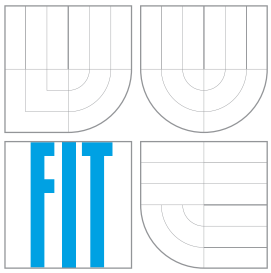
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DYNAMICKÁ APROXIMACE ČÍSLICOVÝCH OBVODŮ

DYNAMIC APPROXIMATION OF DIGITAL CIRCUITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL JÁSENSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2016

Zadání bakalářské práce/10045/2015/xjasen00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Jásenský Michal**
Obor: Informační technologie
Téma: **Dynamická aproximace číslicových obvodů**
Dynamic Approximation of Digital Circuits
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s problematikou přibližného počítání a s metodami evolučního návrhu kombinačních obvodů. Zaměřte se na kartézské genetické programování (CGP).
2. S využitím principů CGP navrhněte metodu, která bude umožňovat evoluční návrh obvodů schopných dynamické rekonfigurace. Cílem rekonfigurace je dynamicky zmenšovat/zvyšovat počet použitých komponent v obvodu a tím snižovat/zvyšovat přesnost výpočtu.
3. Implementujte navrženou metodu a experimentálně ji ověřte.
4. Pro zvolené obvody demonstrujte požadované chování - dynamickou aproximaci.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, prof. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá návrhem metody postavené na kartézském genetickém programování, která umožňuje evoluční návrh obvodů schopných dynamické rekonfigurace. Cílem rekonfigurace obvodu je dynamicky měnit počet použitých komponent v obvodu a tím měnit přesnost výpočtu. Je zde popsána implementace navržené metody. Metoda je experimentálně ověřena a demonstrována na několika zvolených obvodech.

Abstract

This bachelor's thesis deals with design of a method based on cartesian genetic programming, which allows the evolutionary design of circuits capable of dynamic reconfiguration. The goal of reconfiguration is to dynamically change the number of used components and thereby to change the accuracy of calculation. In this thesis, implementation of the proposed method is described. The method is experimentally verified and demonstrated on several selected circuits.

Klíčová slova

dynamická aproximace, číslicové obvody, přibližné počítání, kartézské genetické programování

Keywords

dynamic approximation, digital circuits, approximate computing, cartesian genetic programming

Citace

JÁSENSKÝ, Michal. *Dynamická aproximace číslicových obvodů.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Sekanina Lukáš.

Dynamická aproximace číslicových obvodů

Prohlášení

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

.....
Michal Jásenský
16. května 2016

Poděkování

Velmi rád bych poděkoval vedoucímu mé bakalářské práce panu Prof. Ing. Lukáši Sekaninovi, Ph.D. za odborné rady, připomínky, čas a ochotu při konzultacích.

Obsah

Kapitola 1

Úvod

V určitých oblastech použití číslicových obvodů není vyžadována absolutní přesnost výpočtu a lze se spokojit i s lehce nepřesnými aproximovanými výsledky. Jedná se například o obvody, u kterých se vyžaduje výsledek ve velmi krátké a přesně stanovené době, obvody s nízkým příkonem nebo třeba obrazové filtry, u kterých lidské oko není schopno rozeznat drobnou odchylku v jasu pixelu [5]. Cílem této práce je takovéto obvody generovat a umožnit tímto obvodům dynamicky měnit počet použitých komponent a tím měnit přesnost výsledků produkovaných těmito obvody. O něco podobného, ovšem za použití jiných metod, se snaží například v [2]. Dynamická aproximace je nové výzkumné téma v oblasti, kterou se zabývá výzkumná skupina FIT - Evolvable hardware.

V následujícím textu je nejprve představen evoluční návrh číslicových obvodů v kapitole 2. Detailněji pak algoritmus zvaný kartézské genetické programování (CGP) v kapitole 3.

Implementační detaily a úprava původního CGP, která byla použita pro evoluční aproximaci, jsou popsány v kapitole 4.

Jednotlivé experimenty jsou shrnuty v kapitole 5 a závěrečné zhodnocení práce v kapitole 6.

Kapitola 2

Evoluční algoritmy

Termínem evoluční algoritmy (EA) 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.Popis EA je zpracován podle [1].

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.

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
- potencionální řešení

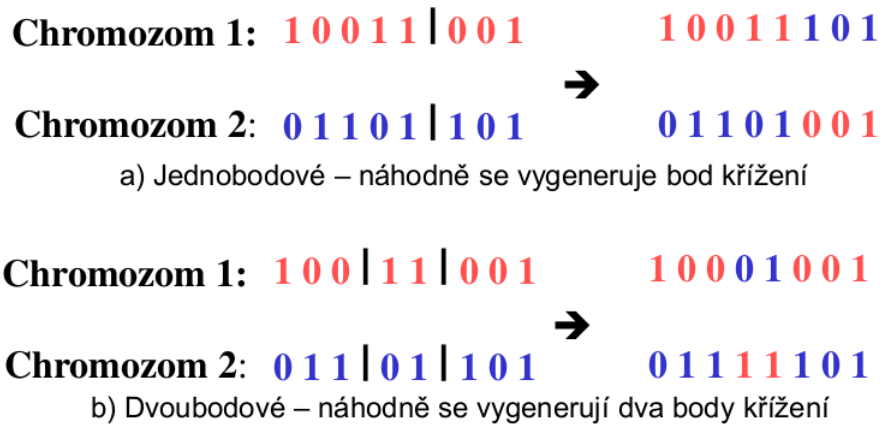
Populace

- složena z jedinců
- obvykle pevná velikost

2.3 Genetické operátory

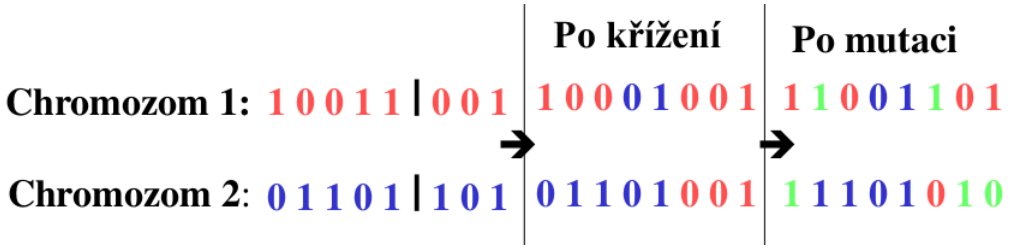
Abychom mohli v průběhu evoluce vytvářet nové jedince, potřebujeme genetické operátory. Tyto operátory se aplikují na jednoho nebo více jedinců že stávající populace. Rozlišujeme čtyři základní operátory - křížení, mutace, selekce a nahrazení.

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 možností Křížení je aplikováno s určitou pravděpodobností. Jeho princip je na obrázku 2.1.



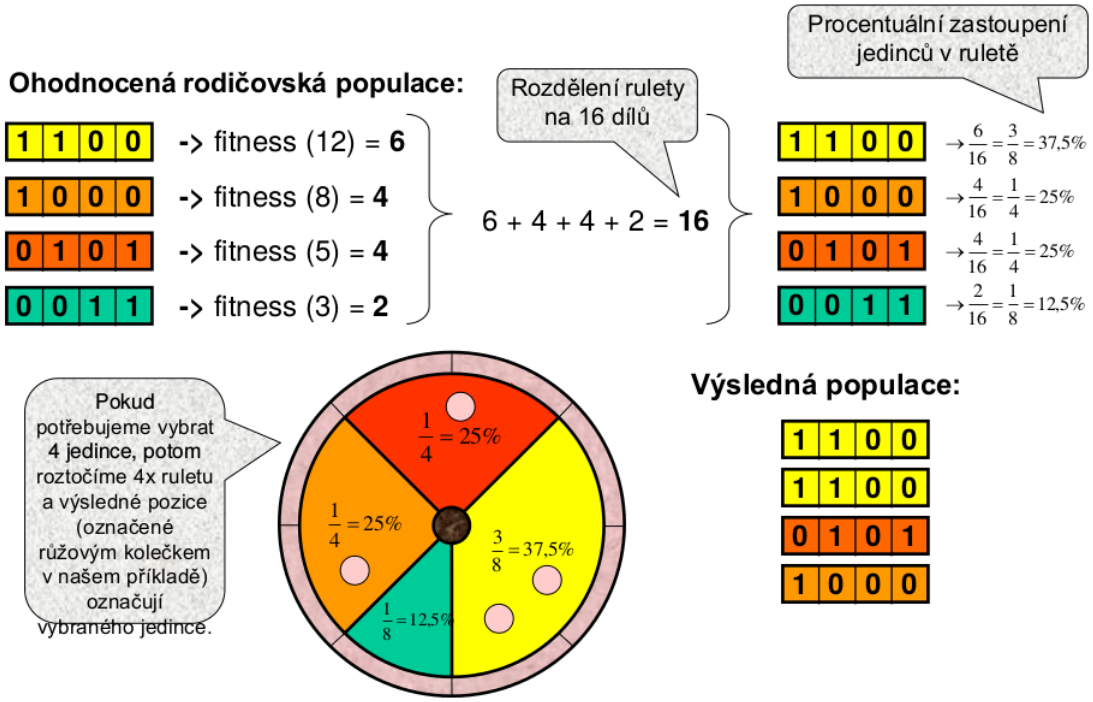
Obrázek 2.1: Křížení v evolučních algoritmech, převzato z [4]

Mutace spočívá v náhodné změně jednoho nebo více genů jedince. Tento operátor bývá použit s malou pravděpodobností. Princip mutace je na obrázku 2.2.



Obrázek 2.2: Mutace v evolučních algoritmech, převzato z [4]

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 2.3), 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 2.3: Výběr rodiče pomocí rulety, převzato z [4]

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

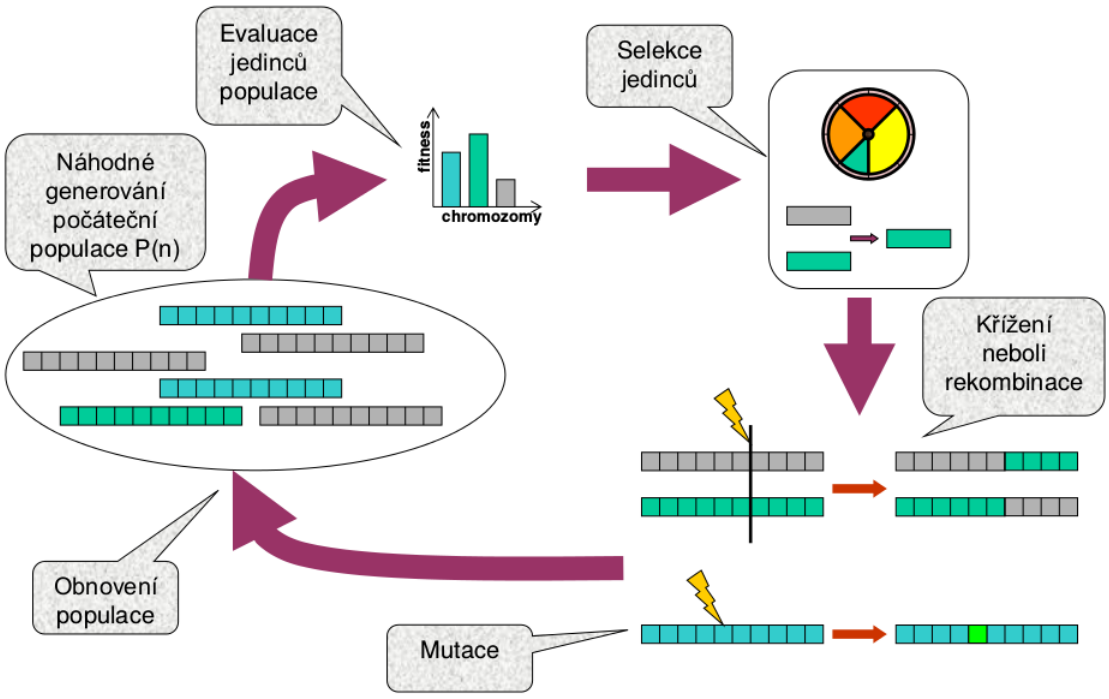
bychom nemuseli vůbec dosáhnout řešení. Důležitou roli hraje také nutnost oprav chromozomu po aplikaci operátorů. Chromozomy mohou mít pouze omezenou množinu povolených kombinací jednotlivých genů. Aplikace operátorů může mít za následek, že chromozom obsahuje jednu z nepovolených kombinací. Zde přichází 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 2.4.



Obrázek 2.4: Evoluční algoritmus, převzato z [4]

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 operátory 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 : [6].

$$T_e = G \cdot (P \cdot T_a + T_p) \tag{2.1}$$

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

Kartézské genetické programování je evoluční algoritmus, který kóduje zadaný problém do acyklické orientované grafové struktury reprezentované pomocí celých čísel. uplatňuje se při návrhu kombinačních obvodů, v umělé inteligenci a strojovém učení [3], při řešení matematických problémů, v evolučním umění a mnoha dalších oblastech. 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, proto se následující text bude zabývat výhradně touto problematikou. Popis CGP je zpracován podle [1].

3.1 Reprezentace obvodu

Kandidátní obvod je popsán jako dvourozměrné pole 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.

Při návrhu logických obvodů mohou elementy 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.

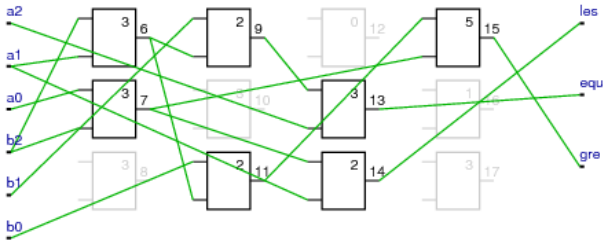
3.2 Kódování chromozomu

Evoluční algoritmy pracují se zakódovanými kandidátními řešeními (chromozomy) problému. Jelikož v CGP máme matici programovatelných bloků, je chromozom zakódován do celočíselných řetězců délky $r \times c \times (n + 1) + o$, kde r je počet řádků, c je počet sloupců a n je počet vstupů jednoho uzlu. Všechny elementy v matici a primární vstupy a výstupy mají přiřazeny unikátní indexy. Nejprve se číslují primární vstupy od nuly a poté elementy

shora dolů a zleva doprava. Každý blok je reprezentován pomocí $n+1$ hodnot. Prvních n hradel udává indexy výstupů, kam jsou připojeny vstupy bloku, poslední hodnota je rezervována pro reprezentovanou funkci. Poslední část tvoří o-tice s velikostí shodnou s počtem primárních výstupů obvodu. Jednotlivé hodnoty v ní představují výstupy elementů nebo primární vstupy, na které jsou primární výstupy připojeny.

Uvažujme následující tvar chromozomu: (3,1,3) (2,3,3) (2,0,3) (4,6,2) (4,4,3) (5,6,2) (9,8,0) (9,0,3) (7,1,2) (11,7,5) (7,11,1) (11,12,3) (14,13,15). Tento chromozom kóduje dvouvstupový rozšířený komparátor s šesti vstupy a třemi výstupy v rekonfigurovatelném poli 4×3 elementů. Chromozomu odpovídá propojení elementů matice a přiřazení funkcí, které je zobrazeno na obrázku 3.1. Chromozom převzat z [7].

Čísla uvnitř bloků určují funkci bloku (2-OR, 3-XOR, 5-NOT(in2)). Číslo výstupu je zapsáno vedle bloku a značí zároveň index daného bloku. Šedou barvou jsou označeny bloky, které se nepodílejí na řešení obvodu



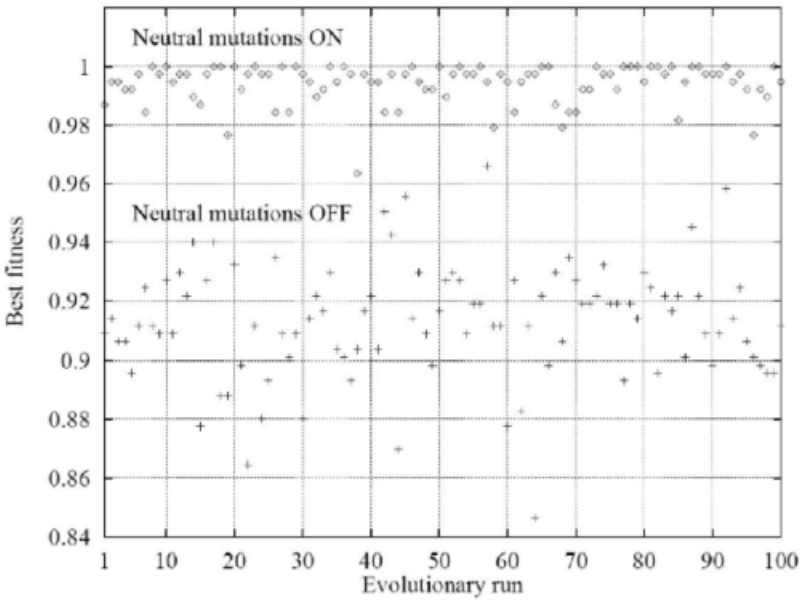
Obrázek 3.1: Dvouvstupový rozšířený komparátor, převzato z [7]

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 4. 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 [6]. 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 vzniknout 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, kde na ose x je 100 nezávislých běhů evolučního návrhu 3-bitové násobičky. 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. V případě více jedinců se stejným ohodnocením je důležité vybírat rodiče pro novou populaci tak, aby se nevolil jako rodič stále jeden jedinec.



Obrázek 3.2: Vliv neutrálních mutací na běh CGP, převzato z [6]

Celý algoritmus CGP je popsán Algoritmem 1.

Algorithm 1: CGP

Input: CGP parametry, fitness funkce
Output: Nejlépe ohodnocený jedinec p a jeho fitness

```
1  Náhodně generovaný rodič p;
2  Q ← {p} ∪ {λ potomků vytvořených mutací z p};
3  EvaluacePopulace(Q);
4  while ⟨ukončující podmínka není splněna⟩ do
5      α ← nejlépe-ohodnocený-jedinec(Q);
6      if fitness(α) ≥ fitness(p) then
7          p ← α;
8      Q ← {p} ∪ {λ potomků z p};
9      EvaluacePopulace(Q);
10 return p, fitness(p);
```

Nejprve se vygeneruje rodič a vytvoří se počáteční populace složená z $1 + \lambda$ jedinců. Následně se ohodnotí každý jedinec populace. Dokud není splněna ukončující podmínka, tak se vykonávají následující kroky. Vybere se z populace nejlépe ohodnocený jedinec. Je-li nejlepších jedinců více, vybere se jedinec, který nebyl rodičem v předchozí generaci. Z vybraného jedince se vygeneruje nová populace a ohodnotí se.

3.4 Fitness funkce

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

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 jejich ceny, zpoždění a dalších požadavků. V této práci je použita fitness funkce, která po nalezení plně funkčního řešení zohledňuje počet členů v navrženém obvodu.

Kapitola 4

Navržená metoda

Navržená metoda má za cíl pomocí úprav standardního CGP generovat obvody schopné dynamické rekonfigurace. Metoda musí zohledňovat dělení obvodů na jednotlivé bloky a přitom zajistit vzrůstající kvalitu generovaných výsledků s přidáváním jednotlivých bloků. Motivací pro vytvoření obvodů se schopností se dynamicky „zvětšovat“, pokud je požadována vyšší kvalita výpočtu, a naopak se „zmenšovat“, pokud je cílem redukovat příkon za cenu větší chybovosti, je snaha o co nejefektivnější využití dostupných zdrojů a energie během provádění výpočtu.

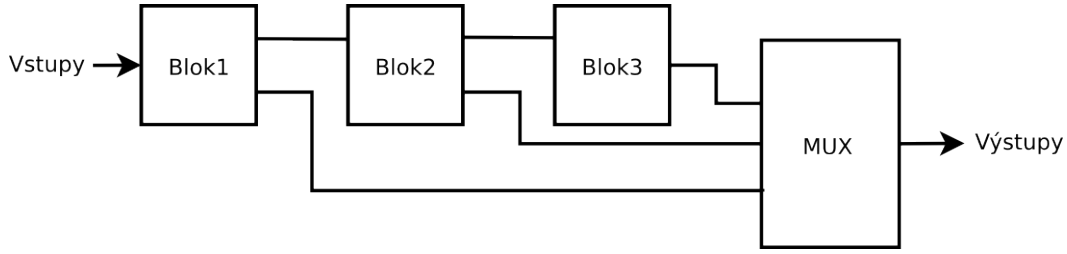
Metoda je implementována v jazyce C++. Jako základ implementace byl posloužit kód z laboratorního cvičení k magisterskému předmětu Biologií inspirované počítače [7]. Celá práce je zaměřena na jednoduché kombinační obvody.

4.1 Pojmy

Navržená metoda přidává k základnímu algoritmu CGP několik nových funkcionalit a je třeba pro lepší porozumění problematice zavést určité pojmy, které jsou zde vysvětleny.

4.1.1 Blok

Cílem práce je nalézt takový obvod, který umí měnit počet použitých hradel. Hradla se ovšem neodebírají a nepřidávají po jednom, ale po skupině hradel. Této skupině budeme říkat blok. Každý blok má své výstupy. Kvalita výstupů přidáváním bloků postupně vzrůstá. Například blok 3 má vyšší kvalitu výstupů než blok 2 a ten má vyšší kvalitu výstupů než blok 1. Každý blok vyžaduje ke správné funkčnosti bloky předchozí. Nelze tedy použít například blok 1 a 3 bez bloku 2. Výstupy bloků jsou přivedeny do multiplexoru a na výstup multiplexoru je distribuován výstup nejvyššího použitého bloku, tedy ten nejkvalitnější výstup. Příklad obvodu se třemi bloky lze vidět na obrázku 4.1.



Obrázek 4.1: Rozdělení obvodu na tři bloky

4.1.2 Řez

CGP pracuje nad maticí hradel. Tuto matici je zapotřebí rozdělit podle potřeby na jednotlivé bloky. To provedeme pomyslným rozříznutím matice na části. V poli *cuts* v souboru *cgp.h* lze nadefinovat jednotlivé řezy. Řezy se provádí po sloupcích a v poli jsou seřazeny od nejnižšího sloupce k nejvyššímu. Řezy udávají, z kterého sloupce můžou být ještě brány výstupy pro daný blok. První číslice v poli *cuts* udává, odkud může první blok ještě brát své výstupy, druhá číslice určuje maximální sloupec druhého bloku atd. Tato informace je důležitá pro správnou funkci mutace.

4.1.3 Hranice

Při použití standardního CGP se nejprve nalezne funkční zapojení. To znamená, že hodnota fitness určující funkčnost dosáhne svého maxima. Poté se algoritmus ještě pokusí minimalizovat počet použitých hradel. U navržené metody se hodnota fitness pohybuje od nuly po své maximum. Maximální fitness by ovšem znamenala, že výstupy jednotlivých bloků by měly všechny 100% funkčnost. Takovýto obvod však nehledáme. Je tedy zapotřebí nastavit požadovanou kvalitu obvodu, takzvanou hranici. Pokud hodnota fitness překročí zadanou hranici, CGP se pokusí snížit počet použitých hradel. Bližší informace k výpočtu fitness se dočtete v sekci 4.4. Hranice se nastavuje v souboru *cgp.h*.

4.2 Chromozom

V navržené metodě způsob zápisu chromozomu téměř odpovídá zápisu popsaném v kapitole 3. Je zapotřebí do chromozomu ještě zapsat informace o výstupech jednotlivých bloků. Z toho důvodu je modifikována závěrečná sekce s výstupy. Tvar výstupů je následovný: výstupy posledního bloku,výstupy předposledního bloku,. . . ,výstupy prvního bloku. Například pro 9-bitovou majoritu se třemi bloky by závěrečná sekce mohla vypadat třeba takto: . . . (12,8,5), kde výstup 12 náleží třetímu bloku, výstup 8 náleží druhému bloku a výstup 5 bloku prvnímu.

4.3 Mutace

Mutace v navržené metodě obsahuje dvě malé úpravy oproti standardnímu CGP. První úpravou je zabránění mutace výstupů po provedení redukce sloupců. Důvody jsou popsány v sekci 4.5. Druhá úprava se týká taktéž mutace výstupů. V případě mutace některého výstupu, musí mutace poznat, ke kterému bloku výstup patří, a řídit se řezy určenými v poli *cuts*.

4.4 Fitness funkce

Fitness funkce ve svém základu využívá princip, kdy se na primární vstupy přivedou všechny možné kombinace a následně se porovnají výstupy jedince s očekávanými výstupy. Výsledkem je informace o počtu bitů, ve kterých se shodují výstupy jedince s očekávanými výstupy. Pokud má obvod i vstupů a o výstupů, je maximální fitness rovna $2^i \cdot o$.

V prezentovaném řešení máme ovšem ještě výstupy jednotlivých bloků, čímž značně narůstá celkový počet výstupů obvodu. Maximální fitness ovšem nebere v potaz výstupy jednotlivých bloků, ale nahlíží na obvod jako na celek, který má vždy aktivní výstupy jen z jednoho bloku. Například u 4-bitové sčítačky máme osm vstupů a počet výstupů je roven pěti, bez ohledu na to, kolik řezů v obvodu máme.

Celková fitness se tedy musí pohybovat v rozmezí 0 až maximální fitness. V poli fit_of_cuts se nachází hodnoty fitness v jednotlivých řezech, které dosahují hodnot 0 až maximální fitness. Všechny tyto dílčí fitness hodnoty je třeba promítnout do celkové fitness. Jednotlivým dílčím fitness hodnotám jsou přiřazeny váhy a celková fitness je spočítána podle následujícího vzorce:

$$\begin{aligned} Fitness f = & fit\_of\_cuts[0]\frac{1}{\beta} + fit\_of\_cuts[1]\frac{2}{\beta} + \ldots + fit\_of\_cuts[n-1]\frac{n}{\beta} + \\ & fit\_out\frac{n+1}{\beta} \\ & \beta = 1 + 2 + \ldots + (n+1) \\ & n \geq 0 \end{aligned} \tag{4.1}$$

Ve vzorci n reprezentuje počet řezů. Jelikož počet bloků v obvodu je o jedničku větší než je počet řezů, je zapotřebí započíst do celkové fitness ještě fitness posledního bloku *fit_out*. Ze vzorce plyne, že bloky, u kterých se očekává horší funkčnost, mají nižší váhu než bloky s vyšší funkčností. Jinými slovy, bloky s nižší funkčností ovlivňují celkovou fitness méně než bloky s vyšší funkčností. Pomocí tohoto vzorce se také počítá hranice. Jednoduše se dosadí požadované hodnoty fitness v jednotlivých řezech a vyjde nám požadovaná hranice.

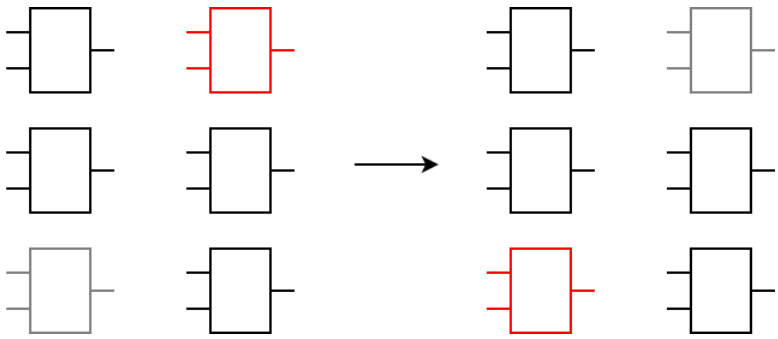
V programu je výpočet dílčích fitness optimalizován. Nevolá se funkce pro výpočet fitness pro každý řez zvlášť, ale při výpočtu fitness posledního bloku se při výpočtu mezivýsledků jednotlivých hradel zjistí i fitness všech předchozích bloků.

Je nutné poznamenat, že volaná fitness funkce představuje zjednodušení oproti případné reálné implementaci, která by vyžadovala zohlednění různých typů chyb a dalších parametrů obvodu.

4.5 Redukce sloupců

Jednou z optimalizací navrhovaného algoritmu je redukce počtu sloupců. Redukcí počtu sloupců se zmenší prohledávaný prostor a tím se urychlí výpočet fitness funkce. Pokus o zmenšení počtu sloupců matice se provede po nalezení hledaného obvodu a potom v případě, že poměr použitých hradel a počtu hradel v matici klesne pod zadanou hodnotu. Redukce se neprovádí před nalezením požadovaného obvodu proto, že není znám počet hradel potřebných k realizaci obvodu. Mohlo by se tedy stát, že redukcí by se zmenšila matice na počet hradel, ve kterém by nebylo možné požadovaný obvod nalézt. Po nalezení obvodu se algoritmus snaží minimalizovat počet použitých hradel a matice tedy může být zmenšena, protože počet použitých hradel může už jen klesat.

Redukce prochází postupně všechny hradla od druhého sloupce k poslednímu a snaží se použitá hradla přesunout do předchozího sloupce. Z tohoto plynou dvě podmínky pro přesun uzlu, které musí platit současně. (1) V předchozím sloupci musí být nevyužitý uzel a (2) nesmí přesunem vzniknout zpětná vazba. Přesun se provede prohozením volného uzlu z předchozího sloupce s použitým uzlem v aktuálním sloupci. Tímto způsobem se prochází matice opakovaně do té doby, až již nelze provést žádný přesun. Na obrázku 4.2 je vidět přesun použitého červeného hradla do předchozího sloupce (šedé hradlo není použito).

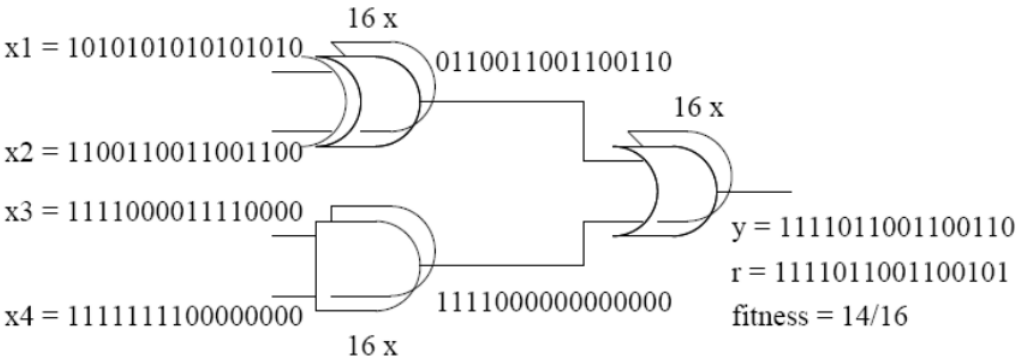


Obrázek 4.2: Přesun použitého hradla do předchozího sloupce

Po provedení všech přesunů můžou být některé poslední sloupce nevyužity. Tyto sloupce lze odstranit a zmenšit tak rozměry matice. Zde ovšem nastává problém s napojením výstupů obvodu. Po zmenšení matice se při mutaci již nelze řídit sloupci nadefinovanými v poli *cuts*. Řešením je po první redukci sloupců nedovolit mutovat výstupy obvodu.

4.6 Akcelerace simulace

Pro ohodnocení kandidátního obvodu s i vstupy je třeba vygenerovat a získat výsledky pro 2^i vstupních vektorů. Pomocí naivního přístupu bychom museli uskutečnit průchod obvodem 2^ikrát pro získání výsledku pro každý vstupní vektor zvlášť. S rostoucím počtem vstupů počet průchodů obvodem exponenciálně roste a tím se neúměrně zvyšuje časová náročnost ohodnocování jedinců populace. Proto se v praxi využívá tzv. paralelní simulace, která využívá faktu, že v programovacích jazycích existují bitové logické operátory umožňující provést jednu logickou operaci současně nad všemi bity operandů v jednom taktu procesoru. V implementaci se využívají jako operandy proměnné typu *long int*. Tyto operandy jsou u 64 bitových architektur procesorů složeny z 64 bitů. Na jeden takt se tak vyhodnotí 64 vstupních vektorů najednou, což znamená urychlení výpočtu 64×. Princip paralelní simulace pro 16 bitové operandy je popsán na obrázku 4.3.



Obrázek 4.3: Paralelní simulace v CGP, převzato z [4]

4.7 Generování testovacích dat

Ne vždy máme po ruce pravdivostní tabulku obvodu, pomocí které bychom sestavili testovací data. Můžeme mít ovšem k dispozici netlist s daným obvodem. Pokud vytvoříme z netlistu počáteční populaci pro CGP, můžeme si testovací data vygenerovat. Tvorba počáteční populace z netlistu je popsána v 4.9.

Nejdříve se upraví velikost matice podle potřeb obvodu uvedeném v netlistu, a to z toho důvodu, aby výpočet testovacích dat nebyl zbytečně dlouhý, pokud by byla matice zbytečně velká. Poté se vytvoří počáteční populace, z které se budou následně generovat testovací data. Následně jsou vygenerovány všechny možné vstupní kombinace obvodu. Výpočet výstupních dat probíhá podobným způsobem jako výpočet fitness. Na vstup obvodu, tedy na vstup jedince z počáteční populace, se přivede vstupní kombinace a vypočítají se výstupní hodnoty. V tuto chvíli by při výpočtu fitness došlo k porovnání dosažených výsledků s požadovanými výsledky, ale zde můžeme dosažené výsledky považovat za ty požadované a uložit si jejich hodnoty do testovacího souboru s příponou *.h*.

V nastavovacím souboru *cgp.h* lze generování testovacích dat nastavit. Parametrem H_GENERUJ lze generování testovacích dat zapnout či vypnout. H_OUTPUT nastavuje cestu k testovacímu souboru, do kterého budou testovací data uložena, a popis k tomuto souboru lze nastavit pomocí H_POPIS.

4.8 Načtení obvodu z netlistu

Implementace umožňuje načítání jedinců ve formátu BLIF. Jelikož na počátku netušíme, jak velká matice bude pro vytvoření počáteční populace zapotřebí, je třeba netlist nejdříve načíst do nějaké vnitřní struktury. Touto strukturou je seznam jednotlivých sloupců a každý sloupec je seznamem jednotlivých hradel. Při načítání netlistu se prochází zapojení od prvního hradla k poslednímu hradlu. Informace o hradlech se ukládají do sloupce. V případě, že je na vstup hradla přiveden výstup hradla předchozího, je potřeba začít nový sloupec. Po dokončení načítání je již známo kolik sloupců a řádků musí matice minimálně obsahovat a lze tak vytvořit počáteční populaci. Pro usnadnění vytváření počáteční populace se ještě upraví číslování jednotlivých vstupů a výstupů hradel tak, aby odpovídalo rozložení v matici s ohledem na volné uzly, které se budou v matici nacházet. Cestu k netlistu lze nastavit v *cgp.h* parametrem BLIF.

4.9 Vytvoření počáteční populace

Před odstartováním CGP algoritmu můžeme vytvořit počáteční populaci náhodně nebo si vytvořit počáteční populaci vlastní. Pro vytvoření počáteční populace potřebujeme znát zapojení obvodu. Tato informace je uložena v netlistu, jehož načítání se věnuje sekce 4.8. Po načtení zapojení do vnitřní struktury seznamů lze počáteční populaci vytvořit. Seznam sloupců se prochází postupně od prvního k poslednímu sloupci. V každém sloupci se prochází jednotlivá hradla a ukládají se do matice. Jakmile jsou všechna hradla v sloupci zapsána do matice, doplní se ještě volné uzly ve sloupci náhodnými hodnotami. Po zápisu všech sloupců ze seznamu se ještě v případě potřeby doplní přebytečné sloupce v matici náhodnými hodnotami.

To, zda se má vygenerovat náhodná počáteční populace nebo se má vytvořit z netlistu, se nastavuje v *cgp.h* parametrem SEEDUJ. Cestu k netlistu značí parametr BLIF.

4.10 Příslušnost jednotlivých hradel k blokům

Poté, co se nám vygeneruje pomocí CGP požadovaný obvod a vytvoří se soubor s chromozomem, můžeme si z vygenerovaného chromozomu nechat určit rozdělení hradel do jednotlivých bloků. Program načte chromozom a vytiskne postupně informace o všech hradlech po sloupcích ve formátu [číslo uzlu]blok. Blokem se zde myslí číslo bloku, ke kterému hradlo náleží. V případě že je číslo bloku rovno nule, znamená to, že hradlo se nepodílí na funkčnosti obvodu.

Příslušnost se zjišťuje po jednotlivých blocích od prvního po poslední blok. Zjistí se, jaká hradla jsou použita v předchozích blocích a zároveň jaká hradla využívá daný blok. Ze seznamu použitých hradel v aktuálním bloku se následně vyjmou ta hradla, která jsou využívána v předchozích blocích. Toto vyjmutí hradla ze seznamu značí, že hradlo ovlivňuje některý z předchozích bloků. Pokud by bylo hradlo přiřazeno k aktuálnímu bloku a tento blok by byl neaktivní, znemožnilo by to funkci bloků aktivních. Následně se už jen uloží informace o příslušnosti hradla a po průchodu všech bloků se tyto informace vytisknou.

Zapnout výpis příslušnosti hradel k blokům lze pomocí PRINT_USED_IN_CUTS v *cgp.h*. Cestu k souboru s chromozomem nastavíme pomocí CHR.

Kapitola 5

Experimenty

Byla provedena sada experimentů na 4-bitové sčítačce, 9-bitové majoritě a 9-bitové blokové majoritě. Pro každý obvod byly vykonány experimenty pro návrh obvodu s jedním a dvěma řezy. Dále pak pro každou variantu dělení obvodu (jeden nebo dva řezy) se experimenty rozvětvily na 5%, 10% a 15% aproximační chybu. Aproximační chybou se myslí, o kolik procent se zhorší funkčnost obvodu při odebrání posledního bloku. Mějme například plně funkční obvod skládající se z tří bloků (dva řezy) a chybou 10%. Po odebrání posledního bloku se zhorší funkčnost obvodu ze 100% na 90% a při odebrání dalšího bloku na 80%.

U všech pokusů byla hodnota L-back nastavena na maximum. Populace sestávala z pěti jedinců. Maximální počet genů, které lze při jedné mutaci zmutovat, byl nastaven na tři. Počet běhů byl 50. K nalezení řešení byla použita hradla z tabulky 5.1.

Použitá hradla	
Číslo	Funkce
0	$in1$
1	$in1 \& in2$
2	$in1 \mid in2$
3	$in1 \oplus in2$
4	$\neg in1$
5	$\neg in2$
6	$in1 \& \neg in2$
7	$\neg(in1 \& in2)$
8	$\neg(in1 \mid in2)$
9	$\neg(in1 \oplus in2)$

Tabulka 5.1: Tabulka použitých hradel

Vzhledem k provedení vyššího počtu experimentů budou v této kapitole popsány jen vybrané experimenty. Zbývající experimenty jsou pro doplnění v příloze A.

Experimenty byly provedeny i pro 9-bitovou sudou paritu, ale vzhledem k charakteru obvodu a malému počtu hradel se nepodařilo dosáhnout požadovaných výsledků. Proto se paritou již nebudeme dále zabývat.

5.1 4 bitová sčítačka

U 4-bitové sčítačky byly experimenty prováděny s náhodně vygenerovanou populací a s populací inicializovanou konvenčním řešením. Konvenční řešení se skládá z jedné poloviční 1-bitové sčítačky a z 1-bitových úplných sčítaček a obsahovala 17 použitých hradel. Rozdíl ve výsledku mezi použitím a nepoužitím inicializace konvenčním řešením byl nepatrný, a proto následně prezentované obvody jsou výběrem nejlepšího z obou těchto variant.

U všech experimentů pro 4-bitovou sčítačku byla nastavena velikost matice na 9 sloupců a 5 řádků. Počet generací byl 10000000.

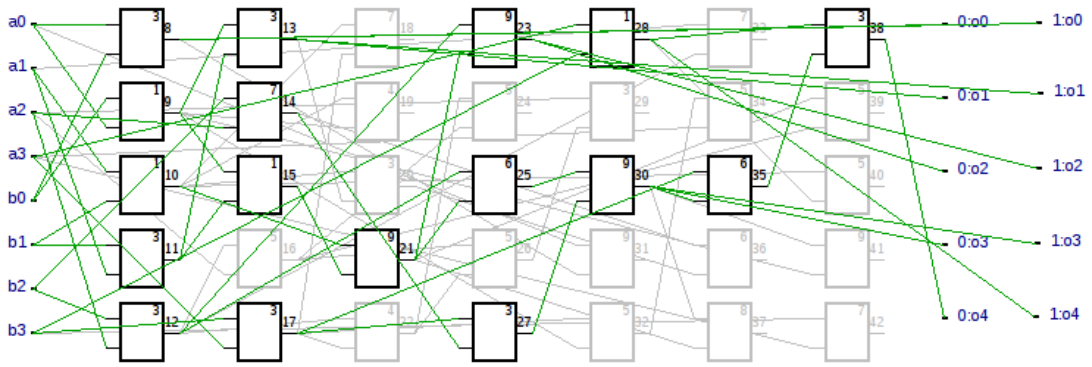
Pro variantu s jedním řezem byly provedeny experimenty s řezem v třetím, čtvrtém a pátém sloupci. Nejlepších výsledků bylo dosaženo pro řez v pátém sloupci a tato varianta bude dále prezentována.

Pro variantu se dvěma řezy byly provedeny experimenty s řezy v druhém a pátém, třetím a pátém, třetím a šestém sloupci. Nejlepších výsledků bylo dosaženo u varianty s řezy v třetím a šestém sloupci a tato varianta bude dále prezentována.

5.1.1 Varianta: 1 řez, aproximační chyba 5%

Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s jedním řezem v pátém sloupci a funkčností obvodu v tomto řezu 95%. To znamená fitness v řezu 1216 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1258,5, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s inicializovanou populací a je zobrazen na obrázku 5.1.

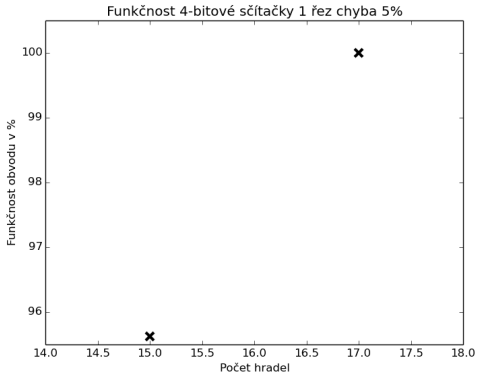


Obrázek 5.1: 4-bitová sčítačka s řezem v pátém sloupci a aproximační chybou 5%

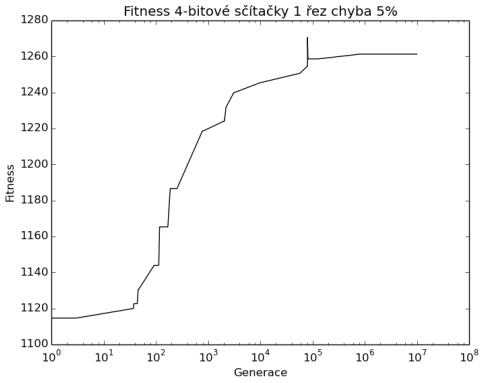
Na grafu 5.2 je vidět, že s přidáním počtu hradel roste funkčnost sčítačky. Sčítačka s patnácti hradly má funkčnost 95,625% a sčítačka se sedmnácti hradly je plně funkční.

Průběh celkové fitness lze sledovat na grafu 5.3. Výkyv fitness u generace 10^5 je zapříčiněn nalezením obvodu s vyšší fitness než udává hranice. Od této chvíle se CGP snažilo minimalizovat počet použitých bloků, což zapříčinilo pokles fitness blíže k hranici a minimalizaci počtu použitých bloků.

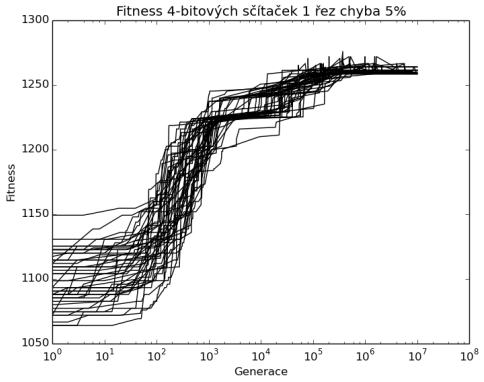
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s aproximační chybou 5% a jedním řezem. Nalezené řešení je zajímavé tím, že nevzrostl počet použitých bloků plně funkčního obvodu oproti analytickému řešení. Pro doplnění jsou v grafu 5.4 vykresleny průběhy všech fitness v tomto experimentu s inicializovanou počáteční populací.



Obrázek 5.2: Vliv počtu bloků na funkčnost 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 5%



Obrázek 5.3: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec)



Obrázek 5.4: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

5.1.2 Varianta: 2 řezy, aproximační chyba 5%

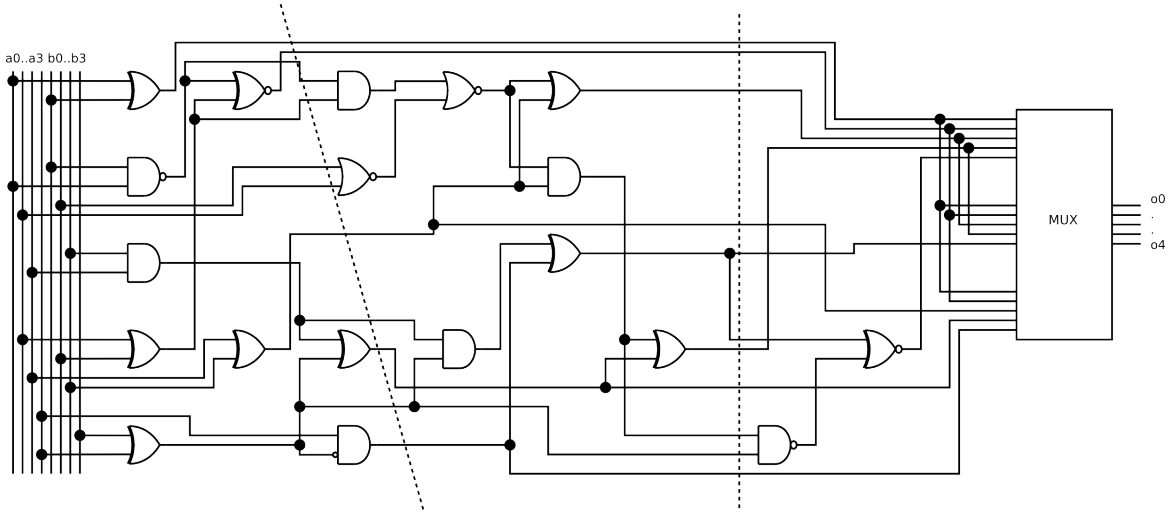
Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s dvěma řezy v třetím a šestém sloupci a aproximační chybou 5%. To znamená fitness v prvním řezu 1152,v druhém řezu 1216 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1237, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s inicializovanou populací a je zobrazen na obrázku 5.5. Přerušované čáry značí dělení obvodu na jednotlivé bloky.

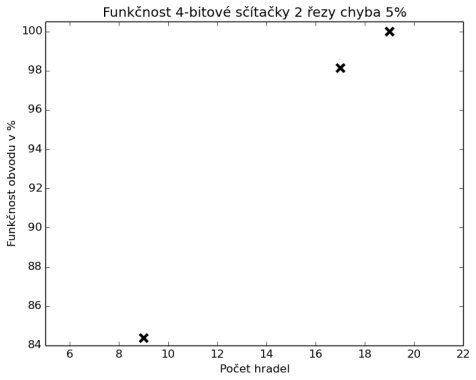
I se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu 5.6. Nalezený obvod sice nemá funkčnost v prvním řezu 90%, ale za to v druhém řezu je funkčnost vyšší.

Průběh celkové fitness lze sledovat na grafu 5.7.

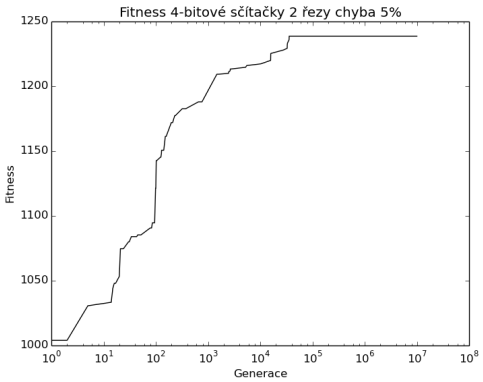
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s aproximační chybou 5% a dvěma řezy. Nalezené řešení při plné funkčnosti obsahuje o dvě hradla navíc oproti konvenčnímu řešení. Pro doplnění jsou v grafu 5.8 vykresleny průběhy všech fitness v tomto experimentu s inicializovanou



Obrázek 5.5: 4-bitová sčítačka s dvěma řezy a aproximační chybou 5%

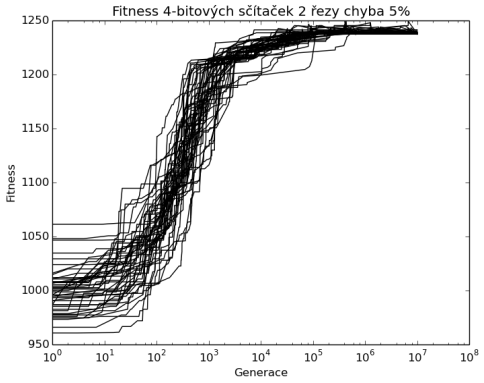


Obrázek 5.6: Vliv počtu bloků na funkčnost 4-bitové sčítačky s dvěma řezy a aproximační chybou 5%



Obrázek 5.7: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec)

počáteční populací



Obrázek 5.8: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

5.2 9 bitová bloková majorita

U 9-bitové blokové majority byly experimenty prováděny s populací inicializovanou konvenčním řešením i náhodně generovanou populací. Konvenční řešení se skládá ze čtyř bloků, které určí majoritu ze tří bitů. Nejprve je spočítána majorita pro vstupní trojice bitů a poté majorita pro mezivýsledky. Konvenční řešení obsahuje 20 použitých hradel. Rozdíl mezi použitím a nepoužitím inicializované počáteční populace byl nepatrný, a proto následně prezentované obvody jsou obvody s náhodnou počáteční populací.

U všech experimentů pro 9-bitovou blokovou majoritu byla nastavena velikost matice na 9 sloupců a 5 řádků. Počet generací byl 10000000.

Pro variantu s jedním řezem byly provedeny experimenty s řezem v třetím, čtvrtém, pátém a šestém sloupci. Nejlepších výsledků bylo dosaženo pro řez v šestém sloupci a tato varianta bude dále prezentována.

Pro variantu se dvěma řezy byly provedeny experimenty s řezy v druhém a pátém, třetím a pátém, čtvrtém a šestém sloupci. Nejlepších výsledků bylo dosaženo u varianty s řezy v čtvrtém a šestém sloupci a tato varianta bude dále prezentována.

5.2.1 Varianta: 1 řez, aproximační chyba 5%

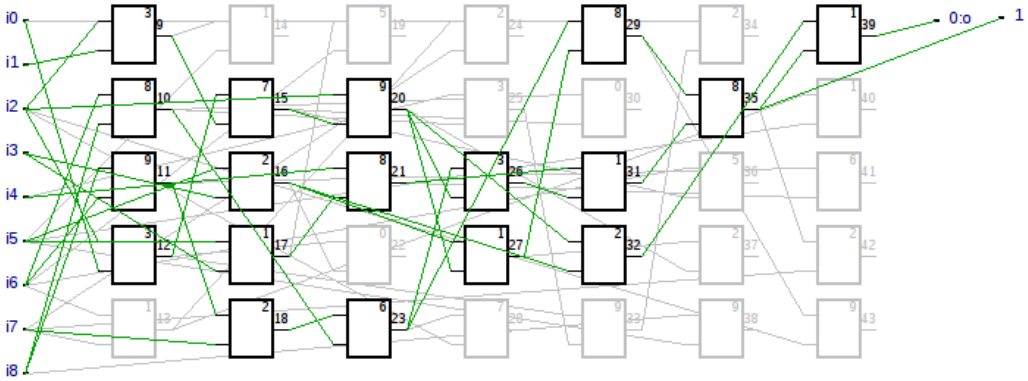
Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s jedním řezem v šestém sloupci a funkčností obvodu v tomto řezu 95%. To znamená fitness v řezu 486 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 503,33, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je na obrázku 5.9.

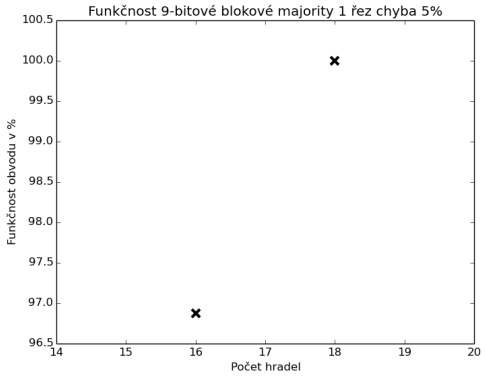
Na grafu 5.10 je vidět, že s přidáním počtu hradel roste funkčnost blokové majority. Bloková majorita s šestnácti hradly má funkčnost 96,875% a bloková majorita s osmnácti hradly je plně funkční.

Průběh celkové fitness lze sledovat na grafu 5.11.

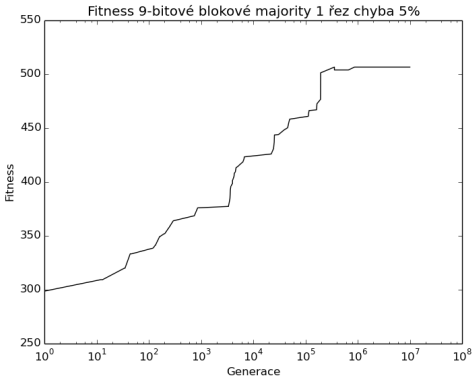
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 5% a jedním řezem. Nalezené řešení je zajímavé tím, že počet použitých hradel plně funkčního obvodu oproti konvenčnímu řešení má o dvě hradla méně. Pro doplnění jsou v grafu 5.12 vykresleny průběhy všech fitness v tomto experimentu.



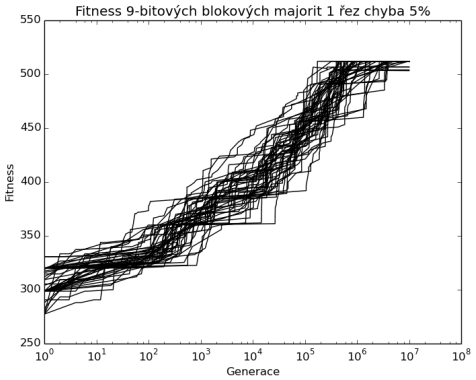
Obrázek 5.9: 9-bitová bloková majoria s řezem v šestém sloupci a aproximační chybou 5%



Obrázek 5.10: Vliv počtu bloků na funkčnost 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 5%



Obrázek 5.11: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec)

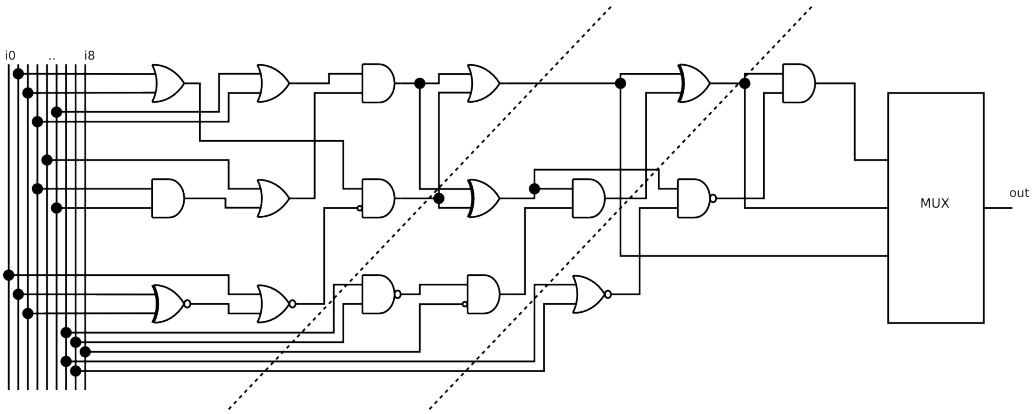


Obrázek 5.12: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

5.2.2 Varianta: 2 řezy, aproximační chyba 10%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s dvěma řezy v čtvrtém a šestém sloupci a aproximační chybou 10%. To znamená fitness v prvním řezu 410,v druhém řezu 461 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 478, která byla při tomto experimentu použita.

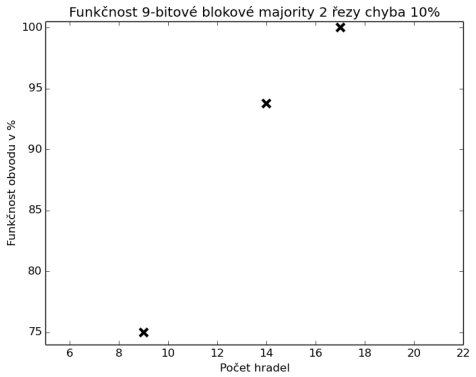
Nejlepší vygenerovaný obvod je zobrazen na obrázku 5.13. Jednotlivé bloky jsou odděleny přerušovanou čarou.



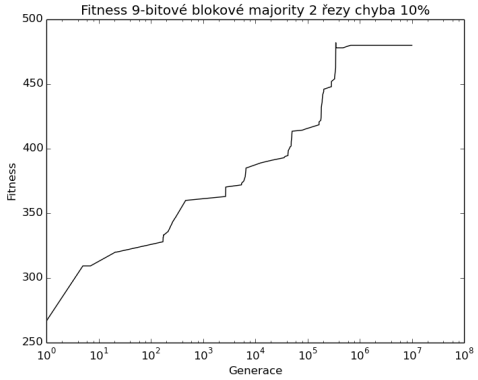
Obrázek 5.13: 9-bitová bloková majorita s dvěma řezy a aproximační chybou 10%

I se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět v grafu 5.14. Nalezený obvod sice nemá funkčnost v prvním řezu 80%, ale za to v druhém řezu je funkčnost vyšší.

Průběh celkové fitness lze sledovat na grafu 5.15.

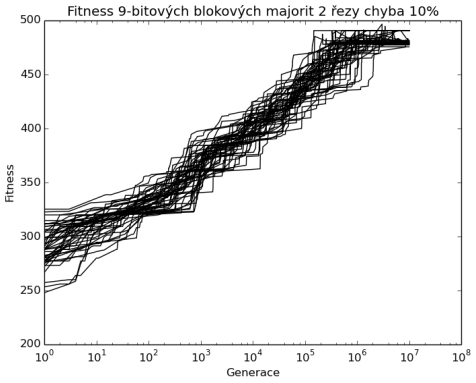


Obrázek 5.14: Vliv počtu bloků na funkčnost 9-bitové blokové majority s dvěma řezy a aproximační chybou 10%



Obrázek 5.15: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec)

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 10% a dvěma řezy. Nalezené řešení při plné funkčnosti obsahuje o tři použité hradla méně oproti konvenčnímu řešení. Pro doplnění jsou na grafu 5.16 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek 5.16: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

5.3 9 bitová majorita

U 9-bitové majority byly experimenty prováděny s náhodně generovanou počáteční populací.

U všech experimentů pro 9-bitovou majoritu byla nastavena velikost matice na 11 sloupců a 7 řádků. Počet generací byl 20000000.

Pro variantu s jedním řezem byly provedeny experimenty s řezem v pátém, šestém a sedmém sloupci. Nejlepších výsledků bylo dosaženo pro řez v sedmém sloupci a tato varianta bude dále prezentována.

Pro variantu se dvěma řezy byly provedeny experimenty s řezy ve čtvrtém a osmém, pátém a osmém sloupci. Nejlepších výsledků bylo dosaženo u varianty s řezy v pátém a osmém sloupci a tato varianta bude dále prezentována.

5.3.1 Varianta: 1 řez, aproximační chyba 15%

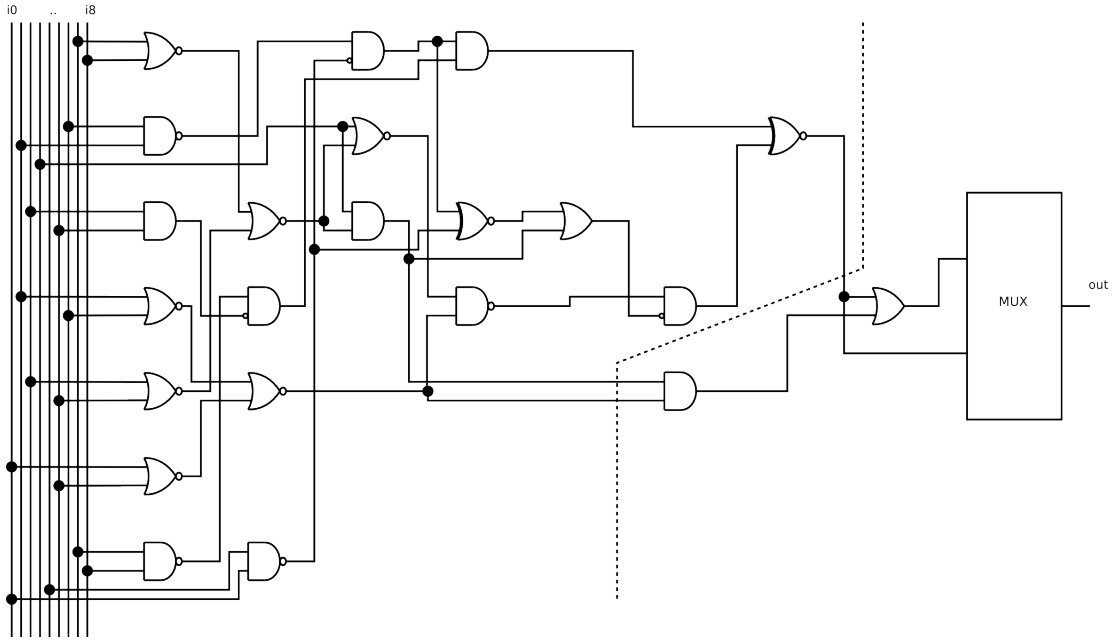
Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s jedním řezem v sedmém sloupci a funkčností obvodu v tomto řezu 85%. To znamená fitness v řezu 435 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 486,33, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je na obrázku 5.17. Přerušovaná čára značí rozdělení obvodu na bloky.

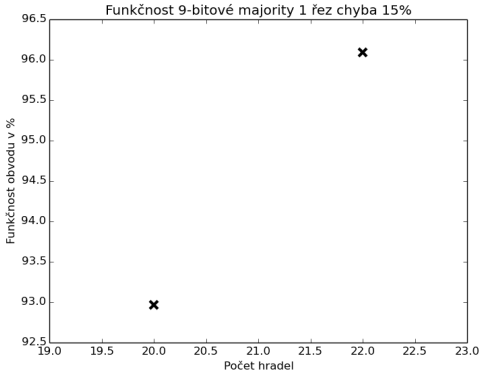
Na grafu 5.18 je vidět, že s přidáním počtu hradel roste funkčnost majority. Majorita s 20 hradly má funkčnost 92,969% a majorita s 22 hradly 96,94%.

Průběh celkové fitness lze sledovat na grafu 5.19.

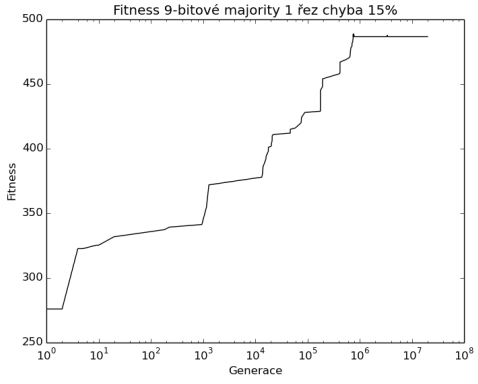
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 15% a jedním řezem. Při snížení chyby o dalších 5% opět poklesl i potřebný počet hradel. Pro doplnění jsou v grafu 5.20 vykresleny průběhy všech fitness v tomto experimentu.



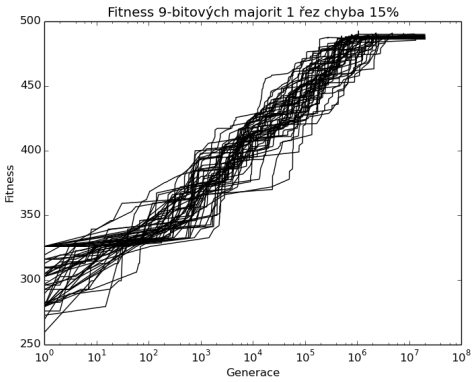
Obrázek 5.17: 9-bitová bloková majorita s řezem v sedmém sloupci a aproximační chybou 15%



Obrázek 5.18: Vliv počtu bloků na funkčnost 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 15%



Obrázek 5.19: Rostoucí fitness 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec)

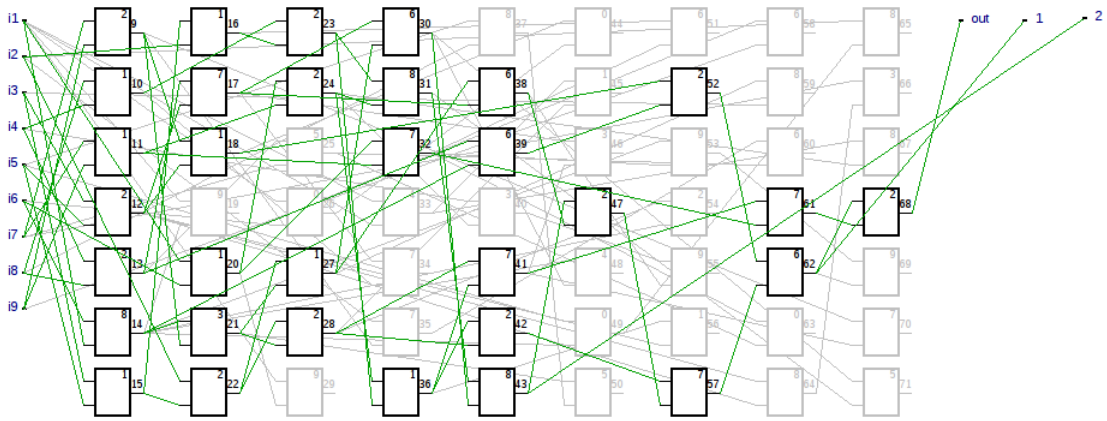


Obrázek 5.20: Rostoucí fitness 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)

5.3.2 Varianta: 2 řezy, aproximační chyba 5%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s dvěma řezy v pátém a osmém sloupci a aproximační chybou 5%. To znamená fitness v prvním řezu 461,v druhém řezu 486 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 494,83, která byla při tomto experimentu použita.

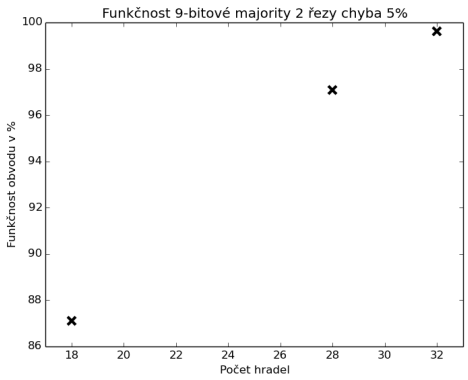
Nejlepší vygenerovaný obvod je zobrazen na obrázku 5.21.



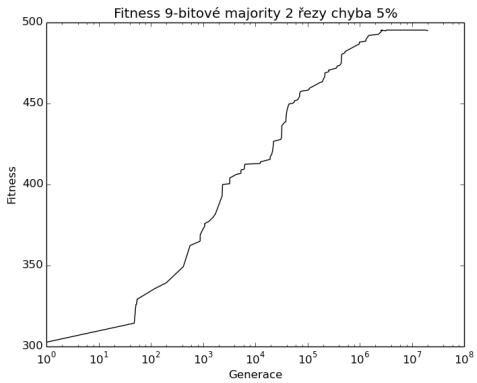
Obrázek 5.21: 9-bitová majorita s dvěma řezy a aproximační chybou 5%

I se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu 5.22.

Průběh celkové fitness lze sledovat v grafu 5.23.

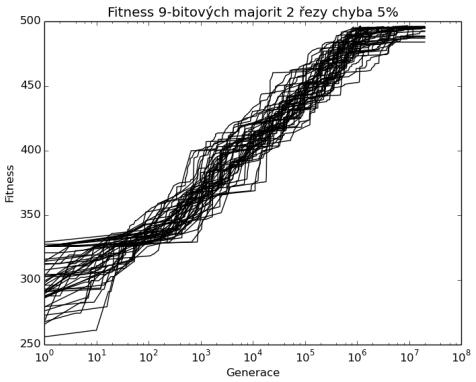


Obrázek 5.22: Vliv počtu bloků na funkčnost 9-bitové majority s dvěma řezy a aproximační chybou 5%



Obrázek 5.23: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec)

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 5% a dvěma řezy. Pro doplnění jsou na grafu 5.24 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek 5.24: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

Kapitola 6

Závěr

V práci byla představena metoda pro generování obvodů umožňujících postupnou aproximaci pomocí CGP. Byly zavedeny a vysvětleny některé nové pojmy. Musela se navrhnout fitness funkce pro účely dělení obvodu na bloky. Metoda byla experimentálně ověřena na třech obvodech s různými parametry. Ukázalo se, že optimální pokles funkčnosti obvodu při odebrání jednoho bloku je mezi 5%-10%. Za těchto podmínek CGP generuje relativně kvalitní řešení. Při poklesu funkčnosti o méně než 5% již nejsme schopni generovat funkční obvody. Naopak při navýšení funkčnosti nad 10% se špatně dosahuje plné funkčnosti za použití všech bloků (pro zvolené obvody). Při dalším vývoji popsané metody by tedy bylo vhodné se pokusit navrhnout vhodnější výpočet fitness funkce. Zajímavých výsledků by taktéž mohlo být dosaženo při generování obvodu po jednotlivých blocích a ne jako celku najednou.

Je nutné poznamenat, že volaná fitness funkce představuje zjednodušení oproti případné reálné implementaci, která by vyžadovala zohlednění různých typů chyb a dalších parametrů obvodu. Toto zjednodušení jsme si mohli dovolit, protože se jedná o první studii na toto téma.

Literatura

[1] Hajná, K.: *Kartézské genetické programování s dynamickou modifikací parametrů*. FIT VUT v Brně, 2015, bakalářská práce.

[2] Jain, S.; Venkataramani, S.; Raghunathan, A.: Approximation through Logic Isolation for the Design of Quality Configurable Circuits. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, ISBN 978-3-9815370-6-2.

[3] Mittal, S.: A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.*, ročník 48, č. 4, 2016: s. 62:1–62:33.

[4] Sekanina, L.: *Biologií inspirované počítače*. FIT VUT v Brně, 2015, výukové materiály.

[5] Sekanina, L.; Harding, L. S.; Banzhaf, W.; aj.: *Image Processing and CGP*. Natural Computing Series, Springer Verlag, 2011, ISBN 978-3-642-17309-7, s. 181–215.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9771

[6] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9123

[7] Vašíček, Z.: *Biologií inspirované počítače - kartézské genetické programování [online]*. FIT VUT v Brně, [cit. 2016-1-28].
URL http://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/

Přílohy

Seznam příloh

Příloha A

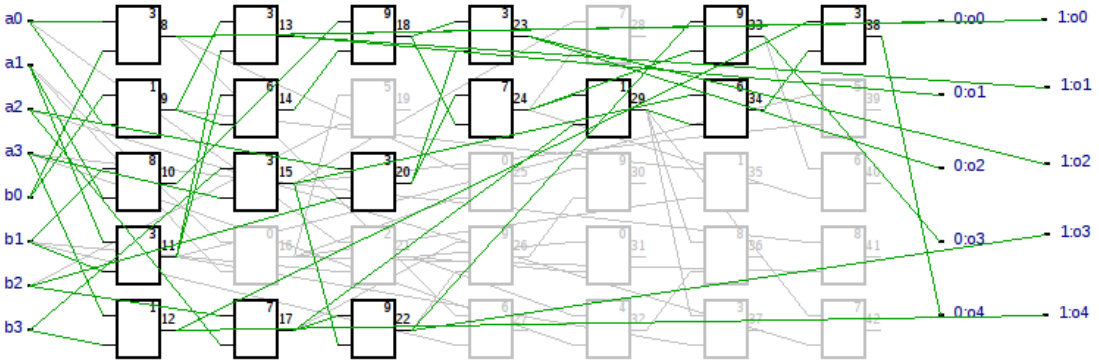
Další experimenty

A.1 4 bitová sčítačka

A.1.1 Varianta: 1 řez, aproximační chyba 10%

Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s jedním řezem v pátém sloupci a funkčností obvodu v tomto řezu 90%. To znamená fitness v řezu 1152 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1237, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s inicializovanou počáteční populací a je zobrazen na obrázku A.1.

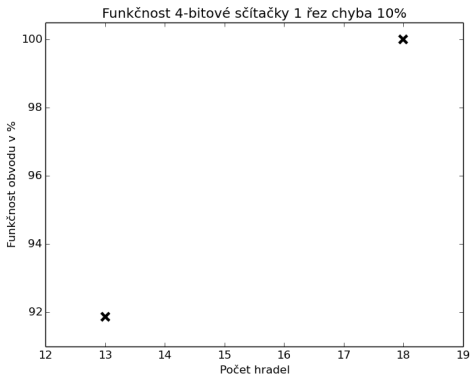


Obrázek A.1: 4-bitová sčítačka s řezem v pátém sloupci a aproximačníchybou 10%

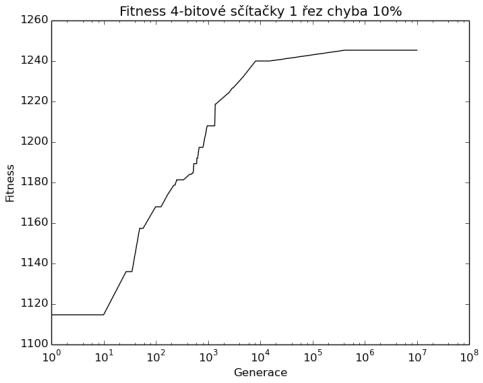
Na grafu A.2 opět je vidět, že s přidáním počtu hradel roste funkčnost sčítačky. Sčítačka s třinácti hradly má funkčnost 91,875% a sčítačka se osmnácti hradly je plně funkční.

Průběh celkové fitness lze sledovat na grafu A.3.

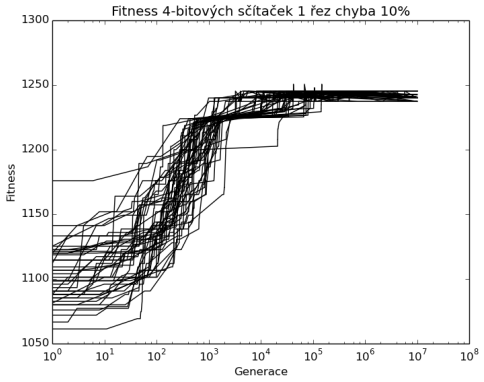
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s aproximační chybou 10% a jedním řezem. Pro doplnění jsou na grafu A.4 vykresleny průběhy všech fitness v tomto experimentu s inicializovanou počáteční populací.



Obrázek A.2: Vliv počtu bloků na funkčnost 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 10%



Obrázek A.3: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec)



Obrázek A.4: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

A.1.2 Varianta: 1 řez, aproximační chyba 15%

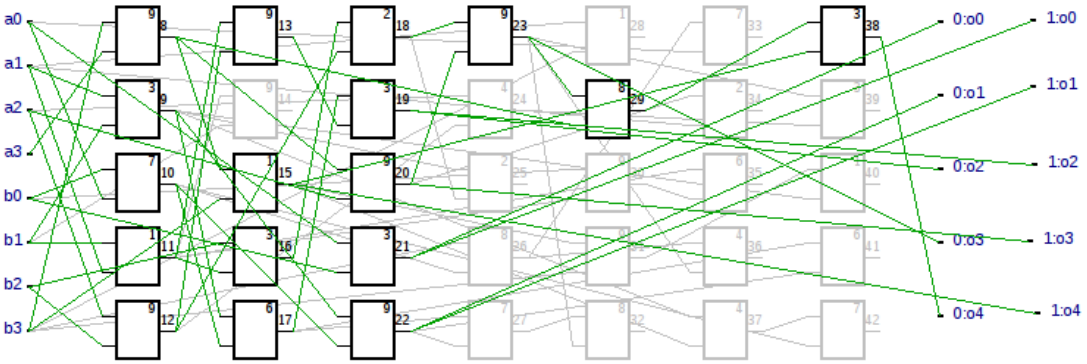
Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s jedním řezem v pátém sloupci a funkčností obvodu v tomto řezu 85%. To znamená fitness v řezu 1088 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1216, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s náhodně vygenerovanou počáteční populací a je zobrazen na obrázku A.5.

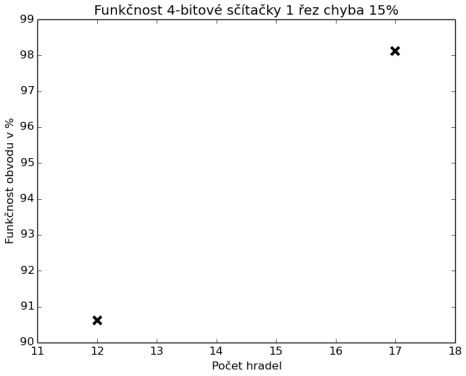
Na grafu A.6 je vidět, že s přidáním počtu hradel opět roste funkčnost sčítačky. Sčítačka s dvanácti hradly má funkčnost 90,625% a sčítačka se sedmnácti hradly 98,125%. Je zde vidět, že se CGP nepodařilo nalézt obvod s úplnou funkčností, ale funkčnost v řezu tím pádem vzrostla.

Průběh celkové fitness lze sledovat na grafu A.7. Opět lze vidět drobné zakolísání fitness po překročení hranice jako u 5.1.1.

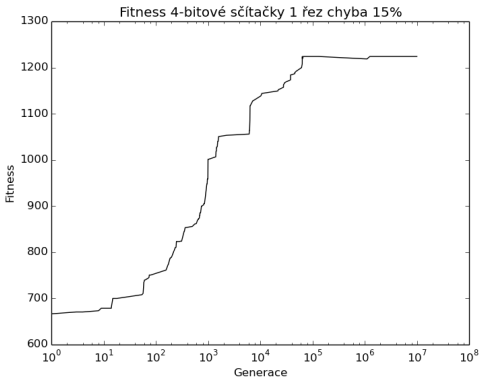
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s aproximační chybou 15% a jedním řezem. Nepodařilo



Obrázek A.5: 4-bitová sčítačka s řezem v pátém sloupci a aproximační chybou 15%

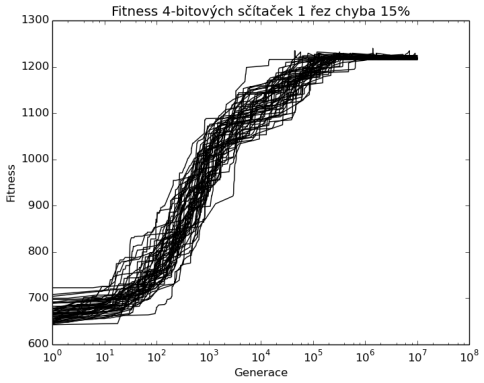


Obrázek A.6: Vliv počtu bloků na funkčnost 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 15%



Obrázek A.7: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec)

se ovšem nalézt obvod který by byl schopen generovat výsledky s plnou funkčností. Pro doplnění jsou na grafu A.8 vykresleny průběhy všech fitness v tomto experimentu s náhodně generovanou počáteční populací.

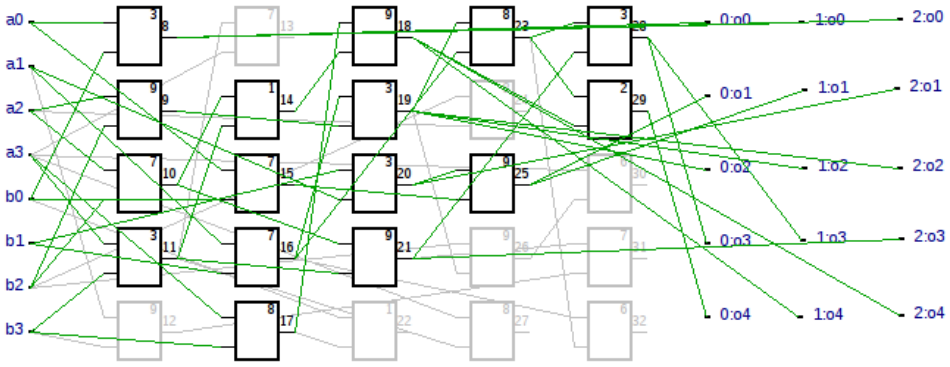


Obrázek A.8: Rostoucí fitness 4-bitové sčítačky s řezem v pátém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)

A.1.3 Varianta: 2 řezy, aproximační chyba 10%

Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s dvěma řezy v třetím a šestém sloupci a aproximační chybou 10%. To znamená fitness v prvním řezu 1024 ,v druhém řezu 1152 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1194,5, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s náhodně generovanou počáteční populací a je zobrazen na obrázku A.9. Na obvodu je vidět použití redukce, kdy původní matice se sedmi sloupci byla redukována na 5 sloupců.

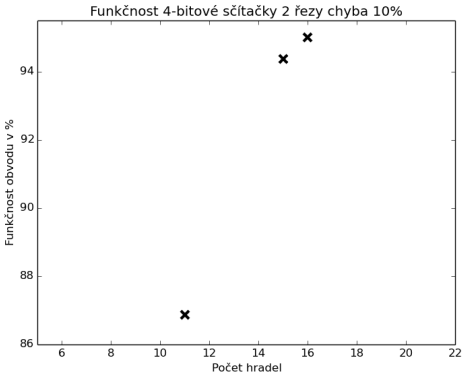


Obrázek A.9: 4-bitová sčítačka s dvěma řezy a aproximační chybou 10%

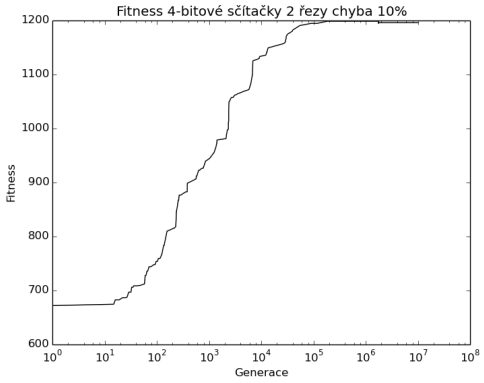
Funkčnost obvodu opět stoupá při růstu počtu hradel, jak je vidět v grafu A.10. Nalezený obvod sice nemá plnou funkčnost při použití 17 hradel, to je ale kompenzováno vyššími funkčnostmi řezů.

Průběh celkové fitness lze sledovat na grafu A.11.

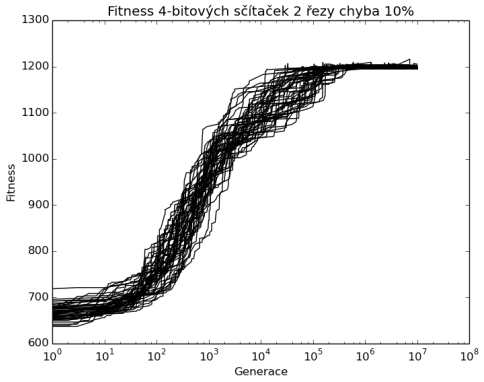
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s aproximační chybou 10% a dvěma řezy. Nepodařilo se ovšem nalézt plně funkční sčítačku. Pro doplnění jsou v grafu A.12 vykresleny průběhy všech fitness v tomto experimentu s náhodně generovanou počáteční populací.



Obrázek A.10: Vliv počtu bloků na funkčnost 4-bitové sčítačky s dvěma řezy a aproximační chybou 10%



Obrázek A.11: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec)



Obrázek A.12: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

A.1.4 Varianta: 2 řezy, aproximační chyba 15%

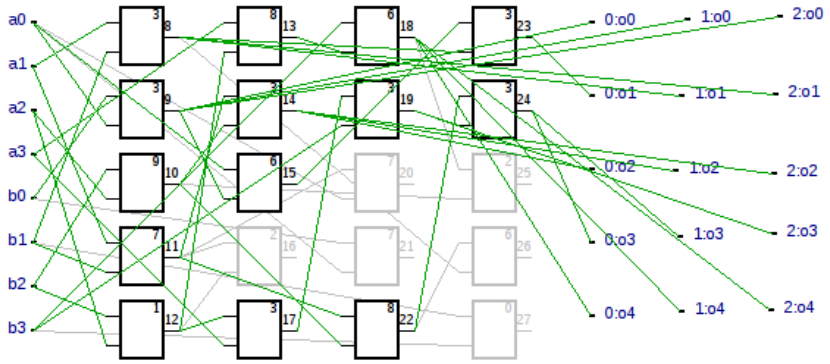
Cílem tohoto experimentu bylo vygenerování obvodu 4-bitové sčítačky s dvěma řezy v třetím a šestém sloupci a aproximační chybou 15%. To znamená fitness v prvním řezu 896,v druhém řezu 1088 a u plně funkční varianty fitness 1280. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 1152, která byla při tomto experimentu použita.

Nejlepší obvod byl vygenerován u varianty s náhodně generovanou počáteční populací a je zobrazen na obrázku A.13. Na obvodu je vidět použití redukce kdy původní matice se sedmi sloupci byla redukována na 4 sloupce.

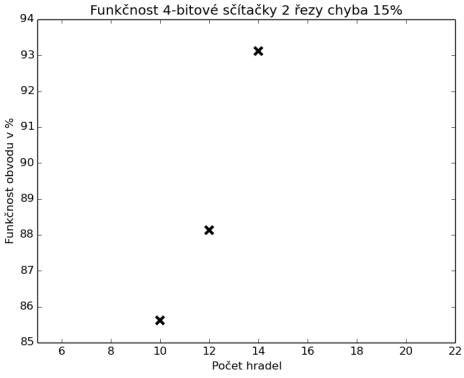
Funkčnost obvodu opět stoupá při růstu počtu bloků, jak je vidět na grafu A.14. Nalezený obvod sice nemá plnou funkčnost při použití 14 bloků, to je ale kompenzováno vyššími funkčnostmi řezů.

Průběh celkové fitness lze sledovat na grafu A.15.

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 4-bitovou sčítačku s chybou 15% a dvěma řezy. Nepodařilo se ovšem naléz plně funkční sčítačku. Pro doplnění jsou na grafu A.16 vykresleny průběhy všech fitness v tomto experimentu s náhodně generovanou počáteční populací.

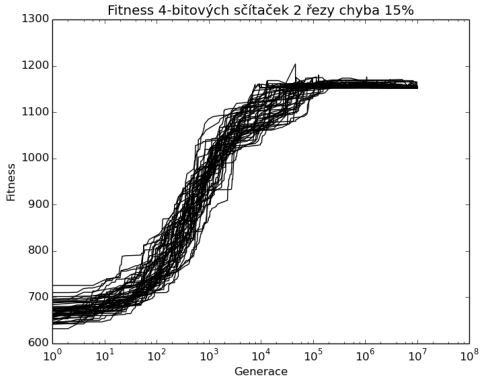


Obrázek A.13: 4-bitová sčítačka s dvěma řezy a aproximační chybou 15%



Obrázek A.14: Vliv počtu bloků na funkčnost 4-bitové sčítačky s dvěma řezy a aproximační chybou 15%

Obrázek A.15: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a aproximační chybou 15% (uveden je nejlepší jedinec)



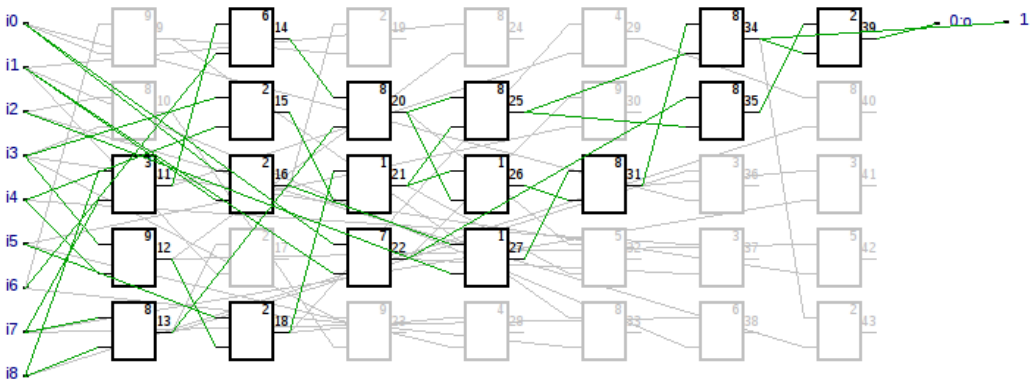
Obrázek A.16: Rostoucí fitness 4-bitové sčítačky s dvěma řezy a chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)

A.2 9 bitová bloková majorita

A.2.1 Varianta: 1 řez, aproximační chyba 10%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s jedním řezem v šestém sloupci a funkčností obvodu v tomto řezu 90%. To znamená fitness v řezu 461 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 495, která byla při tomto experimentu použita.

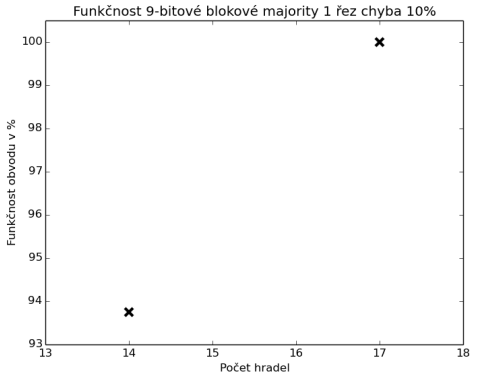
Nejlepší vygenerovaný obvod je na obrázku A.17.



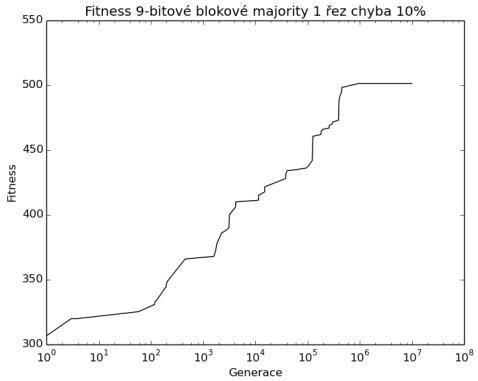
Obrázek A.17: 9-bitová bloková majoria s řezem v šestém sloupci a aproximační chybou 10%

Na grafu A.18 je vidět, že s přidáním počtu hradel roste funkčnost blokové majority. Bloková majorita s čtrnácti hradly má funkčnost 93,75% a bloková majorita se sedmnácti hradly je plně funkční.

Průběh celkové fitness lze sledovat na grafu A.19.



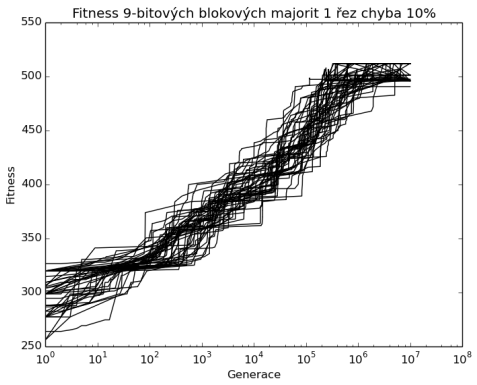
Obrázek A.18: Vliv počtu bloků na funkčnost 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 10%



Obrázek A.19: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec)

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 10% a jedním řezem. Nalezené řešení je zajímavé tím, že počet použitých hradel plně funkčního obvodu

oproti konvenčnímu řešení má o tři hradla méně. Pro doplnění jsou v grafu A.20 vykresleny průběhy všech fitness v tomto experimentu.

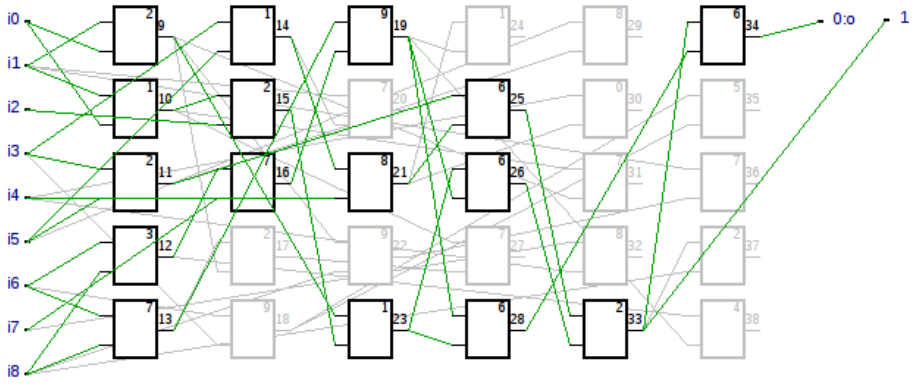


Obrázek A.20: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

A.2.2 Varianta: 1 řez, aproximační chyba 15%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s jedním řezem v šestém sloupci a funkčností obvodu v tomto řezu 85%. To znamená fitness v řezu 435 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 486,33, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je na obrázku A.21.

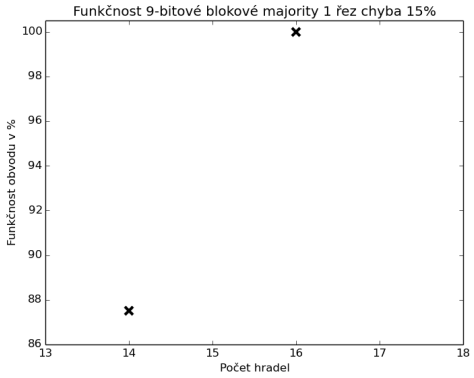


Obrázek A.21: 9-bitová bloková majoria s řezem v šestém sloupci a aproximační chybou 15%

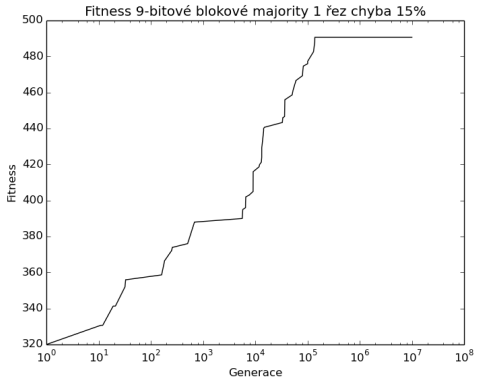
Na grafu A.22 je vidět, že s přidáním počtu hradel roste funkčnost blokové majority. Bloková majorita s čtrnácti hradly má funkčnost 87,5% a bloková majorita se šestnácti hradly je plně funkční.

Průběh celkové fitness lze sledovat na grafu A.23.

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 15% a jedním řezem. Nalezené řešení je zajímavé tím, že počet použitých hradel plně funkčního obvodu

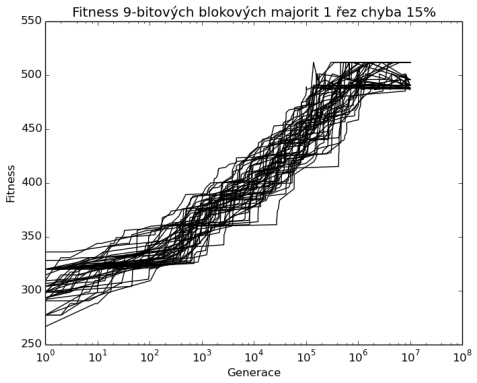


Obrázek A.22: Vliv počtu bloků na funkčnost 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 15%



Obrázek A.23: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec)

oproti konvenčnímu řešení má o čtyři bloky méně. Pro doplnění jsou v grafu A.24 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek A.24: Rostoucí fitness 9-bitové blokové majority s řezem v pátém sloupci a aproximační chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)

A.2.3 Varianta: 2 řezy, aproximační chyba 5%

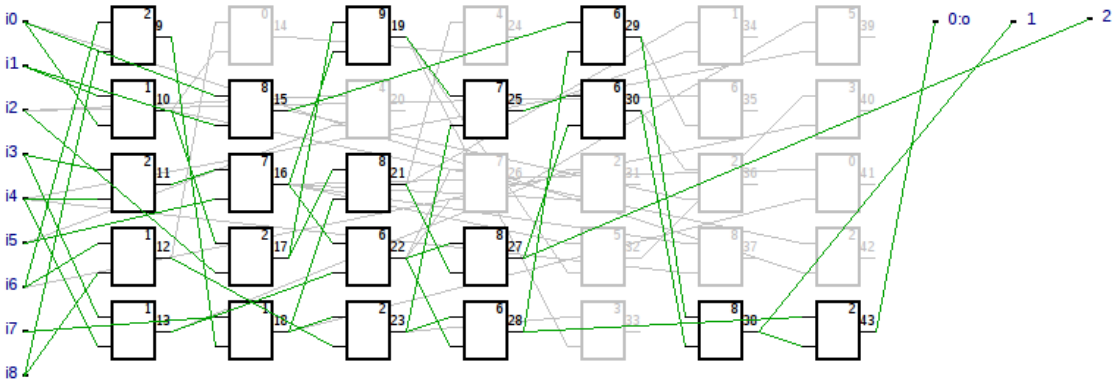
Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s dvěma řezy v čtvrtém a šestém sloupci a aproximační chybou 5%. To znamená fitness v prvním řezu 461,v druhém řezu 486 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 494,83, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je zobrazen na obrázku A.25.

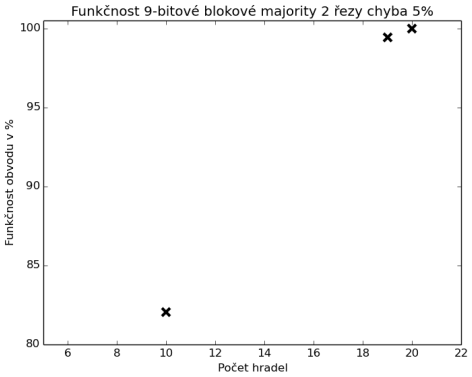
I se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu A.26. Nalezený obvod sice nemá funkčnost v prvním řezu 90%, ale za to v druhém řezu je funkčnost vyšší.

Průběh celkové fitness lze sledovat na grafu A.27.

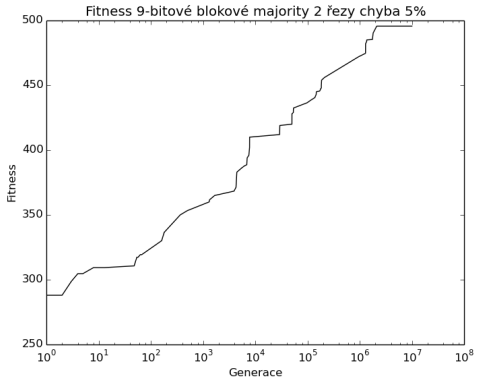
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto ex-



Obrázek A.25: 9-bitová bloková majorita s dvěma řezy a aproximační chybou 5%

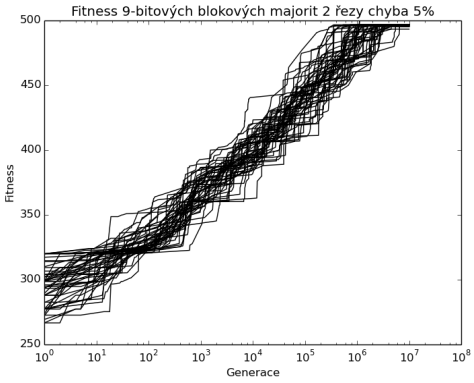


Obrázek A.26: Vliv počtu bloků na funkčnost 9-bitové blokové majority s dvěma řezy a aproximační chybou 5%



Obrázek A.27: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec)

perimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 5% a dvěma řezy. Nalezené řešení při plné funkčnosti obsahuje stejný počet použitých hradel jako konvenční řešení. Pro doplnění jsou na grafu A.28 vykresleny průběhy všech fitness v tomto experimentu.

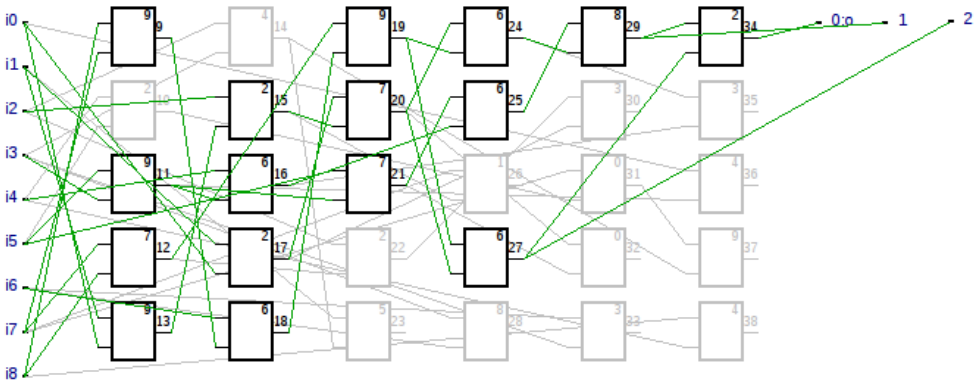


Obrázek A.28: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

A.2.4 Varianta: 2 řezy, aproximační chyba 15%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové blokové majority s dvěma řezy v čtvrtém a šestém sloupci a aproximační chybou 15%. To znamená fitness v prvním řezu 358,v druhém řezu 435 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 460,66, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je zobrazen na obrázku A.29.

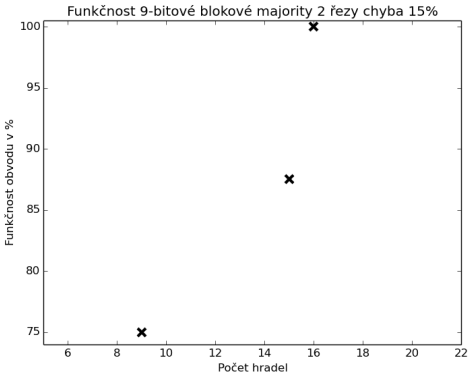


Obrázek A.29: 9-bitová bloková majorita s dvěma řezy a aproximační chybou 15%

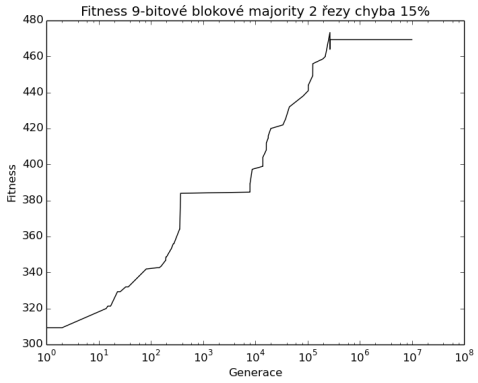
Se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu A.30. Nalezený obvod sice nemá funkčnost v prvním řezu 80%, ale za to v druhém řezu je funkčnost vyšší.

Průběh celkové fitness lze sledovat na grafu A.31. Celková fitness je u tohoto obvodu 469,33, tedy vyšší než je nastavená hranice. To umožnilo navýšení fitness v řezech.

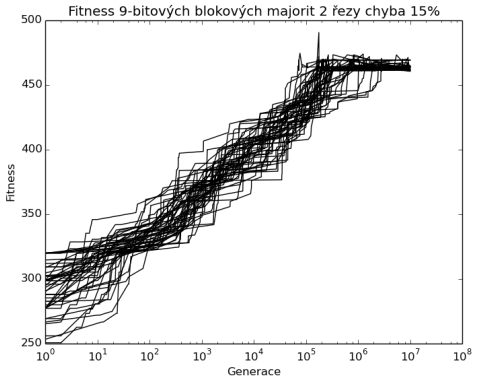
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou blokovou majoritu s aproximační chybou 15% a dvěma řezy. Nalezené řešení při plné funkčnosti obsahuje o čtyři použité hradla méně oproti analytickému řešení. Pro doplnění jsou na grafu A.32 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek A.30: Vliv počtu bloků na funkčnost 9-bitové blokové majority s dvěma řezy a aproximační chybou 15%



Obrázek A.31: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 15% (uveden je nejlepší jedinec)



Obrázek A.32: Rostoucí fitness 9-bitové blokové majority s dvěma řezy a aproximační chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)

A.3 9 bitová majorita

A.3.1 Varianta: 1 řez, aproximační chyba 5%

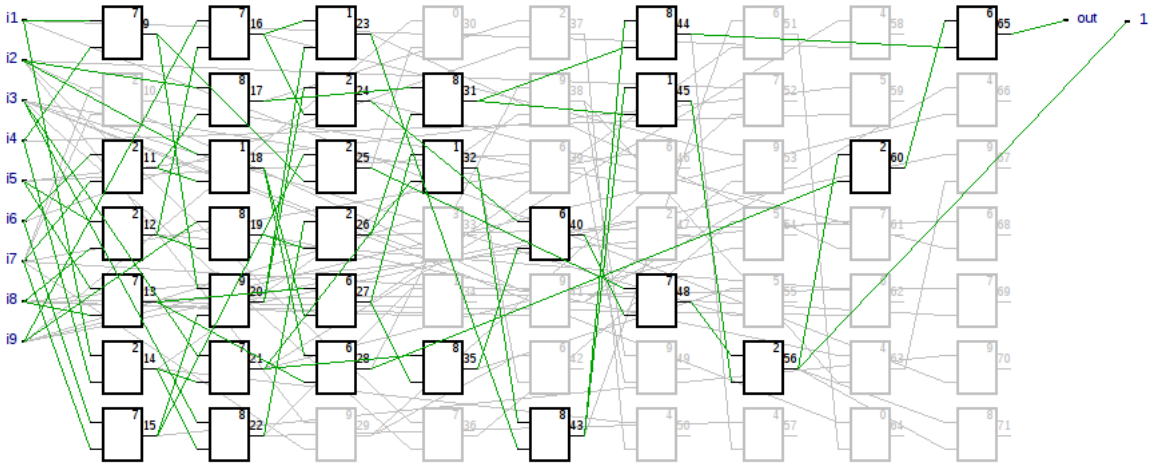
Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s jedním řezem v sedmém sloupci a funkčností obvodu v tomto řezu 95%. To znamená fitness v řezu 486 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 503,33, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je na obrázku A.33.

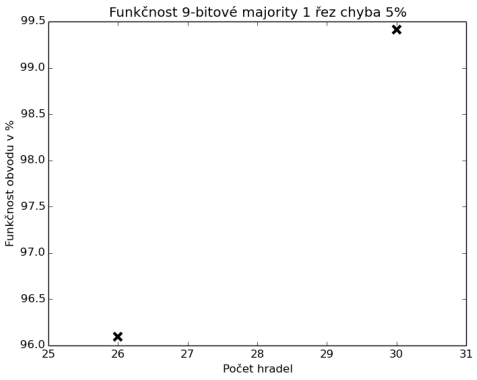
Na grafu A.34 je vidět, že s přidáním počtu hradel roste funkčnost majority. Majorita s 26 hradly má funkčnost 96,094% a majorita s 30 hradly 99,414%.

Průběh celkové fitness lze sledovat na grafu A.35.

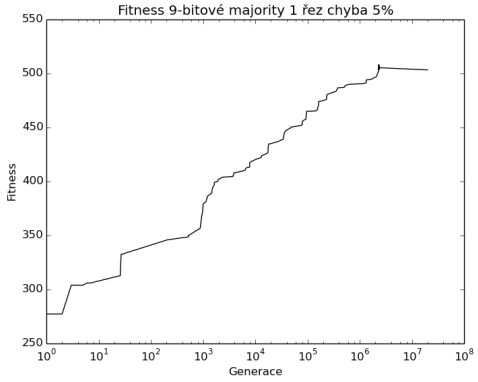
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 5% a jedním řezem. Nalezené řešení je zajímavé tím, že počet použitých hradel plně funkčního obvodu oproti konvenčnímu řešení má o dvě hradla méně. Pro doplnění jsou v grafu A.36 vykresleny průběhy všech



Obrázek A.33: 9-bitová majorita s řezem v sedmém sloupci a aproximační chybou 5%

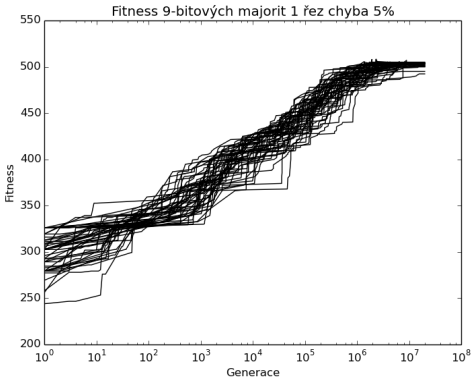


Obrázek A.34: Vliv počtu bloků na funkčnost 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 5%



Obrázek A.35: Rostoucí fitness 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec)

fitness v tomto experimentu. Z grafu je patrné, že se nepodařilo u všech obvodů dosáhnout zadané hranice.

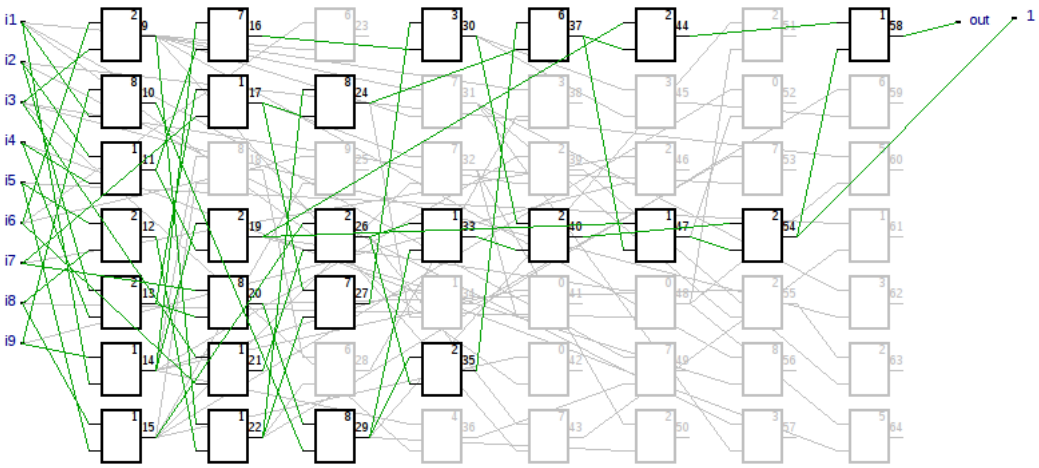


Obrázek A.36: Rostoucí fitness 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 5% (uveden je nejlepší jedinec z 50 různých běhů)

A.3.2 Varianta: 1 řez, aproximační chyba 10%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s jedním řezem v sedmém sloupci a funkčností obvodu v tomto řezu 90%. To znamená fitness v řezu 461 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 495, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je na obrázku A.37.

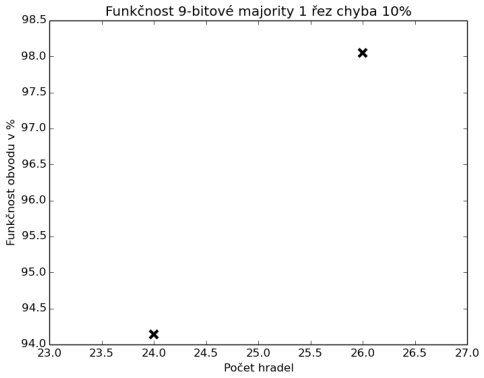


Obrázek A.37: 9-bitová majorita s řezem v sedmém sloupci a aproximační chybou 10%

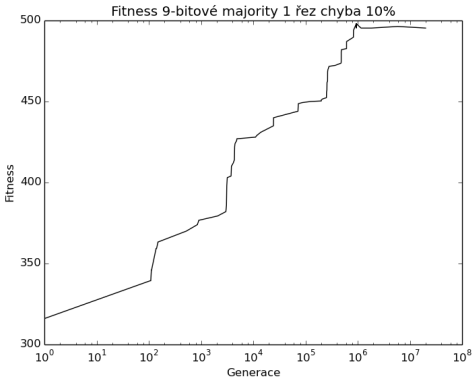
Z grafu A.38 je vidět, že s přidáním počtu hradel roste funkčnost majority. Majorita s 24 bloky má funkčnost 94,14% a bloková majorita se 26 bloky 98,047%.

Průběh celkové fitness lze sledovat na grafu A.39.

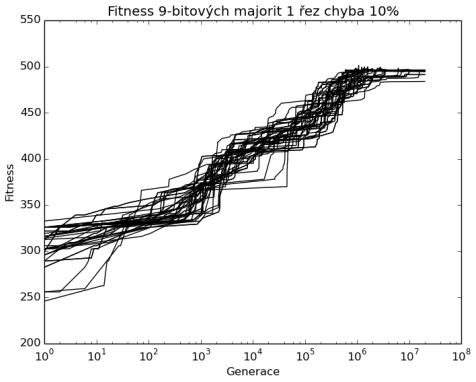
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 10% a jedním řezem. Při snížení chyby z 5% na 10% poklesl i potřebný počet hradel. Pro doplnění jsou v grafu A.40 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek A.38: Vliv počtu bloků na funkčnost 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 10%



Obrázek A.39: Rostoucí fitness 9-bitové majority s řezem v sedmém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec)



Obrázek A.40: Rostoucí fitness 9-bitové majority s řezem v pátém sloupci a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

A.3.3 Varianta: 2 řezy, aproximační chyba 10%

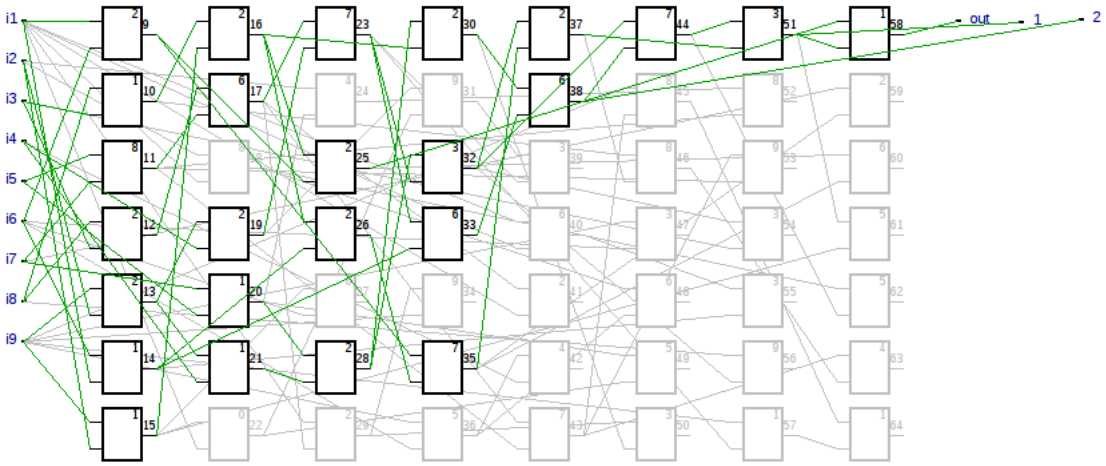
Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s dvěma řezy v pátém a osmém sloupci a aproximační chybou 10%. To znamená fitness v prvním řezu 410,v druhém řezu 461 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 478, která byla při tomto experimentu použita.

Nejlepší vygenerovaný obvod je zobrazen na obrázku A.41.

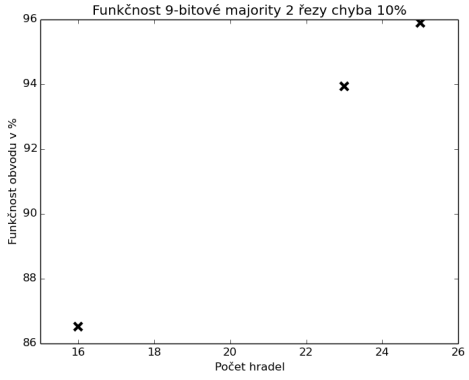
Funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu A.42.

Průběh celkové fitness lze sledovat na grafu A.43.

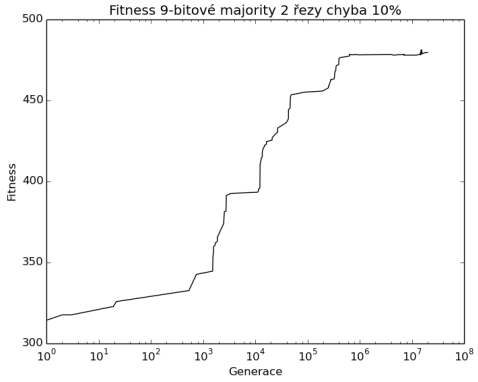
Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 10% a dvěma řezy. Pro doplnění jsou v grafu A.44 vykresleny průběhy všech fitness v tomto experimentu.



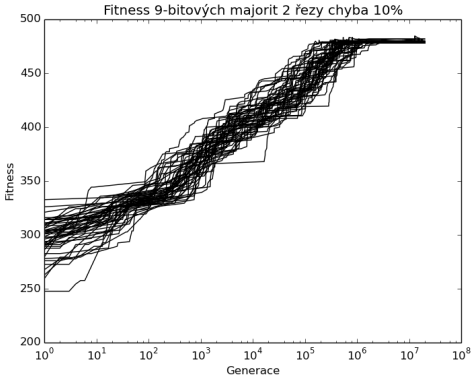
Obrázek A.41: 9-bitová majorita s dvěma řezy a aproximační chybou 10%



Obrázek A.42: Vliv počtu bloků na funkčnost 9-bitové majority s dvěma řezy a aproximační chybou 10%



Obrázek A.43: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec)

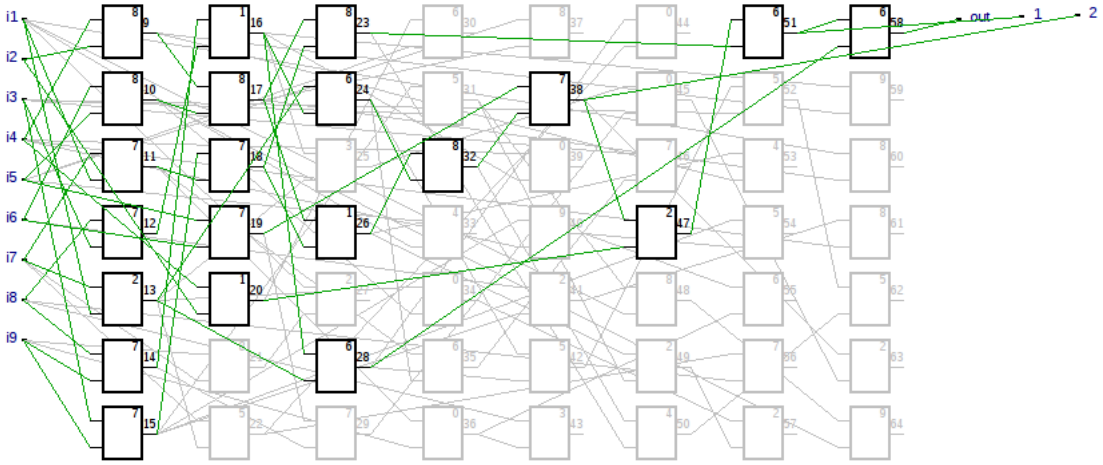


Obrázek A.44: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 10% (uveden je nejlepší jedinec z 50 různých běhů)

A.3.4 Varianta: 2 řezy, aproximační chyba 15%

Cílem tohoto experimentu bylo vygenerování obvodu 9-bitové majority s dvěma řezy v pátém a osmém sloupci a aproximační chybou 15%. To znamená fitness v prvním řezu 358,v druhém řezu 435 a u plně funkční varianty fitness 512. Po dosazení do vzorce pro celkovou fitness dostaneme hranici 460,66, která byla při tomto experimentu použita.

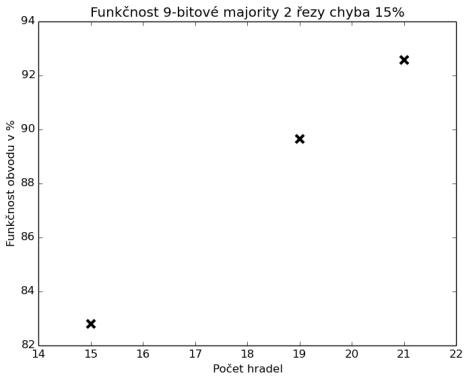
Nejlepší vygenerovaný obvod je zobrazen na obrázku A.45.



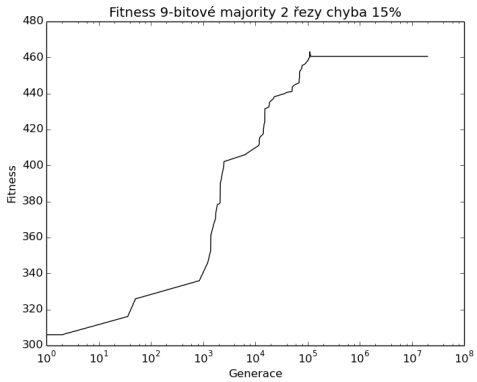
Obrázek A.45: 9-bitová majorita s dvěma řezy a aproximační chybou 15%

Se dvěma řezy funkčnost obvodu stoupá při růstu počtu hradel, jak je vidět na grafu A.46.

Průběh celkové fitness lze sledovat na grafu A.47.

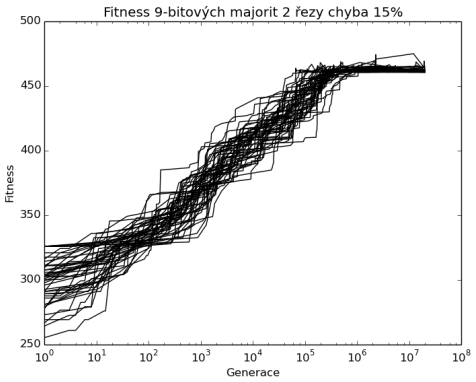


Obrázek A.46: Vliv počtu bloků na funkčnost 9-bitové majority s dvěma řezy a aproximační chybou 15%



Obrázek A.47: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 15% (uveden je nejlepší jedinec)

Dosažené výsledky a nalezený obvod ukazují, že za podmínek nastavených v tomto experimentu lze nalézt 9-bitovou majoritu s aproximační chybou 15% a dvěma řezy. Pro doplnění jsou na grafu A.48 vykresleny průběhy všech fitness v tomto experimentu.



Obrázek A.48: Rostoucí fitness 9-bitové majority s dvěma řezy a aproximační chybou 15% (uveden je nejlepší jedinec z 50 různých běhů)