~b & c);
83: (a & ~c) | (~a & ~b);
84: (a & ~c) | (~a & b & ~c);
85: (a & ~c) | (~a & ~c);
86: (a & ~c) | (~a & (b ^ c));
87: (a & ~c) | (~a & ~(b & c));
88: (a & ~c) | (~a & b & c);
89: (a & ~c) | (~a & ~(b ^ c));
90: (a & ~c) | (~a & c);
91: (a & ~c) | (~a & (~b | c));
92: (a & ~c) | (~a & b);
93: (a & ~c) | (~a & (b | ~c));
94: (~a & (b | c)) | (a & ~c);
95: (~a | (a & ~c));
96: a & (b ^ c);
97: (a & (b ^ c)) | (~a & ~b & ~c);
98: (a & (b ^ c)) | (~a & ~b & c);
99: (a & (b ^ c)) | (~a & ~b);
100: (a & (b ^ c)) | (~a & b & ~c);
101: (a & (b ^ c)) | (~a & ~c);
102: (a & (b ^ c)) | (~a & (b ^ c));
103: (a & (b ^ c)) | (~a & ~(b & c));
104: (a & (b ^ c)) | (~a & b & c);
105: (a & (b ^ c)) | (~a & ~(b ^ c));
106: (a & (b ^ c)) | (~a & c);
107: (a & (b ^ c)) | (~a & (~b | c));
108: (a & (b ^ c)) | (~a & b);
109: (a & (b ^ c)) | (~a & (b | ~c));
110: (a & (b ^ c)) | (~a & (b | c));
111: (~a | (a & (b ^ c)));
112: a & ~(b & c);
113: (a & ~(b & c)) | (~a & ~b & ~c);
114: (a & ~(b & c)) | (~a & ~b & c);
115: (a & ~(b & c)) | (~a & ~b);
116: (a & ~(b & c)) | (~a & b & ~c);
117: (a & ~(b & c)) | (~a & ~c);
118: (a & ~(b & c)) | (~a & (b ^ c));
119: (a & ~(b & c)) | (~a & ~(b & c));
120: (a & ~(b & c)) | (~a & b & c);
121: (a & ~(b & c)) | (~a & ~(b ^ c));
122: (a & ~(b & c)) | (~a & c);
123: (a & ~(b & c)) | (~a & ~(b & ~c));
124: (a & ~(b & c)) | (~a & b);
125: (a & ~(b & c)) | (~a & (b | ~c));
126: (a & ~(b & c)) | (~a & (b | c));
127: (~a | (a & ~(b & c)));
128: a & b & c;
129: (a & b & c) | (~a & ~b & ~c);
130: (a & b & c) | (~a & ~b & c);
131: ((~a & ~b) | (a & b & c));
132: (~a & b & ~c) | (a & b & c);
133: ((~a & ~c) | (a & b & c));
134: (~a & (b ^ c)) | (a & b & c);
135: ((~a & ~(b & c)) | (a & b & c));
136: (~a & b & c) | (a & b & c);
137: ((~a & ~(b ^ c)) | (a & b & c));
138: (~a & c)  | (a & b & c)& 0xFF;
139: ((~a & ~(b & ~c)) | (a & b & c));
140: (~a & b) | (a & b & c);
141: ((~a & ~(~b & c)) | (a & b & c));
142: (~a & (b | c)) | (a & b & c);
143: (~a | (a & b & c));
144: a & ~(b ^ c);
145: (a & ~(b ^ c)) | (~a & ~b & ~c);
146: (a & ~(b ^ c)) | (~a & ~b & c);
147: (a & ~(b ^ c)) | (~a & ~b);
148: (a & ~(b ^ c)) | (~a & b & ~c);
149: (a & ~(b ^ c)) | (~a & ~c);
150: (a & ~(b ^ c)) | (~a & (b ^ c));
151: ((~a & ~(b & c)) | (a & ~(b ^ c)));
152: (a & ~(b ^ c)) | (~a & b & c);
153: (a & ~(b ^ c)) | (~a & ~(b ^ c));
154: (a & ~(b ^ c)) | (~a & c);
155: ((~a & ~(b & ~c)) | (a & ~(b ^ c)));
156: (a & ~(b ^ c)) | (~a & b);
157: ((~a & ~(~b & c)) | (a & ~(b ^ c)));
158: ((~a & (b | c)) | (a & ~(b ^ c)));
159: (~a | (a & ~(b ^ c)));
160: a & c;
161: (a & c) | (~a & ~b & ~c);
162: (a & c) | (~a & ~b & c);
163: (a & c) | (~a & ~b);
164: (a & c) | (~a & b & ~c);
165: (a & c) | (~a & ~c);
166: (a & c) | (~a & (b ^ c));
167: ((~a & ~(b & c)) | (a & c));
168: (a & c) | (~a & b & c);
169: (a & c) | (~a & ~(b ^ c));
170: (a & c) | (~a & c);
171: ((~a & ~(b & ~c)) | (a & c));
172: (~a & b) | (a & c);
173: ((~a & ~(~b & c)) | (a & c));
174: (~a & (b | c)) | (a & c);
175: (~a | (a & c));
176: a & (~b | c);
177: (a & (~b | c)) | (~a & ~b & ~c);
178: (a & (~b | c)) | (~a & ~b & c);
179: (a & (~b | c)) | (~a & ~b);
180: (a & (~b | c)) | (~a & b & ~c);
181: (a & (~b | c)) | (~a & ~c);
182: (a & (~b | c)) | (~a & (b ^ c));
183: (a & (~b | c)) | (~a & ~(b & c));
184: (a & (~b | c)) | (~a & b & c);
```

```
185: (a & (~b | c)) | (~a & ~(b ^ c));
186: (a & (~b | c)) | (~a & c);
187: (a & (~b | c)) | (~a & ~(b & ~c));
188: (a & (~b | c)) | (~a & b);
189: (a & (~b | c)) | (~a & ~(~b & c));
190: (a & (~b | c)) | (~a & (b | c));
191: (~a | (a & (~b | c)));
192: a & b;
193: (a & b) | (~a & ~b & ~c);
194: (a & b) | (~a & ~b & c);
195: (a & b) | (~a & ~b);
196: (a & b) | (~a & b & ~c);
197: (a & b) | (~a & ~c);
198: (a & b) | (~a & (b ^ c));
199: (a & b) | (~a & ~(b & c));
200: (a & b) | (~a & b & c);
201: (a & b) | (~a & ~(b ^ c));
202: (a & b) | (~a & c);
203: (a & b) | (~a & ~(b & ~c));
204: (a & b) | (~a & b);
205: (a & b) | (~a & (b | ~c));
206: (a & b) | (~a & (b | c));
207: (~a | (a & b));
208: a & (b | ~c);
209: (a & (b | ~c)) | (~a & ~b & ~c);
210: (a & (b | ~c)) | (~a & ~b & c);
211: (a & (b | ~c)) | (~a & ~b);
212: (a & (b | ~c)) | (~a & b & ~c);
213: (a & (b | ~c)) | (~a & ~c);
214: (a & (b | ~c)) | (~a & (b ^ c));
215: (a & (b | ~c)) | (~a & ~(b & c));
216: (a & (b | ~c)) | (~a & b & c);
217: (a & (b | ~c)) | (~a & ~(b ^ c));
218: (a & (b | ~c)) | (~a & c);
219: (a & (b | ~c)) | (~a & ~(b & ~c));
220: (a & (b | ~c)) | (~a & b);
221: (a & (b | ~c)) | (~a & ~(~b & c));
222: (a & (b | ~c)) | (~a & (b | c));
223: (~a | (a & (b | ~c)));
224: a & (b | c);
225: (a & (b | c)) | (~a & ~b & ~c);
226: (a & (b | c)) | (~a & ~b & c);
227: (a & (b | c)) | (~a & ~b);
228: (a & (b | c)) | (~a & b & ~c);
229: (a & (b | c)) | (~a & ~c);
230: (a & (b | c)) | (~a & (b ^ c));
231: (a & (b | c)) | (~a & ~(b & c));
232: (a & (b | c)) | (~a & b & c);
233: (a & (b | c)) | (~a & ~(b ^ c));
234: (a & (b | c)) | (~a & c);
235: (a & (b | c)) | (~a & ~(b & ~c));
236: (a & (b | c)) | (~a & b);
237: (a & (b | c)) | (~a & ~(~b & c));
238: (a & (b | c)) | (~a & (b | c));
239: (~a | (a & (b | c)));
240: a;
241: a | (~a & ~b & ~c);
242: a | (~a & ~b & c);
243: a | (~a & ~b);
244: a | (~a & b & ~c);
245: a | (~a & ~c);
246: a | (~a & (b ^ c));
247: a | (~a & ~(b & c));
248: a | (~a & b & c);
249: a | (~a & ~(b ^ c));
250: a | (~a & c);
251: a | (~a & ~(b & ~c));
252: a | (~a & b);
253: a | (~a & ~(~b & c));
254: a | b | c;
255: 0b11111111;
```

Literatura

[1] Burian, P.: Vlastnosti kartézského genetického programování. In *Elektrotechnika a informatika. Část 2*, Elektronika, ZCU v Plzni, 2010, s. 21 – 24.

[2] Chalupník, V.: Biologické algoritmy (3) - Evoluční algoritmy. [online] http://www.root.cz/clanky/biologicke-algoritmy-3-evolucni-algoritmy/.

[3] Hajná, K.: Kartézské genetické programování s LUT. In *Excel@FIT*, VUT v Brně, 2015.

[4] Hynek, J.: *Genetické algoritmy a genetické programování*. Grada, 2008, iSBN 978-80-247-2695-3.

[5] J.F, W.: *Digital Design - Principles and Practices*. Prentice Hall, 2001, iSBN 0131863894.

[6] Miller, J.: Cartesian Genetic Programming. [online] http://www.cartesiangp.co.uk/.

[7] Miller, J.: *Cartesian genetic programming*. New York: Springer, 2011, iSBN 9783642173103.

[8] Sekanina, L.: Biologií inspirované počítače. [online] https://www.fit.vutbr.cz/study/courses/BIN/private/.cs.

[9] Sekanina, L.; a kolektiv: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Gerstner, 2010, iSBN 978-80-200-1729-1.

[10] Škrabal, O.: *Genetické algoritmy a rozvrhování*. Diplomová práce, VUT v Brně, 2010.