Sem vložte zadanie Vašej práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práca

Optimalizácia logických obvodov pomocou kartézskeho genetického programovania

Bc. Stanislav Pelák

Vedúci práce: Fišer Petr Ing., Ph.D.

9. mája 2013

Poďakovanie

Týmto by som sa chcel poďakovať Ing. Petrovi Fišerovi, Ph.D. za odborné vedenie diplomovej práce a pomoc pri jej realizácii, a taktiež za výbornú spoluprácu ako po profesionálnej, tak aj ľudskej stránke.

A Renke...za všetko.

Prehlásenie

Týmto prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informáčné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, najmä skutočnosť, že České vysoké učení technické v Praze má právo na uzatvorenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe dňa 9. mája 2013 .....

Odkaz na túto prácu

Pelák, Stanislav. *Optimalizácia logických obvodov pomocou kartézskeho genetického programovania.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

Logical circuits are the main building blocks of all computers and electronics in general. The quality of their design therefore directly affects the quality of the information system as a whole.

Conventional design of logical circuits is based on available knowledge and/or creating a complex logic function out of already existing parts. It is for this reason that this design is limited only to circuits intuitive enough to be designed by a human, which may not always be optimal. As a result, this problem creates the issue of automatization of logic circuits' design in order to avoid these limitations and conventional practices.

Cartesian Genetic Programming serves this purpose. This thesis examines its principles, possibilities, limitations, configuration and its influence on the algorithm's efficiency.

Keywords CGP, logical circuits, optimization, evolution, genetic programming

Abstrakt

Logické obvody sú základným stavebným prvkom všetkých počítačov a elektroniky všeobecne. Kvalita ich návrhu preto priamo ovplyvňuje kvalitu informačného systému ako celku.

Pri konvenčnom návrhu logických obvodov človekom sa vychádza z dostupných znalostí a existujúcich stavebných prvkov. Práve ľudský faktor obmedzuje tento návrh na množinu „intuitívnych“ obvodov, ktoré ale nemusia byť vždy optimálne. Vzniká preto otázka automatizácie tohto návrhu s cieľom nájsť optimálne realizácie logických funkcií nezávisle na zaužívaných konvenciách.

Na tento účel sa používa práve kartézske genetické programovanie. Jeho základnými princípmi, možnosťami, obmedzeniami, konfiguráciou a jej vplyvom na výpočtovú silu algoritmu sa zaoberá táto práca.

Kľúčové slová CGP, logické obvody, optimalizácia, evolúcia, genetické programovanie

Obsah

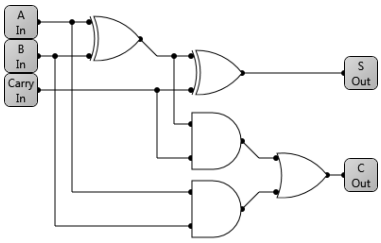
Zoznam obrázkov

Zoznam tabuliek

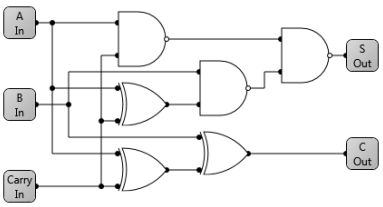
Úvod

Motivácia

Základom všetkých informačných systémov je logický obvod pozostávajúci zo vzájomne prepojených logických členov - hradiel. Každé z týchto hradiel implementuje práve jednu z logických funkcií booleovej algebry, čím transformuje vstupné signály na signály výstupné.



(a) Dvojbitová sčítačka s prenosom zložená z piatich hradiel.



(b) Dvojbitová sčítačka s prenosom zložená zo šiestich hradiel.

Pomocou hradiel implementujúcich základné operácie booleovej algebry (*AND* – konjunkcia, *OR* – disjunkcia a *NOT* – negácia), je možné reprezentovať ľubovoľnú komplexnú logickú funkciu [1]. Táto reprezentácia však nie je unikátna – pre jedinú logickú funkciu (definovanú pravdivostnou tabuľkou[1]) existuje nekonečne mnoho rôznych realizácií v podobe logických obvodov (obrázok vyššie). Tieto obvody sa môžu navzájom líšiť

[1] popisuje hodnoty výstupných premenných logického obvodu v závislosti na konfigurácií vstupných premenných obvodu. [9]

ako usporiadaním hradiel, tak aj množinou použitých hradiel (nahradenie podobvodu jednoduchých hradiel hradlom realizujúcim komplexnejšiu logickú funkciu[2]), alebo optimálnosťou návrhu.

Kvalita návrhu logických obvodov je preto významným predpokladom efektívneho fungovania informačného systému ako celku.

Cieľ práce

Táto práca sa zameriava na automatizáciu návrhu logických obvodov s využitím evolučných algoritmov metódou kartézskeho genetického programovania. Jej cieľom je preskúmať princípy a možnosti tohto prístupu, prezentovať návrh a implementáciu algoritmu zhotoveného na základe tejto analýzy a vyhodnotiť výsledky získané rôznym nastavením parametrov implementovaného algoritmu.

V kapitole 1 sú predstavené metódy návrhu logických obvodov a diskutované ich výhody a nevýhody. Kapitola 3 sa zameriava na všeobecnú charakteristiku a princípy CGP (kartézskeho genetického programovania) ako evolučného algoritmu. Návrh algoritmu CGP pre potreby tejto práce, implementačné detaily a úpravy oproti všeobecnej implementácii popisuje kapitola 4. V kapitole 5 je skúmaná efektívnosť implementovaného algoritmu a jej závislosť na vstupných parametroch výpočtu, pričom je hľadané optimálne nastavenie vstupných parametrov a ich závislosť na riešenom probléme. Skúmaný je aj prínos evolúcie ako takej oproti náhodnej konštrukcii obvodu.

Zistené poznatky a diskusia o úspechoch, neúspechoch a ďalšom skúmaní v tejto oblasti sú konečne zhrnuté v závere tejto práce.

[2]napr. XOR, NOR, NAND, XNOR a pod. – všetky tieto funkcie je možné reprezentovať obvodom pozostávajúcim z hradiel implementujúcich niektorú zo základných booleovych operácií.

Kapitola 1

Metódy návrhu logických obvodov

Metódy návrhu logických obvodov pre realizáciu požadovanej logickej funkcie môžeme rozdeliť na dve skupiny [5]:

1. Konvenčný návrh „zhora-dolu“
2. Návrh „*assemble-and-test*“

1.1 Konvenčný návrh logických obvodov

Pri tradičnom prístupe „zhora-dolu“ je obvod navrhovaný človekom. Základom prístupu je presná špecifikácia problému, množina známych princípov a pravidiel resp. obmedzení pre konštrukciu logických obvodov. Na základe týchto predpokladov je možné obvod skonštruovať pomocou dostupných techník (napr. Karnaughove mapy, rozhodovacie diagramy, a pod.) alebo problém rozložiť na menšie podproblémy a riešiť každý z nich individuálne.

1.1.1 Výhody

- Získaný obvod môže byť zložený z menších univerzálnych obvodov (napr. sčítačka).
  - Znovupoužiteľnosť.
  - Jednoduché nahradenie podobvodu napr. jeho optimalizovanou verziou.
  - Pri výrobe je možné použiť už existujúce súčiastky.

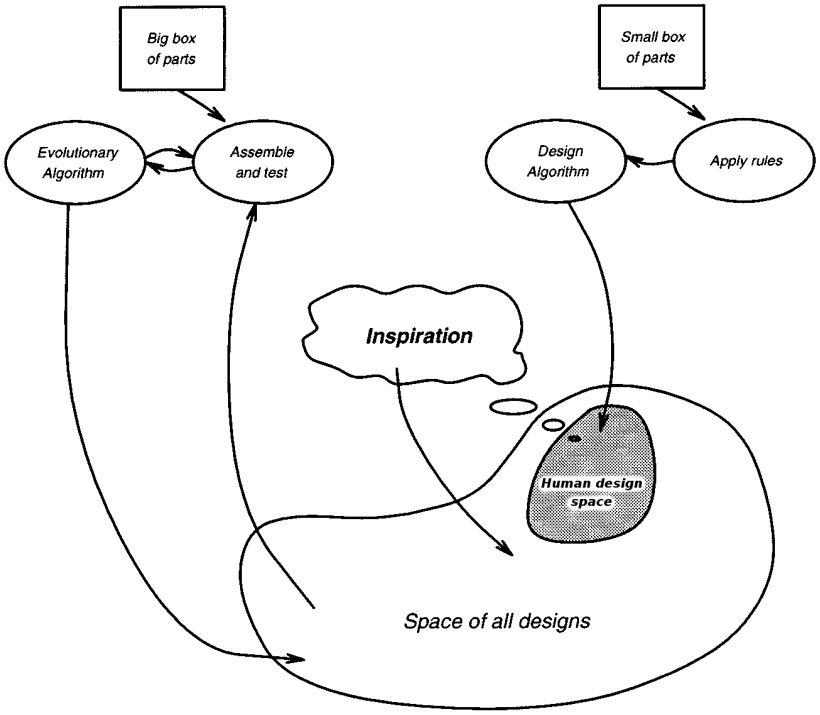
- Intuitívnosť štruktúry.
  - Lepšia „čitateľnosť“ obvodu človekom.
  - Pri návrhu obvodu je možné vychádzať z existujúcich obvodov s podobnou vnútornou štruktúrou.

1.1.2 Nevýhody

Nevýhody tohoto prístupu spočívajú práve v intuitívnosti. Návrh je založený na využití známych princípov a pravidiel resp. zložení komplexného problému z relatívne jednoducho riešiteľných alebo už vyriešených podproblémov. Takto získané obvody nemusia byť vždy optimálne.

Takýto návrh je navyše obmedzený množinou známych podproblémov a pravidiel. Všetky obvody navrhnuteľné človekom preto vytvárajú obmedzenú podmnožinu množiny všetkých možných návrhov obvodu realizujúceho danú logickú funkciu (znázornené na Obr. 1.1).

Skonštruovať obvod mimo tejto podmnožiny sa môže človeku podariť, avšak iba vďaka náhode, omylu alebo inšpirácii.



Obr. 1.1: Konvenčný a evolučný návrh logických obvodov (Zdroj: [5])

1.2 „Assemble-and-test“

Na rozdiel od konvenčného prístupu, metóda „*assemble-and-test*“ nie je pri riešení komplexného problému obmedzená na relatívne malú množinu známych podproblémov a techník ich riešenia. Vychádza z početnej množiny stavebných prvkov (napr. hradiel), z ktorých je najprv zložený kandidátny obvod na riešenie daného problému a až následne sa skúma, či zhotovený obvod skutočne rieši daný problém resp. realizuje požadovanú logickú funkciu. Evolučným algoritmom sa následne kandidátny obvod upravuje tak, aby riešenie konvergovalo k optimálnemu obvodu pre daný problém.

1.2.1 Výhody

Nespornou výhodou tohto prístupu je, že je ním možné skonštruovať ľubovoľný obvod z celej množiny obvodov realizujúcich požadovanú logickú funkciu – takýto návrh nie je zaťažený známymi pravidlami alebo konvenciami konštruovania logických obvodov (Obr. 1.1). Oproti konvenčnému prístupu má teda väčší potenciál pre nájdenie optimálneho riešenia aj pri relatívne komplikovaných problémoch.

1.2.2 Nevýhody

Použitie nekonvenčného logického obvodu môže znamenať okrem obmedzenej znovupoužiteľnosti a modifikovateľnosti aj nutnosť výroby unikátnych jednoúčelových súčiastok a tým predraženie výsledného produktu, na rozdiel od získania požadovanej funkcionality vhodným zložením už existujúcich súčiastok.

Kapitola 2

Úvod do evolučných algoritmov

Evolučným algoritmom sa rozumie optimalizačný algoritmus iteratívne pracujúci s množinami – *generáciami* potenciálne platných riešení daného problému. Tieto riešenia sa v kontexte evolúcie označujú ako *jedinci*.

Základná myšlienka týchto algoritmov je inšpirovaná prírodou – v každej iterácii výpočtu algoritmus pracuje s práve jednou generáciou obsahujúcou stanovený počet jedincov, pričom cieľom výpočtu je zabezpečiť zlepšovanie relatívnej kvality generácií v priebehu výpočtu, až kým nebude nájdené dostatočne presné riešenie problému alebo splnená iná podmienka pre ukončenie algoritmu (napr. dosiahnutie maximálneho povoleného počtu iterácií), kedy algoritmus skončí zlyhaním.[11]

Za týmto účelom je potrebné stanovenie metriky vyhodnocovania kvality jedincov generácie.

2.1 Jedinec a jeho „kvalita“

Jedinec je nositeľom určitej „genetickej informácie“, resp. *genotypu*, ktorý ho charakterizuje.

„Kvalita“ jedinca vyjadruje jeho schopnosť plniť stanovenú úlohu, pričom odráža jeho relatívnu vzdialenosť od iných jedincov, prípadne od *optimálneho* riešenia alebo riešenia s postačujúcou kvalitou. Pre priebeh algoritmu je preto nevyhnutná schopnosť *porovnávať* jedincov.

Kvalitatívne hodnotenie jedincov je realizované pomocou *fitness funkcie*, ktorá každému jedincovi na základe stanovenej metriky jednoznačne pridelí určitú (napr. číselnú) hodnotu – *fitness*. Jedinci s vyššou *fitness* sú považovaní za „lepšie“ riešenia danej úlohy.

2.2 Generácie jedincov a ich evolúcia

Každá generácia v priebehu výpočtu obsahuje určitý počet jedincov. V každej iterácii výpočtu je z generácie na základe *selekčnej stratégie* vybraných niekoľko „najsilnejších“ jedincov – *predkov*, z ktorých je potom za použitia *genetických operácií* vygenerovaná nová, silnejšia generácia *potomkov*. Tento cyklus sa označuje ako *evolúcia*.

Genetickými operáciami aplikovanými na jedincov sú *kríženie* a *mutácia*.

2.2.1 Kríženie

Do operácie kríženia vstupujú minimálne dvaja jedinci. Z genotypu každého z týchto jedincov je vybraná podmnožina *génov*, ktorá sa v najväčšej miere zasluhuje o kvalitu daného jedinca. Vhodným skombinovaním – *skrížením* získaných génov je vytvorený nový jedinec, ktorý by mal takýmto spôsobom od oboch svojich rodičov zdediť ich „dobré vlastnosti“, a teda vo výsledku vykazovať väčšiu kvalitu (resp. fitness), ako obidvaja jeho rodičia.

Účelom kríženia je postupné zvyšovanie priemernej fitness generácií v priebehu výpočtu, a teda konvergencia k optimálnemu riešeniu danej úlohy.

2.2.2 Mutácia

Mutáciou sa u evolučných algoritmov rozumie *náhodná zmena* génu genotypu jedinca. Do tejto operácie teda vstupuje len jeden jedinec.

Účelom mutácie je zanášanie náhody do evolúcie, a teda zabezpečenie dostatočnej *diverzifikácie* jedincov pre umožnenie konvergencie výpočtu do *globálneho* optima.

2.3 Selekčný tlak

Rýchlosť konvergencie výpočtu je ovplyvňovaná *selekčným tlakom*. Príliš vysoký selekčný tlak spôsobí rýchlu konvergenciu výpočtu, ktorý však v takom prípade nemusí nutne skončiť v globálne optimálnom riešení.

Konvergencii do lokálneho optima je možné predísť ako reštartami výpočtu, tak aj znížením selekčného tlaku (v tomto prípade napríklad zvýšením mutácie). Algoritmus má tak väčšiu šancu náhodne „zablúdiť“, čím je schopný pokryť väčšiu „plochu“ stavového priestoru jedincov a tým skonvergovať do globálneho optima. Táto konvergencia je však oproti predošlému spôsobu pomalšia.

V praxi je pre efektívny výpočet algoritmu potrebné nájsť vhodné nastavenie selekčného tlaku pre zabezpečenie požadovanej rovnováhy medzi kvalitou a rýchlosťou výpočtu algoritmu.

2.3.1 Algoritmus

Výpočet genetického algoritmu prebieha v tomto cykle [4]:

1. Vygeneruj počiatočnú generáciu jedincov o veľkosti p (počet generácií $g = 0$).
2. Vypočítaj *fitness* pre každého jedinca generácie.
3. Na základe fitness vyber z generácie rodičov nasledujúcej generácie.
4. Kombináciou (krížením) niektorých (všetkých) vybraných rodičov vygeneruj potomka nasledujúcej generácie.
5. So stanovenou pravdepodobnosťou aplikuj náhodnú mutáciu.
6. Z rodičov a vygenerovaného potomka zostroj novú generáciu ($g \leftarrow g + 1$).
7. Kým $g <$ maximálny počet vygenerovaných generácií a nebolo nájdené riešenie postačujúcej kvality, pokračuj krokom 2.
8. Vráť najlepšie nájdené riešenie. V prípade dosiahnutia maximálneho počtu generácií bez nájdenia riešenia postačujúcej kvality algoritmus zlyhal.

Kapitola 3

Kartézske genetické programovanie (CGP) – princípy

Kartézske genetické programovanie (Cartesian Genetic Progaming – CGP) je formou evolučného algoritmu zameraného na evolúciu logických obvodov a programov [15][7]. Jedinci predstavujú acyklické orientované grafy, ktorých uzly obsahujú hradlá realizujúce určitú logickú funkciu a orientované hrany vyjadrujú prepojenie hradiel v obvode. Pomenovanie „kartézske“ vychádza z organizácie uzlov grafu do dvojrozmernej mriežky.

Cieľom tohto algoritmu je objaviť obvody realizujúce požadovanú logickú funkciu za použitia čo najmenšieho počtu hradiel (minimalizácia veľkosti obvodu). CGP patrí do skupiny „*assemble-and-test*“ (viď 1.2) prístupov k návrhu logických obvodov.

3.1 Jedinec a jeho reprezentácia

Jedincom sa u CGP rozumie kandidátny logický obvod. Reprezentovaný je mriežkou uzlov resp. *hradiel* tvorenou n_r riadkami a n_c stĺpcami. Každé z týchto $|L_n| = n_r n_c$ hradiel realizuje práve jednu logickú funkciu z množiny logických funkcií Γ ($|\Gamma| = n_f$).

Každý jedinec (logický obvod) má stanovený rovnaký počet n_i vstupov $\{X_1, \ldots, X_{n_i}\}$ a n_o výstupov $\{Y_1, \ldots, Y_{n_o}\}$. Tento počet je určený počtom vstupov a výstupov výslednej logickej funkcie realizovanej celým obvodom.

3.1.1 Prepojenie hradiel jedinca

Výstupný signál jedinca ako logického obvodu je získaný transformáciou vstupného signálu vzájomne poprepájanými uzlami resp. hradlami obvodu.

Každý uzol c_{ij} ($i \in \{1, \ldots, n_r\}$, $j \in \{1, \ldots, n_c\}$) obsahuje n_a vstupných hrán, ktorými je pripojený na iné uzly jedinca. Počet vstupov (n_a) závisí na konkrétnej logickej funkcií realizovanej daným uzlom.

Uzly obvodu sú navzájom prepojené tak, aby v obvode nedochádzalo k cyklom[3]. Každý uzol môže byť svojím vstupom pripojený na ktorýkoľvek z uzlov v l predchádzajúcich stĺpcoch mriežky. Uzly v prvých l stĺpcoch môžu tiež mať na svoje vstupy privedené vstupy obvodu. Žiadny uzol však nemôže byť pripojený na uzly rovnakého stĺpca alebo stĺpcov s vyšším indexom, ako je index stĺpca daného uzla.

Parameter l (*levels-back*) teda vyjadruje mieru vnútornej previazanosti uzlov obvodu [5]. Naberá hodnoty z intervalu $\langle 1, n_c \rangle$. Nastavenie parametra l na hodnotu 1 spôsobí pripojenie vstupov každého uzla len na uzly predošlého stĺpca (susedný stĺpec vľavo), pričom uzly v prvom stĺpci môžu byť pripojené len na vstupy obvodu. Naopak, najbenevolentnejšie prepojenie uzlov obvodu je dosiahnuté nastavením parametra l na hodnotu $l = n_c$. Takéto nastavenie umožní pripojenie vstupov každého uzla na ľubovoľný uzol „predošlých“ stĺpcov obvodu (alebo ľubovoľný vstup obvodu) [4].

3.1.2 Kódovanie jedinca

Jedinec resp. logický obvod je jednoznačne určený svojím *genotypom*.

Genotyp u CGP je reťazec čísel obsahujúci

$$n_r n_c (n_a + 1) + n_o \tag{3.1}$$

znakov – „*génov*“.

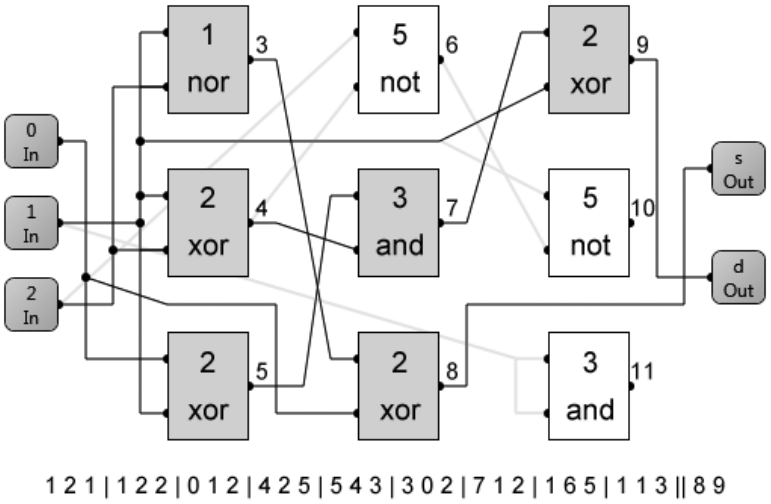
Každému uzlu obvodu je jednoznačne pridelený číselný identifikátor. Hodnotami $\langle 0, n_i - 1 \rangle$ sú označené vstupy obvodu, jednotlivým uzlom sú následne po poradí pridelené identifikátory $\langle n_i, n_r n_c - 1 \rangle$.

Každý z $n_r n_c$ uzlov obvodu je v genotype charakterizovaný $(n_a + 1)$-číselným podreťazcom. Prvých n_a génov obsahuje identifikátory uzlov, ktorých výstupy sú privedené na vstup daného uzla. Tieto gény sa označujú ako „*gény prepojenia*“[4]. Za nimi nasleduje tzv. „*gén funkcie*“[5] obsahujúci kód (f) logickej funkcie realizovanej daným uzlom resp. hradlom.

[3] Ombedzenie na acyklické obvody nie je všeobecnou podmienkou CGP. Problematika CGP pri cyklických obvodoch však presahuje rozsah tejto práce.

[4] z ang. orig. *connection genes*[4]

[5] z ang. orig. *function gene*[4]



Obr. 3.1: Ukážka vnútornej štruktúry jedinca a jeho genotypu pre $n_r = 3$, $n_c = 3$, $l = 3$, $n_i = 3$, $n_o = 2$, $\Gamma = \{$or (0), nor (1), xor (2), and (3), nand(4), not(5)$\}$. Vyfarbené uzly sa podieľajú na funkcionalite obvodu. [7]

Posledných n_o génov obsahuje identifikátory uzlov, ktorých výstupy sú zároveň výslednými výstupmi obvodu [6]. Štruktúra genotypu jedinca je zobrazená na Obr. 3.2 (b).

3.1.3 Fenotyp

Fenotypom sa označuje reálna reprezentácia obvodu pomocou vzájomne prepojených logických hradiel.

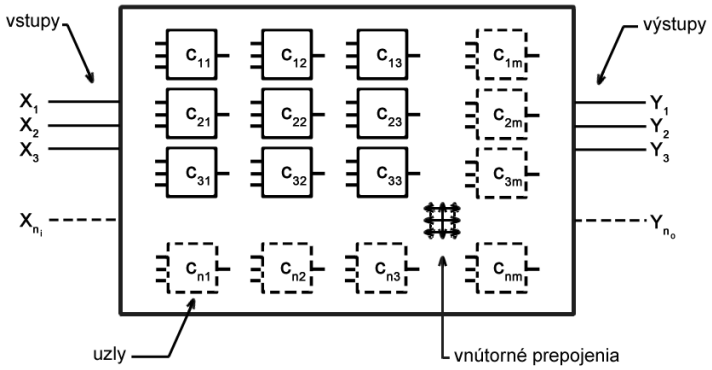
V priebehu evolučného algoritmu, kedy dochádza k zmene vnútornej štruktúry obvodu, môže nastať situácia, kedy výstup niektorého z uzlov obvodu (alebo celej „vetvy“ uzlov) nie je pripojený k žiadnemu ďalšiemu uzlu ani výstupu obvodu. Fenotyp jedinca tak môže obsahovať aj tzv. *neutrálne* uzly (podrobnejšie v časti 3.3), ktoré nemajú vplyv na funkcionalitu obvodu.

Fenotyp a jeho mapovanie na genotyp znázorňuje Obr. 3.1 a Obr. 3.2.

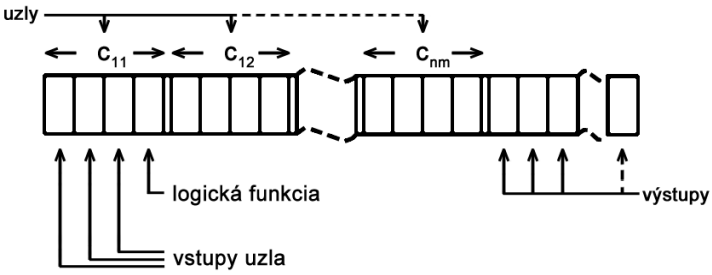
3.1.4 Obmedzenia

CGP sa v prevažnej miere používa pre optimalizáciu *acyklických* obvodov[6] [4]. Vzniku cyklov v obvode sa zamedzuje obmedzením, ktoré na vstupy

[6]Výpočet CGP na cyklických obvodoch prekračuje rozsah tejto práce.



(a) Fenotyp jedinca



(b) Genotyp jedinca

Obr. 3.2: Mapovanie fenotypu na genotyp

ľubovoľného uzla obvodu povoľuje priviesť jedine výstupy uzlov z l predošlých stĺpcov (l – parameter *levels-back*).

Pre hodnoty génov genotypu jedinca platia striktné obmedzenia, ktoré musia byť dodržané ako už pri samotnej inicializácií obvodu, tak aj pri jeho zmene spôsobenej mutáciou počas výpočtu evolučného algoritmu.

- Označme f_{ij} identifikátor logickej funkcie realizovanej uzlom c_{ij} a $|\Gamma| = n_f$ počet všetkých primitívnych uvažovaných logických funkcií realizovaných uzlami. Pre ľubovoľný uzol c_{ij} musí potom v každom okamihu výpočtu platiť:

$$0 \leq f_{ij} < n_f. \tag{3.2}$$

- Označme ďalej C_{kj} identifikátor uzla, ktorého výstup je privedený na k-ty vstup uzla c_{ij} nachádzajúceho sa v j-tom riadku obvodu. Pre ľu-

bovoľný uzol c_{ij} (kde $j \in \langle 0, n_c - 1 \rangle$), jeho ľubovoľný vstup C_{kj} a parametre l (*levels-back*), n_i a n_r musia byť v každom okamihu výpočtu rovnako splnené nasledujúce podmienky:

  - Ak $j < l$,
$$0 \leq C_{kj} \leq n_i + jn_r. \tag{3.3}$$
  - Ak $j \geq l$,
$$n_i + (j - l)n_r \leq C_{kj} \leq n_i + jn_r. \tag{3.4}$$

- Pre výstupy O_i obvodu musí platiť:
$$0 \leq O_i \leq n_i + L_n. \tag{3.5}$$

3.2 Evolúcia v CGP

CGP patrí do skupiny evolučných algoritmov. Ako taký pracuje iteratívne s celými generáciami jedincov, pričom každý jedinec môže byť potenciálnym riešením danej úlohy.

3.2.1 Generácia jedincov

Každá generácia vygenerovaná počas výpočtu algoritmu obsahuje $1 + \lambda$ jedincov[7]. Pri inicializácií prvej generácie môžu byť všetci jedinci vygenerovaní náhodne alebo odvodení od konvenčného riešenia daného problému. V ďalších iteráciách výpočtu je vždy z aktuálnej generácie na základe použitého selekčného mechanizmu[8] vybraný jedinec, z ktorého je následne odvodených ďalších λ jedincov novej generácie (podrobnejšie v časti 3.2.2).

3.2.2 Odvodzovanie nových jedincov

V evolučných algoritmoch sa vo všeobecnosti zmena „genofondu“, resp. odvodzovanie jedincov novej generácie realizuje dvoma spôsobmi:

(a) krížením,

(b) mutáciou.

Obidva spôsoby a ich uplatnenie v CGP je popísané v nasledujúcich častiach [4].

[7] Evolučná stratégia $1 + \lambda$.

[8] Všeobecne sa pri evolučných algoritmoch vyberá ako predok nasledujúcej generácie jedinec s najvyššou hodnotou *fitness* funkcie. Podrobnejšie o selekčnom mechanizme v CGP v časti 3.2.4.

3.2.2.1 Kríženie a CGP

Myšlienkou kríženia v genetických algoritmoch je vytvorenie „silnejšej“ generácie jedincov skombinovaním vhodných častí genotypov predkov z predošlej generácie. Jedinci vytvorení takýmto spôsobom by mali od svojich predkov zdediť ich „dobré“ vlastnosti a ich vhodnou kombináciou vytvárať v postupných iteráciách výpočtu generácie konvergujúce k optimálnemu riešeniu. Problém využitia tejto metódy v CGP však spočíva vo vnútornej organizácií logických obvodov.

Logický obvod veľmi úzko závisí nielen na použitých hradlách, ale aj na ich rozmiestnení v obvode, resp. ich vzájomnom prepojení. Najmenšia zmena genotypu obvodu tak môže spôsobiť dramatickú zmenu v jeho výstupe. Z tohto dôvodu nie je možné určiť „silnú“ časť genotypu obvodu, ktorá by sa zúčastnila kríženia.

Kríženie sa preto u CGP **nepoužíva**.

3.2.2.2 Mutácia

Ako jediný genetický operátor sa u CGP používa *mutácia*. Pri mutácii sa aplikuje **náhodná** zmena náhodnej časti genotypu obvodu nového jedinca. Môže ísť o:

- zmenu logickej funkcie implementovanej mutovaným hradlom (s tým spojená zmena počtu vstupov hradla),
- zmenu vstupu mutovaného uzla (zmena prepojenia hradiel),
- zmenu výstupného hradla obvodu.

Počet mutácií vykonaných nad genotypom pri odvodzovaní nového jedinca je vstupným parametrom výpočtu. Môže byť zadaná absolútne, ako počet mutovaných génov, alebo ako percentuálny podiel.

3.2.3 Vyhodnocovanie kvality jedinca – fitness

V evolučných algoritmoch je potrebné stanoviť metriku, na základe ktorej bude vyhodnocovaná kvalita jedinca. Tá je vyjadrená číselným koeficientom, ktorý obvodu na základe spomínanej metriky pridelí *fitness* funkcia.

Pri probléme, akým je CGP, kedy algoritmus hľadá čo najmenší obvod realizujúci referenčnú logickú funkciu, je ako kvalitatívny faktor definovaná funkcia zohľadňujúca počet správnych výstupov na všetky možné konfigurácie vstupných premenných. Zároveň je pre uplatnenie optimalizačného

kritéria *fitness* daného obvodu tým väčšia, čím menej hradiel potrebuje obvod na realizáciu požadovanej logickej funkcie. V tradičnej implementácií CGP[9] je teda hodnota *fitness* prideľovaná podľa vzorca [15]:

$$fit(c) = \begin{cases} b & \text{ak } b < n_o 2^{n_i}, \\ b + (n_r n_c - z) & \text{inak,} \end{cases} \tag{3.6}$$

kde parameter c označuje kandidátny logický obvod – jedinca, b počet korektných výstupov obvodu na všetky možné konfigurácie vstupov a parameter z počet reálne zapojených hradiel obvodu. Parameter z sa samozrejme zohľadňuje jedine v prípade, ak je skúmaný jedinec *funkčný*, tzn. $b = b_{max} = n_o 2^{n_i}$.

3.2.4 Selekčný mechanizmus, selekčný tlak

Tradičná implementácia evolučného algoritmu vyberá v každej iterácií z aktuálnej generácie jedného alebo niekoľkých „najsilnejších“ jedincov, z ktorých sa odvodí nová generácia. Spôsob implementácie selekčného mechanizmu pre účely tejto práce bude diskutovaný v časti 4.2.4.

Selekčným tlakom sa u evolučného algoritmu rozumie mechanizmus ovplyvňujúci rýchlosť konvergencie algoritmu k lokálne najlepšiemu riešeniu. Zvýšenie selekčného tlaku všeobecne zrýchľuje konvergenciu, avšak riešenie, do ktorého algoritmus skonverguje, nemusí byť vždy globálne najlepším riešením. Naopak, znížením selekčného tlaku spravidla dochádza k pomalšej, no presnejšej konvergencii.

U CGP sa evolúcia realizuje výhradne mutovaním génov genotypu jedincov. Táto zmena je úplne náhodná a nie je nijak ovplyvnená „kvalitou“ mutovaného jedinca alebo ostatných jedincov danej generácie. Výpočet CGP teda vo veľkej miere závisí na náhode. Z toho dôvodu nie je jednoduché určiť, akými spôsobmi je možné ovplyvvniť rýchlosť konvergencie výpočtu k optimálnemu riešeniu. Táto otázka bude jedným z predmetov skúmania v experimentoch popísaných v kapitole 5.

3.3 Redundancia v CGP

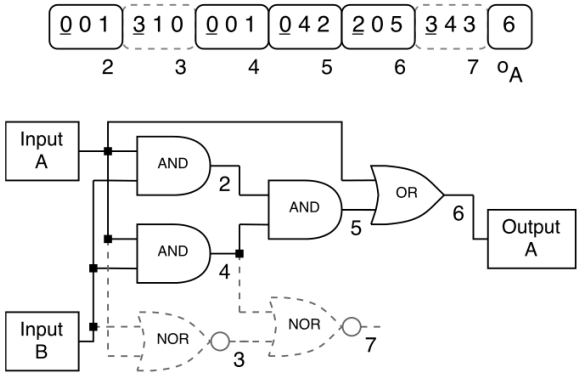
Z popisu jedinca ako obvodu reprezentovaného mriežkou uzlov resp. hradiel (viď časť 3.1) vyplýva, že sa v genotype jedinca nachádzajú aj také uzly, ktoré nie sú zapojené v aktívnej časti obvodu a preto nemajú na výslednú

[9]Podrobnejšie o prideľovaní fitness implementovanom v tejto práci v časti 4.2.3.

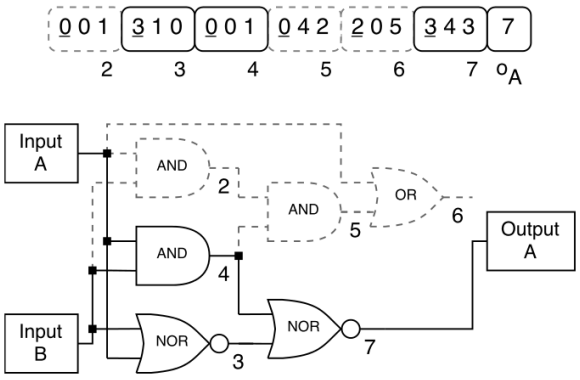
logickú funkciu obvodu žiadny vplyv. Tento jav sa tiež označuje ako *neutralita* [4]. Vzniknúť mohli už pri inicializácií obvodu alebo ako následok evolúcie.

Keďže neutrálne uzly nemajú vplyv na funkciu obvodu, nemajú vplyv ani na jeho *fitness*. Mutáciou neutrálneho génu funkčného jedinca teda vznikne opäť **funkčný jedinec s rovnakou *fitness***. Obvody, ktoré boli z predka odvodené takýmto spôsobom teda nie je potrebné testovať na funkčnosť ani počítať ich *fitness* [4].

Neutrálne uzly sú preto schopné v priebehu evolúcie ľubovoľne mutovať a vytvárať tak náhodné fragmenty obvodu bez toho, aby narušili funkčnosť celého jedinca. Následné nahradenie časti aktívneho obvodu doposiaľ neaktívnym fragmentom môže potenciálne viesť na opäť funkčného jedinca, prípadne aj s lepšou hodnotou *fitness*. Navyše, nahradením celej časti obvodu je možné dosiahnuť komplexnejšej reorganizácie štruktúry obvodu, ktorá by bez redundancie nemusela byť pri danej miere mutácie dosiahnuteľná (Obr. 3.3).



(a) Pred mutáciou



(b) Po zmutovaní výstupu obvodu

Obr. 3.3: Zmena genotypu a fenotypu mutáciou – zmena jedného génu (výstup obvodu) spôsobí nahradenie celej časti obvodu inou, pred mutáciou neaktívnou časťou (zdroj: [4]).

Kapitola 4

Návrh a implementácia algoritmu

Základ algoritmu použitého v tejto práci vychádza zo všeobecnej charakteristiky CGP popísanej v predošlej kapitole. Implementačné detaily a detaily, ktoré sa oproti tomuto všeobecnému prístupu líšia alebo ho rozširujú, budú popísané v nasledujúcich častiach.

4.1 Načítanie dát

Algoritmus načítava referenčný obvod vo formáte *BLIF* [14]. Tento formát popisuje vstupy, výstupy a jednotlié hradlá obvodu, ako aj ich vzájomné prepojenie, na základe jednoznačných názvov resp. identifikátorov hradiel. Logická funkcia hradla je rovnako definovaná svojím názvom.

Na obrázku 4.1 je zobrazená reprezentácia dvojbitovej sčítačky obsahujúcej tri vstupy a päť hradiel, z ktorých dve sú výstupmi obvodu.

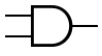
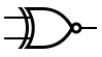
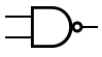
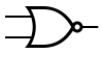
```
.model 1-adder
.inputs pi0 pi1 pi2
.outputs po0 po1
.gate nand2 a=pi2 b=pi0 O=n7
.gate xora2 a=pi2 b=pi0 O=n5
.gate nand2 a=n5 b=pi1 O=n6
.gate xora2 a=n7 b=n6 O=po0
.gate xora2 a=n5 b=pi1 O=po1
.end
```

Obr. 4.1: Reprezentácia dvojbitovej sčítačky vo formáte BLIF

Formát *BLIF* umožňuje serializovať logický obvod aj odlišnými spôsobmi, pre potreby tejto práce však bol využívaný len tento spôsob reprezentácie.

Pre načítanie vstupných súborov bola využitá knižnica **BOOM** [8]. Prehľad prípustných logických funkcií, ich hradiel a reprezentácie v boolovej algebre je znázornený v tabuľke 4.1.

Tabuľka 4.1: Prehľad prípustných logických funkcií

AND		$A \cdot B$	XOR		$A \oplus B$
OR		$A + B$	XNOR		$\overline{A \oplus B}$
INV		$\overline{A}$	BUF		A
NAND		$\overline{A \cdot B}$	ONE		1
NOR		$\overline{A + B}$	ZERO		0

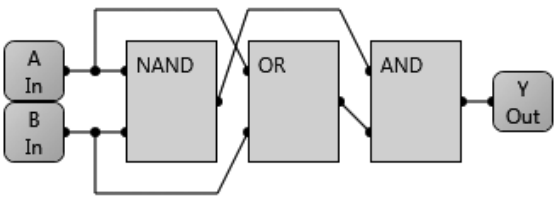
4.2 Evolúcia

4.2.1 Generovanie a selekcia jedincov

V priebehu evolúcie algoritmu sa pracuje výhradne s *funkčnými* jedincami[10].

Počiatočný jedinec (predok prvej generácie) je vytvorený podľa referenčného obvodu načítaného zo vstupných dát algoritmu. Skúmal sa vplyv použitia niekoľkých rôznych inicializácií počiatočného jedinca na priebeh a úspešnosť algoritmu:

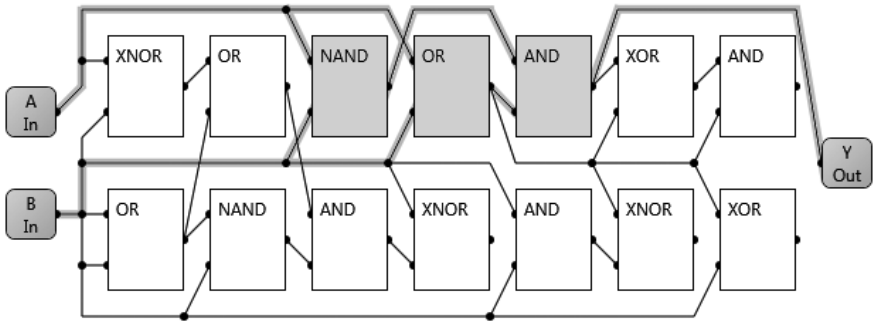
1. **inicializácia bez neutrálnych uzlov** (1 rad hradiel – Obr. 4.2),



Obr. 4.2: Inicializácia obvodu bez neutrálnych uzlov

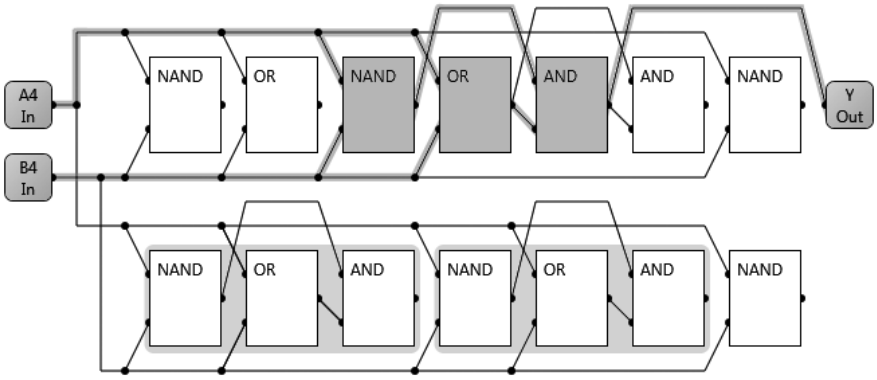
2. **inicializácia s neutrálnymi uzlami** (pridané stĺpce a rady; neutrálne uzly inicializované *náhodne* – Obr. 4.3),

[10]Nie je nutnou podmienkou algoritmu. Dôvody pre zvolenie tejto stratégie sú diskutované v časti 4.2.3.



Obr. 4.3: Inicializácia obvodu s neutrálnymi uzlami inicializovanými náhodne. Zvýraznené sú aktívne uzly a hrany.

3. **inicializácia s neutrálnymi uzlami** (pridané stĺpce a rady; neutrálne uzly inicializované *fragmentami referenčného obvodu* – Obr. 4.4).



Obr. 4.4: Inicializácia obvodu s neutrálnymi uzlami inicializovanými fragmentami referenčného obvodu. Zvýraznené sú aktívne uzly, hrany a dva fragmenty referenčného obvodu (v spodnom rade).

Ďalší jedinci generácie sa z predka generujú mutáciou určeného počtu génov genotypu. Ak pritom vznikne jedinec, ktorý je nefunkčný, zahodí sa. Ak vznikne funkčný jedinec s najvyššou doposiaľ nájdenou *fitness* (najmenším počtom zapojených hradiel zo všetkých doposiaľ nájdených obvodov), uloží sa ako aktuálne najlepšie riešenie. Na začiatku výpočtu je najlepšie nájdené riešenie inicializované referenčným obvodom.

Takýmto spôsobom sa vygeneruje λ funkčných jedincov. Generácia tak obsahuje spolu $\lambda + 1$ jedincov. Z jedincov, ktorí vznikli mutáciou predka,

sa náhodne vyberie jeden ako predok nasledujúcej generácie a celý proces sa opakuje. Žiadny jedinec tak nemôže byť predkom dvoch po sebe idúcich generácií (kvôli snahe zvýšiť diverzitu generácií).

4.2.2 Kontrola ekvivalencie

Významnú úlohu z hľadiska časovej náročnosti v procese evolúcie zohráva práve kontrola ekvivalencie (funkčnosti) nového jedinca a referenčného obvodu. Pre optimalizáciu výpočtovej náročnosti tejto operácie bol využitý zásadný predpoklad:

- **Mutácia neutrálneho uzla funkčného obvodu vedie opäť na funkčný obvod s rovnakou *fitness*.** (viď 3.3) [4]

Pre testovanie funkčnosti jedinca vzniknutého mutáciou *aktívnej* časti obvodu bola použitá kontrola ekvivalencie založená na probléme **SAT**[11].

4.2.2.1 Kontrola ekvivalencie založená na SAT

Kandidátny obvod je ekvivalentný s referenčným obvodom (realizuje rovnakú logickú funkciu) práve vtedy, ak na všetky možné kombinácie vstupov vracia rovnaké výstupy ako referenčný obvod (*výstupy kandidátneho a referenčného obvodu sa pri žiadnej kombinácií vstupov nelíšia*). Obvody sú preto zapojené do jedného logického obvodu tak, ako zobrazuje Obr. 4.5.

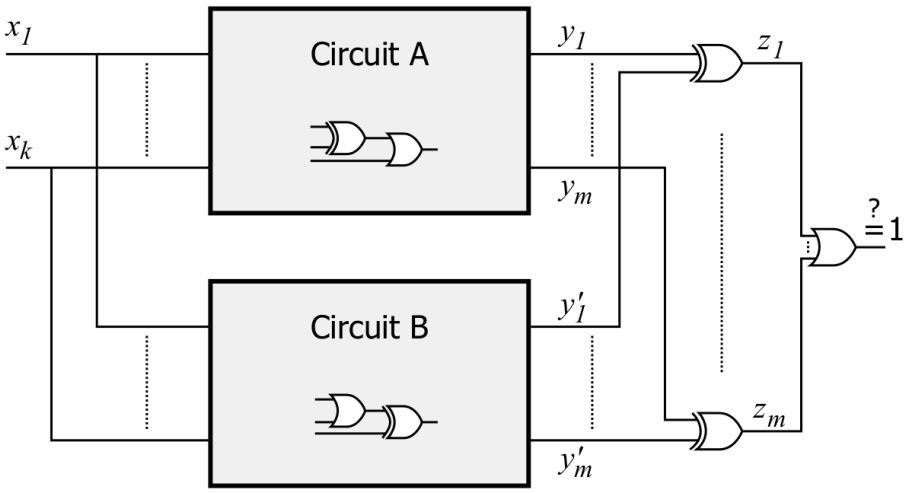
Na vstupy kandidátneho aj referenčného obvodu sú privedené rovnaké signály ($\{x_1, \ldots, x_k\}$). Následne je každý z výstupov prvého obvodu (y_i) spojený s príslušným výstupom druhého obvodu (y_i') hradlom XOR. Výsledný signál tohto hradla bude 1 (*true*) práve vtedy, ak sa budú dané výstupy obvodov líšiť. Spojenie týchto n_o XOR hradiel jedným n_o vstupovým hradlom OR vzniká obvod, ktorého výstup je 1 (*true*) práve vtedy, ak sa niektorá dvojica prílušných výstupov líši.

Reprezentáciou tohto obvodu v *CNF*[12] (viď tabuľka 4.2 a Obr. 4.6) [12] je pravdivostná formula, ktorej **splniteľnosť dokazuje neekvivalenciu kandidátneho a referenčného obvodu.**

Pre riešenie problému splniteľnosti pravdivostnej formuly (SAT) je použitý nástroj ***zchaff*** [13], ktorému sa zadáva formula vo formáte DIMACS [10]. Prevod logického obvodu z jeho schémy do CNF a DIMACS formátu zobrazuje Obr. 4.6.

[11] SAT – (*Boolean satisfiability problem*) NP-úplný problém splniteľnosti pravdivostnej formuly.

[12] Konjunktívna normálna forma – zápis logickej formuly v podobe konjunkcie klauzulí, pričom každá klauzula obsahuje disjunkciu literálov. [3]



Obr. 4.5: Schéma zapojenia obvodov pre kontrolu ekvivalencie pomocou SAT

Tabuľka 4.2: Reprezentácia logických funkcií v CNF

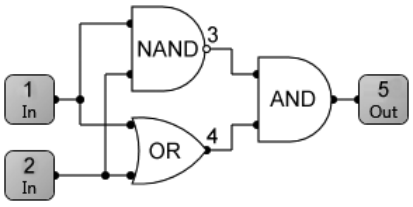
AND	$(\neg a \vee \neg b \vee o) \wedge (a \vee \neg o) \wedge (b \vee \neg o)$
NAND	$(\neg a \vee \neg b \vee \neg o) \wedge (a \vee o) \wedge (b \vee o)$
OR	$(a \vee b \vee \neg o) \wedge (\neg a \vee o) \wedge (\neg b \vee o)$
NOR	$(a \vee b \vee o) \wedge (\neg a \vee \neg o) \wedge (\neg b \vee \neg o)$
XOR	$(\neg a \vee \neg b \vee \neg o) \wedge (\neg a \vee b \vee o) \wedge (a \vee \neg b \vee o) \wedge (a \vee b \vee \neg o)$
XNOR	$(a \vee b \vee o) \wedge (a \vee \neg b \vee \neg o) \wedge (\neg a \vee b \vee \neg o) \wedge (\neg a \vee \neg b \vee o)$
BUF	$(\neg a \vee o) \wedge (a \vee \neg o)$
INV	$(\neg a \vee \neg o) \wedge (a \vee o)$
ZERO	$(\neg o)$
ONE	(o)

4.2.3 Výpočet fitness

Implementovaný výpočet *fitness* kandidátneho obvodu sa oproti spôsobu popísanom v časti 3.2.3 (vzťah 3.6) mierne odlišuje. S uvedenou metrikou sú totiž spojené dva problémy:

1. **Výpočtová náročnosť**

   - Pre vyhodnotenie kvality kandidátneho jedinca by bolo potrebné otestovať všetky možné konfigurácie vstupných premenných. Ich



(a) Schéma obvodu

```
c comment
p cnf 5 9
-1 -2 -3 0
1 3 0
2 3 0
1 2 -4 0
-1 4 0
-2 4 0
-3 -4 5 0
3 -5 0
4 -5 0
```

(b)
DIMACS

$$(\neg A \vee \neg B \vee \neg C) \wedge (A \vee C) \wedge (B \vee C) \wedge$$
$$(A \vee B \vee \neg D) \wedge (\neg A \vee D) \wedge (\neg B \vee D) \wedge$$
$$(\neg C \vee \neg D \vee E) \wedge (C \vee \neg E) \wedge (D \vee \neg E)$$

(c) CNF

Obr. 4.6: Prevod obvodu zo schémy (a) do reprezentácie v CNF (c) a CNF do formátu DIMACS (b)

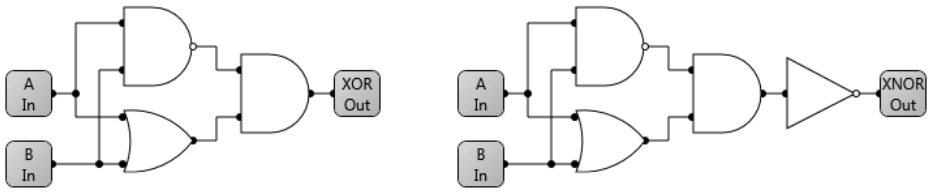
počet však rastie exponenciálne s počtom vstupných premenných ($N = 2^n$).

2. **Problém vzdialenosti**

- Z dôvodu vysokej závislosti výslednej logickej funkcie na vnútornej štruktúre logického obvodu (použité hradlá, ich rozmiestnenie a prepojenie), je problematické určiť vzájomnú „vzdialenosť“ alebo „podobnosť“ dvoch logických obvodov, resp. určiť relatívnu vzdialenosť daného obvodu od „najbližšieho“ funkčného obvodu (tejto problematike sa venuje aj časť 3.2.2.1 *Kríženie a CGP*). *Nedá sa* preto skonštatovať, že by transformácia nefunkčného obvodu[13] na funkčný obvod[14] vyžadovala tým menej krokov, čím viac správnych výstupov daný kandidátny obvod vracia.

[13]Obvod funkčne neekvivalentný s referenčným obvodom.
[14]Obvod funkčne ekvivalentný s referenčným obvodom.

Túto skutočnosť demonštruje Obr. 4.7 – pridaním invertora pred výstup logického obvodu realizujúceho logickú funkciu XOR (Obr. 4.7 (a)), dostávame obvod realizujúci logickú funkciu XNOR (Obr. 4.7 (b)). Tieto obvody sa navzájom líšia jedným hradlom, pričom pre transformáciu obvodu XNOR na XOR je potrebná **len jedna** mutácia (pripojenie výstupu obvodu na hradlo AND namiesto INV). Funkcie XOR a XNOR sa však nezhodujú v **žiadnom** výstupe (Tabuľka 4.3).



(a) XOR (b) XNOR

Obr. 4.7: Obvody realizujúce logickú funkciu XOR (a) a XNOR (b), ktorý vznikol z obvodu (a) pridaním invertora na výstup obvodu.

Tabuľka 4.3: Pravdivostná tabuľka funkcií XOR a XNOR

XOR			XNOR		
A	B	Q	A	B	Q
0	0	0	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1

Vysokú výpočtovú záťaž by rovnako znamenalo porovnávanie topológie obvodov a vyhľadávanie podobných fragmentov. Navyše, vplyv fragmentu obvodu na výslednú logickú funkciu sa môže (z rovnakých dôvodov, ako sú uvedené vyššie) v rôznom kontexte líšiť.

Kvôli problému „vzdialenosti“ sa počet správnych výstupov obvodu na všetky možné konfigurácie vstupných premenných nedá považovať za spoľahlivú metriku kvality nefunkčného obvodu, resp. tento údaj nemusí spoľahlivo vyjadrovať minimálny počet krokov potrebných na transformáciu

daného nefunkčného obvodu na obvod funkčný. Nefunkčný obvod navyše nie je riešením optimalizačnej úlohy.

Z týchto dôvodov sú všetky nefunkčné obvody v tejto práci považované za *rovnako zlé*. Ich hodnota *fitness* je stanovená na -1 a neuvažujú sa ani ako jedinci generácií evolučného algoritmu.

U funkčného obvodu sa pre výpočet *fitness* používa upravený vzťah:

$$fit(c) = n_r n_c - z. \tag{4.1}$$

Obvodu, ktorý neobsahuje žiadne neutrálne uzly, je teda pridelená hodnota *fitness*: $fit(c) = 0$.

4.2.4 Selekčný mechanizmus

Z netriviálnosti určenia vzdialenosti dvoch logických obvodov (podrobnejšie v predošlej časti) však vyplýva vysoká pravdepodobnosť, že nový jedinec získaný mutáciou hoci aj funkčného jedinca nebude funkčne ekvivalentný s referenčným logickým obvodom. Výber jedinca s najvyššou hodnotou *fitness* teda negarantuje konvergenciu algoritmu k optimálnemu riešeniu. Z tohto dôvodu bol selekčný mechanizmus mierne pozmený.

Použitý selekčný mechanizmus obmedzuje výber predka pre nasledujúcu generáciu len na **funkčný obvod**, takže pri výbere predka sa nezohľadňuje jeho *fitness*. Ak aktuálna generácia obsahuje funkčných jedincov niekoľko, selekčný mechanizmus z nich vyberie náhodne. V takom prípade sa však ako kandidát na predka neuvažuje jedinec, ktorý bol ako predok použitý v poslednej iterácii [15].

Pre výber predka (p') pre nasledujúcu generáciu teda platí vzťah [15]:

$$p' = \begin{cases} p & \text{ak } \forall i, i = 1 \ldots \lambda : f_{x_i} < b_{max}, \\ x_j & \text{inak,} \end{cases} \tag{4.2}$$

kde p je predok aktuálnej generácie a x_j je náhodne vybraný jedinec z množiny funkčných jedincov okrem rodiča aktuálnej generácie.

Takto nastavený selekčný mechanizmus si samozrejme nie je schopný udržiavať globálne najlepšie nájdené riešenie, ktoré je preto potrebné uchovávať zvlášť (β) a v každej iterácií algoritmu aktualizovať (ak sa v novej generácii nájde lepšie riešenie). Za výsledné riešenie evolučného algoritmu sa potom neberie najsilnejší jedinec poslednej generácie, ale β.

4.3 Zaznamenávanie a vyhodnocovanie priebehu algoritmu

Pre porovnávanie priebehu a úspešnosti algoritmu je potrebné určiť metriku, ktorou bude možné vyhodnotiť jednotlivé reštarty algoritmu čo najobjektívnejšie. Za týmto účelom bol zvolený prístup, ktorým je zabezpečené, aby všetky reštarty bežali rovnako dlho.

Časová náročnosť výpočtu algoritmu je však vo veľkej miere závislá na použitej platforme. Z toho dôvodu jej použitie pre tento účel nie je vhodné.

Pre zabezpečenie rovnakej doby behu algoritmu je v tejto práci stanovené obmedzenie na maximálny počet vygenerovaných obvodov. Každý obvod vzniknutý mutáciou predka, či už funkčný alebo nefunkčný, je započítaný do celkového počtu vygenerovaných obvodov. Pri prekročení stanoveného limitu (10 miliónov obvodov) je algoritmus ukončený nezávisle na reálnom čase potrebnom pre vygenerovanie jedného jedinca.

4.3.1 Výstup algoritmu

Počas výpočtu algoritmu sa uchováva najlepšie doposiaľ nájdené riešenie (najmenší nájdený funkčný obvod – viď časť 4.2.1). Pri aktualizácií tohto riešenia sa nové najlepšie riešenie serializuje do súboru vo formáte *BLIF* (podrobnejšie popísaný v časti 4.1 – Načítanie dát).

4.3.2 Overenie výsledku

Obvody získané ako výsledky výpočtu algoritmu sa overujú, či skutočne realizujú ekvivalentnú logickú funkciu s funkciou referenčného obvodu. Pre tento účel sa používa nástroj **ABC** [2].

4.4 Platforma

- Algoritmus bol implementovaný v jazyku *C++*.

Kapitola 5

Analýza závislosti efektivity výpočtu algoritmu na vstupných parametroch

Pred výpočtom alrogitmu CGP je potrebná inicializácia vstupných parametrov a nastavenie základných výpočtových prístupov algoritmu. Vplyv inicializácie jednotlivých parametrov na priebeh a efektívnosť[15], resp. kvalitu algoritmu sa skúma v nasledujúcich podkapitolách. Ide o parametre:

- stanovenie miery previazanosti uzlov jedinca (parameter l),
- veľkosť generácie (parameter λ),
- miera mutácie (parameter m), jej závislosť na veľkosti fenotypu a štruktúre referenčného obvodu,
- veľkosť mriežky resp. genotypu jedinca (parametre n_r, n_c),
  - vplyv redundancie,
- spôsob inicializácie redundantných génov jedinca.

5.1 Testovací problém

Pre účely testovania vplyvu parametrov na priebeh algoritmu bol použitý testovací obvod **xor5**. Ide o obvod realizujúci päťvstupovú funkciu XOR

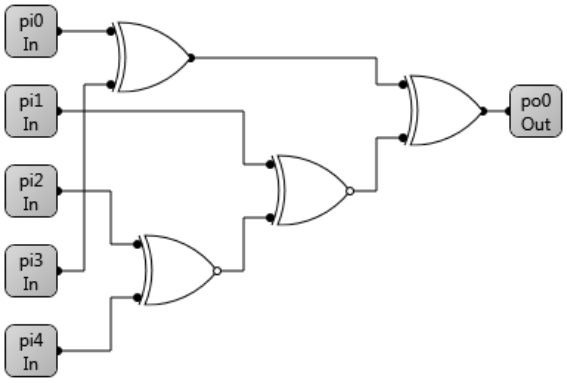
[15]Vzhľadom na zvolenú metriku vyhodnocovania priebehu výpočtu algoritmu v závislosti na počte vygenerovaných jedincoch sa ako *efektívnosť* výpočtu rozumie schopnosť algoritmu nájsť optimálne riešenie pri čo najmenšom počte vygenerovaných jedincov.

s jedným výstupom. Algoritmu je na vstupe daná realizácia tohto obvodu za použitia 52 hradiel, pričom najmenší dosiahnuteľný obvod realizujúci túto logickú funkciu je možné zložiť zo štyroch hradiel (jedno z možných riešení zobrazuje Obr. 5.1).

Parametre testovacieho problému sú zhrnuté v tabuľke 5.1.

Tabuľka 5.1: Parametre testovacieho problému

Názov obvodu:	**xor5**
Počet vstupov:	5
Počet výstupov:	1
Počet hradiel:	52
Optimálny počet hradiel:	**4**
Zlepšenie:	**92,3%**



Obr. 5.1: Jedna z možných optimálnych realizácií funkcie xor5 pomocou dvoch hradiel XOR a dvoch XNOR (obvod vygenerovaný implementovaným algoritmom CGP).

5.2 Miera previazanosti uzlov jedinca

Pre uzol $n_{i,j}$, nachádzajúci sa v i-tom stĺpci a j-tom riadku jedinca, určuje miera previazanosti uzlov jedinca (reprezentovaná parametrom l) maximálny počet stĺpcov pred stĺpcom i, ku ktorým (resp. k ich uzlom) môžu byť pripojené vstupy uzla $n_{i,j}$ (podrobnejšie v časti 3.1.1 – Prepojenie hradiel jedinca).

Nutnosť obmedzenia previazanosti uzlov môže vyplývať z konkrétneho zadania úlohy alebo iných (napr. konštrukčných) obmedzení. V problémoch riešených v tejto práci však nie je žiadny dôvod pre takéto obmedzenie. Z tohto dôvodu sa v ďalších experimentoch uvažuje s *maximálnou* previazanosťou uzlov jedinca, tzn. $l = n_c$. Uzly jedinca tak môžu byť svojími vstupmi pripojené k ľubovoľným uzlom v predchádzajúcich stĺpcoch mriežky jedinca.

5.3 Vplyv veľkosti generácie na výpočet algoritmu

Každá generácia vytvorená počas výpočtu algoritmu obsahuje celkom $\lambda + 1$ jedincov, pričom jeden z jedincov vystupuje ako predok danej generácie. Zvyšných λ jedincov je následne vygenerovaných jeho mutáciou (viď časť 4.2.1).

Vplyv veľkosti generácie na výpočet algoritmu bol skúmaný pre hodnoty parametra $\lambda \in \{1, 2, 5, 10, 20\}$. Pre každú z hodnôt bolo vykonaných spolu 40 meraní za použitia dvoch stratégií inicializácie prvého jedinca:

1. **inicializácia bez redundancie**,
   - Prvý jedinec je inicializovaný referenčným obvodom tak, že neobsahuje žiadne nezapojené/redundantné uzly.
   - *Fitness*[16] prvého jedinca je 0.
2. **inicializácia s redundantnými uzlami**.
   - Prvý jedinec obsahuje trojnásobný počet uzlov (hradiel) ako referenčný obvod.
   - Zapojené uzly sú sústredené uprostred riadku mriežky.
   - Redundantné uzly sú inicializované fragmentami referenčného obvodu (viď časť 4.2.1).

V oboch prípadoch obsahuje fenotyp jedinca len jeden riadok maximálne prepojených uzlov ($l = n_c$).

Miera mutácie bola pevne stanovená na hodnotu $m = 2$ – noví jedinci boli odvodzovaní mutáciou práve dvoch génov predka[17].

[16] Počíta sa ako počet redundantných uzlov obvodu (podrobnejšie v časti 3.2.3).

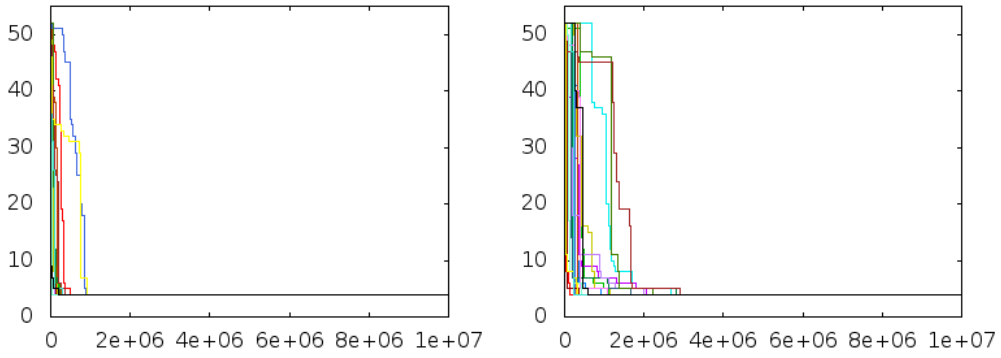
[17] Hodnota mutácie blízkej $m = 2$ bola uplatnená v experimentoch viacerých zdrojov (napr. [15] alebo [5]). Preskúmaniu vplyvu počtu mutovaných génov na kvalitu výpočtu sa venuje podkapitola 5.4

Zhrnutie inicializácie vstupných parametrov výpočtu zobrazuje tabuľka 5.2[18].

Tabuľka 5.2: Inicializácia vstupných parametrov CGP pre testovanie vplyvu veľkosti populácie na efektivitu výpočtu algoritmu.

Veľkosť populácie	λ	**1, 2, 5, 10, 20**
Počet riadkov	m_r	1
Počet stĺpcov	m_c	52/156
Previazanosť	l	52/156
Mutácia	m	2
Pozícia aktívnych uzlov (od stĺpca)	pos	1/52
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu

5.3.1 Evolučná stratégia $1 + 1$



(a) Bez redundancie

(b) S redundanciou

Obr. 5.2: Priebeh CGP pri použití evolučnej stratégie 1+1. Grafy zobrazujú vývoj veľkosti najlepšieho nájdeného riešenia v závislosti na počte vygenerovaných jedincov (os x) pre jednotlivé reštarty.

[18]Parameter *pos* určuje, od ktorej pozície v genotype jedinca (v prípade obvodu s redundantnými uzlami) budú začínať aktívne uzly inicializované kópiou referenčného obvodu.

Generácia je tvorená rodičom a jedným potomkom. Pre výber rodiča nasledujúcej generácie, ktorým nesmie byť predok aktuálnej generácie, je tak len jediná možnosť.

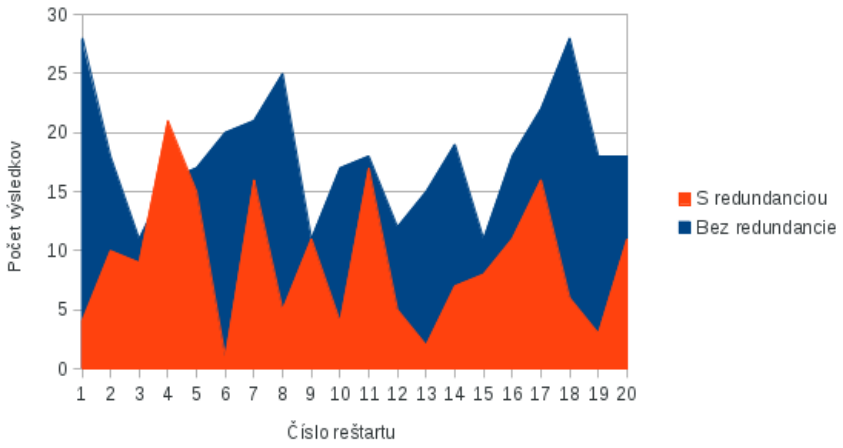
Vykonaných bolo celkom 20 reštartov. Ich priebeh je znázornený na obrázku 5.2, kde každá z kriviek reprezentuje vývoj veľkosti fenotypu najlepšieho nájdeného jedinca v priebehu jedného reštartu.

Porovnanie celkového počtu získaných riešení v priebehu výpočtu zobrazuje Obr. 5.3.

5.3.1.1 Pozorovanie

Pri generovaní len jedného nového potomka sa ukázali obidva prístupy (s redundanciou aj bez) relatívne efektívne. Oba boli schopné objaviť optimálne riešenie relatívne rýchlo (približne do $3 \cdot 10^6$ a bez redundancie už približne do 10^6 vygenerovaných jedincov). Dalo by sa skonštatovať, že výpočet nevyužívajúci redundanciu bol relatívne efektívnejší.

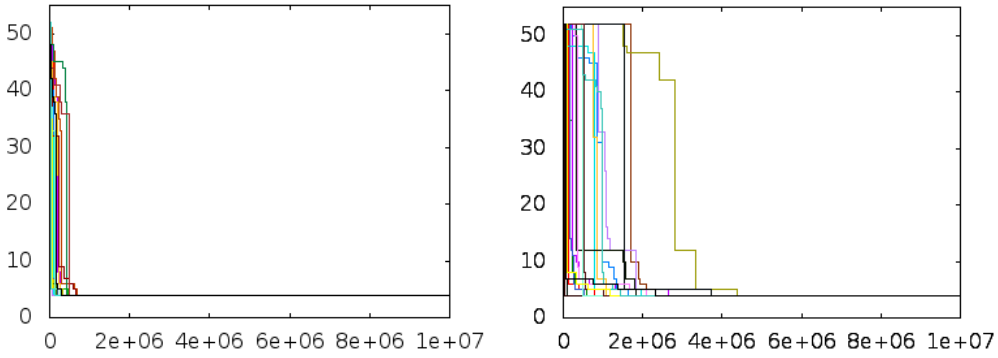
Z obrázkov 5.2 a 5.3 je tiež možné pozorovať odlišnosť v správaní sa výpočtu v závislosti na využívaní redundancie. Kým riešenie bez redundancie vrátilo takmer v každom reštarte väčší počet postupne sa zlepšujúcich riešení, pri redundancii má algoritmus väčšiu tendenciu ku „skokovým“ zlepšeniam riešenia, pričom je celkový počet nájdených zlepšení v priebehu výpočtu menší.



Obr. 5.3: Porovnanie celkového počtu získaných medzivýsledkov v priebehu výpočtu algoritmu pre $\lambda = 1$.

5.3.2 Evolučná stratégia $1 + 2$

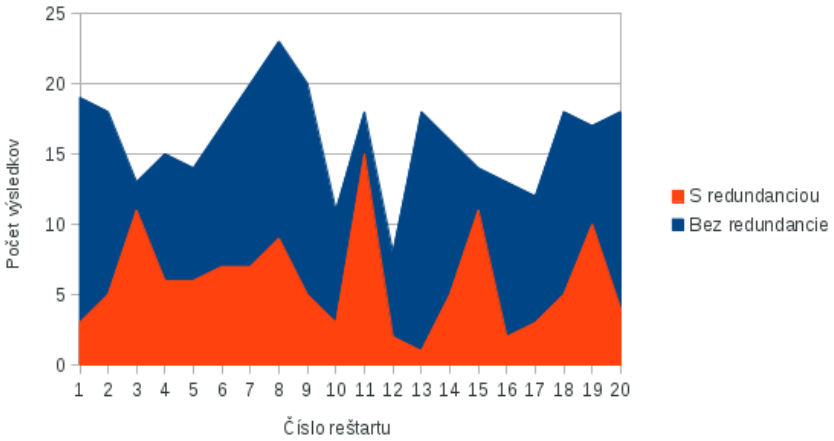
Priebeh 20 reštartov algoritmu je zobrazený na obrázku 5.4.



(a) Bez redundancie

(b) S redundanciou

Obr. 5.4: Priebeh 20 reštartov CGP pri použití evolučnej stratégie 1+2.

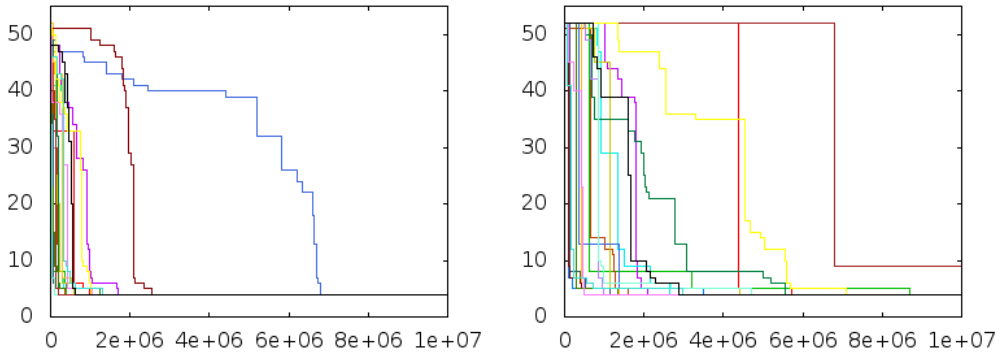


Obr. 5.5: Porovnanie celkového počtu získaných medzivýsledkov v priebehu výpočtu algoritmu pre $\lambda = 2$.

5.3.2.1 Pozorovanie

Zväčšenie generácie o jedného jedinca spôsobilo len nepatrné zhoršenie efektívnosti algoritmu pri použití prístupu bez redundancie (podrobné zhrnutie výsledkov v časti 5.3.6). Prístup s redundanciou zaznamenal rovnako zhoršenie, avšak oproti prvému prístupu o čosi výraznejšie.

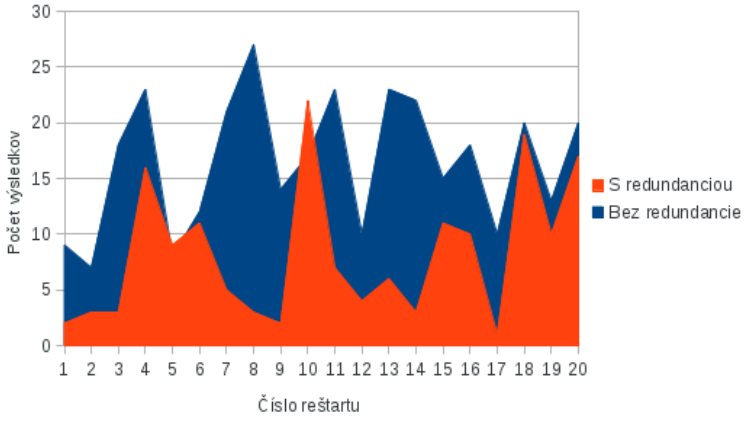
5.3.3 Evolučná stratégia $1 + 5$



(a) Bez redundancie

(b) S redundanciou

Obr. 5.6: Priebeh 20 reštartov CGP pri použití evolučnej stratégie 1+5.



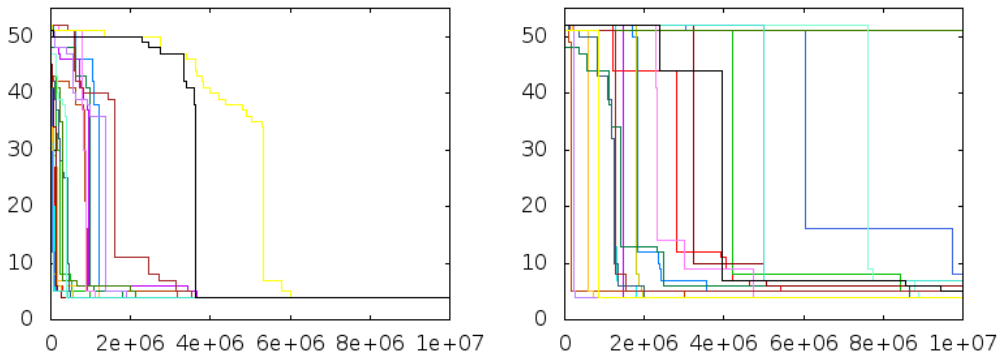
Obr. 5.7: Porovnanie celkového počtu získaných medzivýsledkov v priebehu výpočtu algoritmu pre $\lambda = 5$.

5.3.3.1 Pozorovanie

Z Obr. 5.6 je možné pozorovať zhoršenie v efektívnosti algoritmu pri zväčšení počtu jedincov generácií. Pri využití redundancie dokonca algoritmus v jednom reštarte (z celkového počtu 20) optimálne riešenie vôbec nenašiel (Obr. 5.6).

Podobne ako v predošlom meraní však pretrváva trend „skokových“ a menej častých zlepšení pri redundancii (Obr. 5.7).

5.3.4 Evolučná stratégia $1 + 10$



(a) Bez redundancie

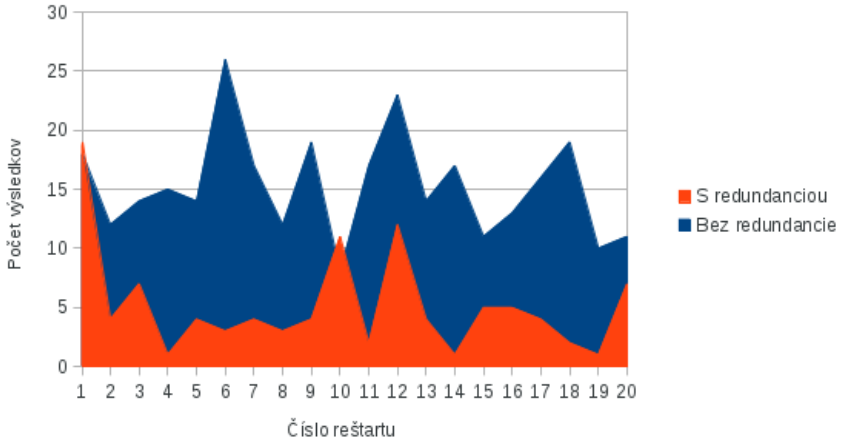
(b) S redundanciou

Obr. 5.8: Priebeh 20 reštartov CGP pri použití evolučnej stratégie 1+10.

5.3.4.1 Pozorovanie

Na Obr. 5.8 je vidieť výrazné zhoršenie výpočtovej sily algoritmu v hľadaní optimálneho riešenia. Značné zhoršenie zaznamenáva prístup s využitím redundancie, pri ktorom sa vo viacerých reštartoch optimálne riešenie vôbec nenašlo. V jedom z reštartov dokonca nebolo nájdené žiadne lepšie riešenie oproti referenčnému obvodu.

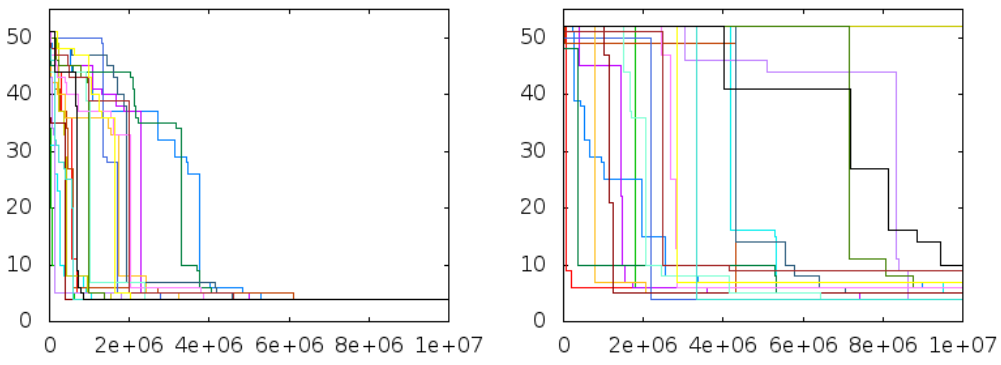
Počet vrátených riešení (Obr. 5.9) má rovnakú tendenciu ako v predošlých meraniach. Toto porovnanie však nie je úplne objektívne, keďže niektoré reštarty neskončili nájdením optimálneho riešenia.



Obr. 5.9: Porovnanie celkového počtu získaných medzivýsledkov v priebehu výpočtu algoritmu pre $\lambda = 10$.

5.3.5 Evolučná stratégia $1 + 20$

Priebeh vykonaných 20 reštartov merania s použitím generácie o veľkosti 21 jedincov je zobrazený na obrázku 5.10.



(a) Bez redundancie

(b) S redundanciou

Obr. 5.10: Priebeh 20 reštartov CGP pri použití evolučnej stratégie 1+20.

5.3.6 Zhodnotenie

Zväčšenie generácie spôsobí generovanie viacerých funkčných jedincov z rovnakého rodiča, pričom každý z týchto jedincov môže byť potenciálne najlepším nájdeným (prípadne optimálnym) riešením. Predok nasledujúcej generácie je z aktuálnej generácie vybraný náhodne, pričom sa ostatní funkční jedinci zahodia.

Vykonané testy ukázali, že zväčšenie generácie má na efektivitu výpočtu skôr nepriaznivý vplyv. Na tieto zmeny bolo citlivejšie riešenie využívajúce redundanciu, ktoré sa oproti prístupu bez redundancie zhoršovalo výraznejšie, a to až do takej miery, že v stanovenom limite (maximálne 10^7 vygenerovaných obvodov) nestihlo hneď v niekoľkých prípadoch nájsť optimálne riešenie.

Pri meraní sa tiež objavil rozdiel medzi použitými prístupmi v počte nájdených „medzivýsledkov“ pred nájdením optimálneho riešenia. Kým pri inicializácií prvého jedinca referenčným obvodom bez redundantných uzlov algoritmus konvergoval „postupne“ (väčší počet riešení s relatívne malým zlepšením), pri využití redundancie dochádzalo častejšie ku „skokovým“ zlepšeniam o väčší počet hradiel naraz. Tieto výsledky sú tiež, samozrejme, závislé na konkrétnom probléme, charaktere obvodu (jeho optimalizovateľnosti) a jeho počiatočnej reprezentácií.

Skokové zlepšenia sa objavujú aj pri použití prístupu bez redundancie (napr. Obr. 5.10 (a)), no pozorovateľné sú skôr pri väčšej veľkosti generácie (spôsobené pravdepodobne opakovaným mutovaním rovnakého predka pred jeho „zahodením“, resp. nahradením ďalším predkom) a tiež až od určitej hranice resp. po čiastočnom zlepšení obvodu (podľa Obr. 5.10 (a) približne od 30 – 25 aktívnych hradiel obvodu), kedy už obvod redundantné gény *obsahuje* (vzniknuté mutáciou).

Tabuľka 5.3: Priemerné počty vygenerovaných jedincov v priebehu výpočtu v závislosti na veľkosti gnerácie.

Lambda	**Počet vygenerovaných jedincov**	
	Bez redundancie	**S redundanciou**
1	264 815,70	1 179 617,35
2	280 509,50	1 604 815,40
5	1 069 916,65	3 006 510,20
10	1 723 239,65	6 148 354,50
20	2 684 742,75	8 900 008,00

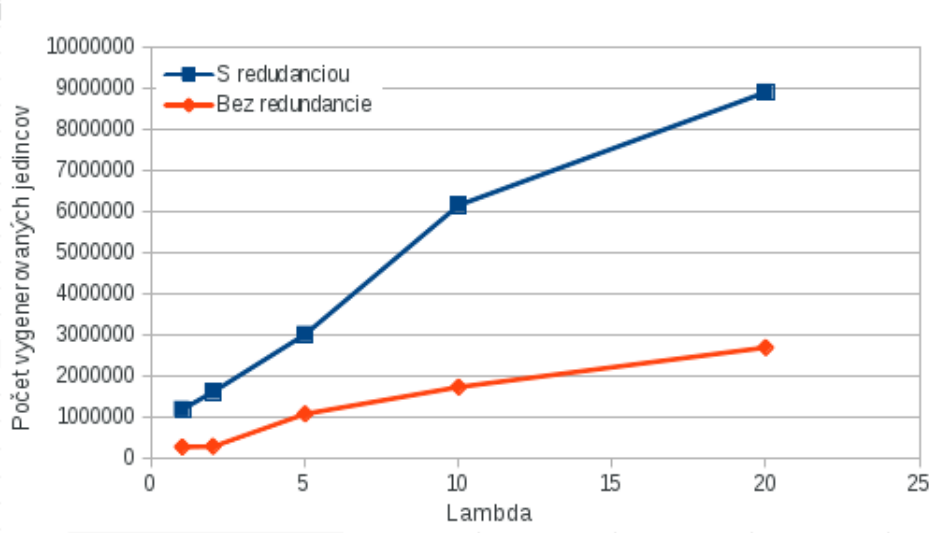
Tabuľka 5.3 a Obr. 5.11 zobrazujú vývoj priemerného počtu vygenerovaných jedincov počas priebehu algoritmu potrebných pre ukončenie algoritmu

(nájdenie optimálneho riešenia alebo ukončenia dosiahnutím maximálneho počtu vygenerovaných jedincov).

Z tohto merania je teda možné vyvodiť nasledujúce závery:

- **Zväčšenie počtu jedincov generácie CGP nemá na efektivitu výpočtu priaznivý vplyv.**
- **Zavedením redundancie sa zvyšuje potenciál výpočtu ku skokovým zlepšeniam, nemusí však zvýšiť efektivitu výpočtu.**

V ďalších meraniach sa na základe týchto výsledkov bude používať evolučná stratégia $1 + 1$, tzn. $\lambda = 1$.



Obr. 5.11: Vývoj priemerného počtu vygenerovaných jedincov pre ukončenie algoritmu v závislosti na veľkosti generácie.

5.4 Vplyv miery mutácie na výpočet algoritmu

Mierou mutácie je vyjadrené, koľkokrát sa v genotype jedinca vyberie náhodný gén, ktorý sa následne náhodne zmení v medziach platných obmedzení (viď časť 3.1.4)[19].

[19]Podrobnejšie o mutácií v časti 3.2.2.

Pre účely merania boli stanovené miery mutácie *absolútne*, a to postupne na hodnoty **1, 2, 3, 5, 7** a **10**. Meranie bolo opäť vykonané na dvoch prístupoch inicializácie počiatočného jedinca, a to *bez využitia* a *s využitím redundancie* ($r = 1, c = 52/156$). Pr každú konfiguráciu vstupných parametrov bolo vykonaných celkom 20 reštartov.

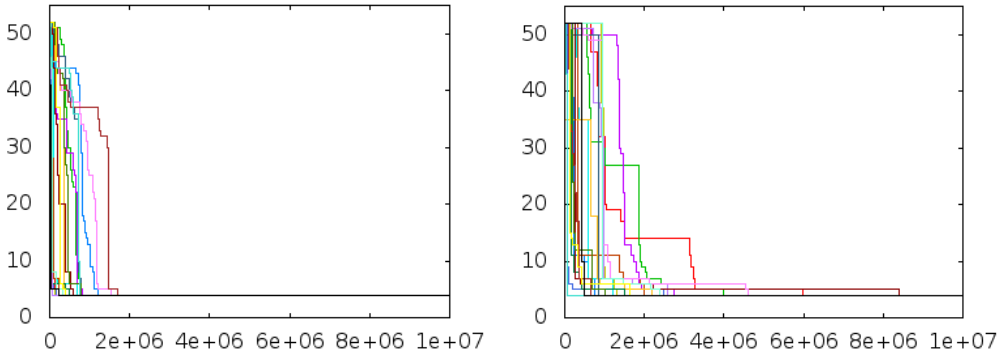
Inicializácia vstupných parametrov je zhrnutá v tabuľke 5.4.

Tabuľka 5.4: Inicializácia vstupných parametrov CGP pre testovanie vplyvu miery mutácie na efektivitu výpočtu algoritmu.

Veľkosť populácie	λ	1
Počet riadkov	m_r	1
Počet stĺpcov	m_c	52/156
Previazanosť	l	52/156
Mutácia	m	**1, 2, 3, 5, 7, 10**
Pozícia aktívnych uzlov (od stĺpca)	pos	1/52
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu

5.4.1 Mutácia 1 génu genotypu

Priebeh meraní je zobrazený na Obr. 5.12.



(a) Bez redundancie

(b) S redundanciou

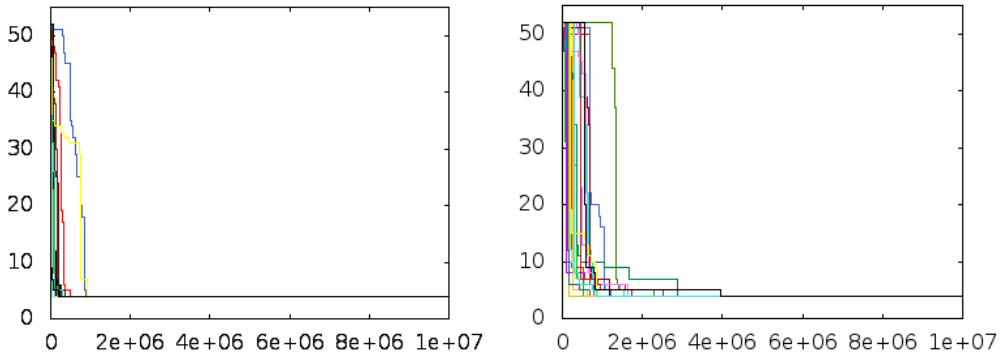
Obr. 5.12: Priebeh 20 reštartov CGP pri mutácií 1 génu genotypu.

5.4.1.1 Pozorovanie

Obidva prístupy dokázali v každom meraní objaviť optimálne riešenie. V niekoľkých reštartoch však bolo menej efektívne riešenie využívajúce redundanciu, ktoré potrebovalo pre nájdenie optimálneho riešenia vygenerovať v dvoch reštartoch viac ako $5 \cdot 10^6$ jedincov, z toho v jednom až viac ako $8 \cdot 10^6$ jedincov.

Pre nájdenie optimálneho riešenia potreboval prístup bez redundancie priemerne **611 828** vygenerovaných jedincov, pričom prístup s redundanciou až **2 342 129,7** jedincov.

5.4.2 Mutácia 2 génov genotypu



(a) Bez redundancie

(b) S redundanciou

Obr. 5.13: Priebeh 20 reštartov CGP pri mutácií 2 génov genotypu.

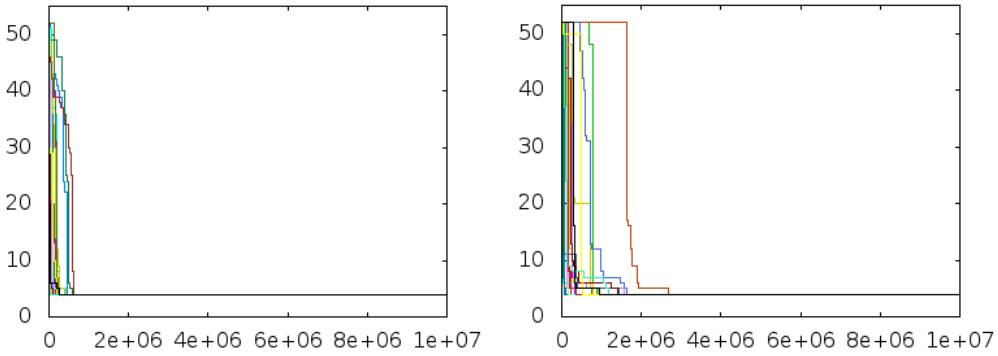
5.4.2.1 Pozorovanie

Pri zvýšení mutácie na dva gény genotypu pozorujeme zlepšenie efektivity obidvoch použitých prístupov.

Priemerný počet vygenerovaných jedincov potrebný pre nájdenie optimálneho riešenia je pre prístup bez redundancie **264 815,7** jedincov, pre redundanciu **1 429 034,05** jedincov.

5.4.3 Mutácia 3 génov genotypu

5.4.3.1 Pozorovanie



(a) Bez redundancie

(b) S redundanciou

Obr. 5.14: Priebeh 20 reštartov CGP pri mutácií 3 génov genotypu.

Ďalším zvýšením mutácie sa grafy javia ešte „užšie“ ako v predošlom prípade. Väčšina reštartov teda našla optimálne riešenie ešte efektívnejšie. Pri využití redundancie postačovalo na nájdenie optima (s výnimkou jedného reštartu) vygenerovanie menej ako $2 \cdot 10^6$ jedincov, pričom v priemere bolo potrebných **830 758,3** vygenerovaných jedincov. Prístupu bez redundancie postačovalo dokonca priemerne len **222 205,15** vygenerovaných jedincov.

5.4.4 Mutácia 5 génov genotypu

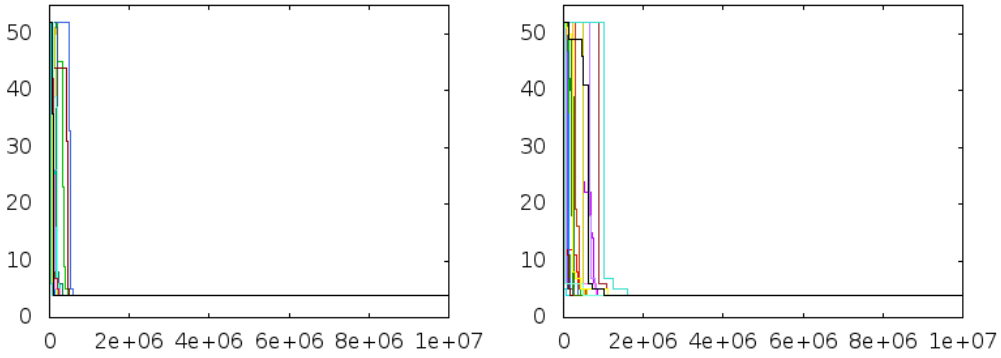
5.4.4.1 Pozorovanie

Na obrázku 5.15 je vidieť veľmi priaznivý vplyv ďalšieho zvýšenia mutácie, ktorý zaznamenali obidva prístupy. Optimálne riešenie bolo nájdené priemerne po **225 453,8** (5.15 (a)), resp. **596 858,35** (5.15 (b)) vygenerovaných jedincoch.

5.4.5 Mutácia 7 génov genotypu

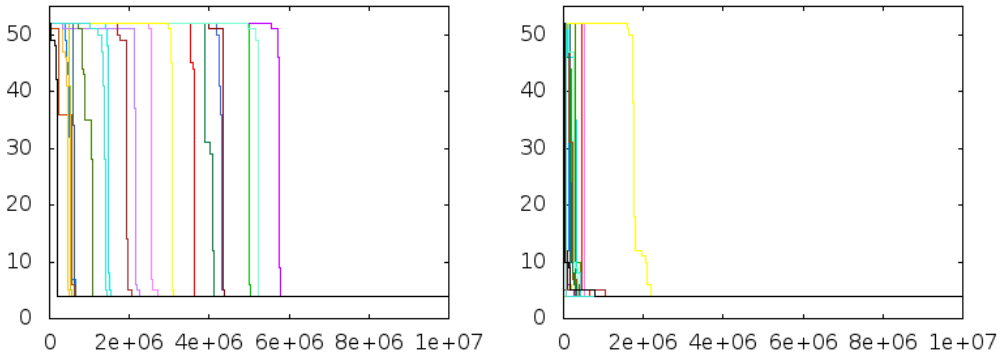
5.4.5.1 Pozorovanie

Pri ďalšom navýšení miery mutácie na 7 mutovaných génov genotypu jedinca bol zaznamenaný prekvapivý vplyv na skúmané prístupy. Kým prístup



(a) Bez redundancie (b) S redundanciou

Obr. 5.15: Priebeh 20 reštartov CGP pri mutácií 5 génov genotypu.



(a) Bez redundancie (b) S redundanciou

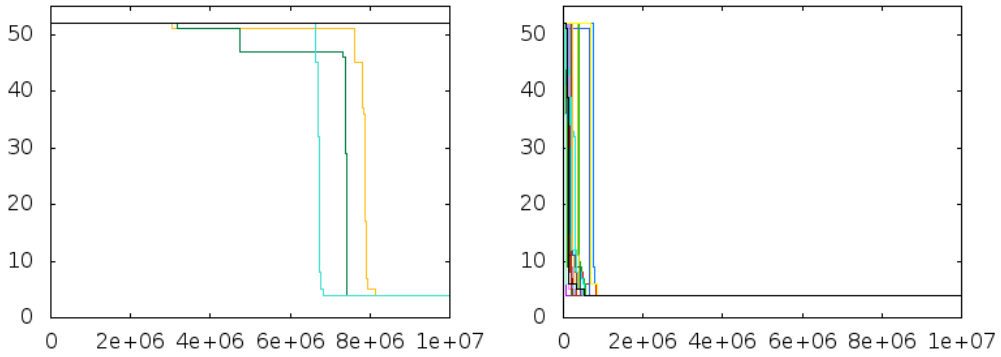
Obr. 5.16: Priebeh 20 reštartov CGP pri mutácií 7 génov genotypu.

zavádzajúci redundanciu už pri inicializácií dosiahol opäť vyššej efektivity výpočtu ako v predošlom meraní (priemerne **479 886,85** vygenerovaných jedincov pre nájdenie optima), prístup inicializovaný bez redundantných génov, naopak, zaznamenal *výrazný pokles* efektivity – priemerne potreboval pre nájdenie optimálneho riešenia problému vygenerovať až **2 496 074,55** jedincov.

Pre toto nastavenie parametrov sa prístup využívajúci redundanciu ukázal po prvýkrát ako *efektívnejší*.

5.4.6 Mutácia 10 génov genotypu

Priebeh výpočtu znázorňuje obrázok 5.17.



(a) Bez redundancie

(b) S redundanciou

Obr. 5.17: Priebeh 20 reštartov CGP pri mutácií 10 génov genotypu.

5.4.6.1 Pozorovanie

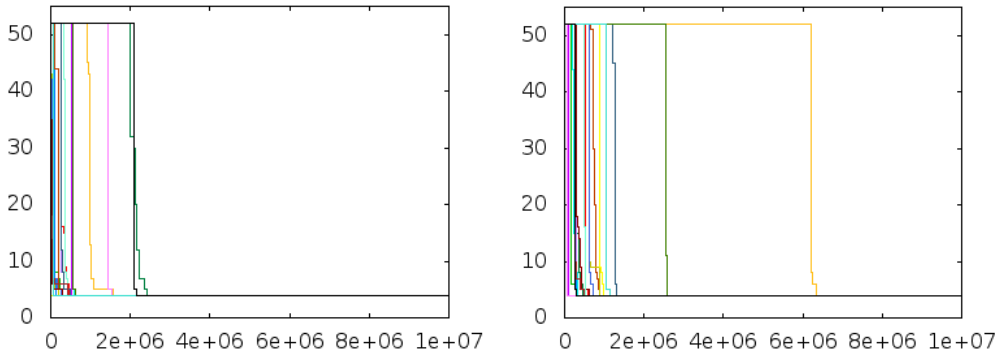
Zvýšením miery mutácie na 10 génov došlo k ďalšiemu rapídnemu zhoršeniu efektivity výpočtu inicializovaného bez redundantných uzlov. Z celkovo 20 reštartov obmedzených na maximálne 10^7 vygenerovaných jedincov sa len trom podarilo nájsť optimálne riešenie. Zvyšných 17 reštartov nenašlo žiadne, oproti referenčnému obvodu ani čiastočne lepšie riešenie. Z iného uhla pohľadu je možné skonštatovať, že všetky reštarty algoritmu inicializovaného bez redundancie, ktoré našli akékoľvek lepšie riešenie (čím sa vo fenotype jedinca objavili redundantné uzly), skončili nájdením optimálneho riešenia.

Oproti tomu, prístup využívajúci redundanciu dosiahol zvýšením miery mutácie ešte lepšej efektívnosti ako v predošlom meraní. Všetky reštarty tohto prístupu skončili nájdením optimálneho riešenia, pričom na to potrebovali priemerne **439 017,3** vygenerovaných jedincov. Pri prvom prístupe algoritmus skončil priemerne po vygenerovaní **9 618 831,45** jedincov (pre neúspešné pokusy sa uvažuje maximálny počet 10^7 vygenerovaných jedincov).

5.4.7 Doplňujúce meranie: Mutácia 12 a 15 génov genotypu

Na základe predošlých výsledkov a pretrvávajúcom priaznivom vplyve zvyšovania miery mutácie na efektívnosť prístupu využívajúceho redundanciu boli vykonané ešte dve doplňujúce merania, aby sa preskúmala zmena efektívnosti výpočtu vplyvom ďalšieho navyšovania miery mutácie a bolo tak možné určiť optimálnu mieru mutácie pre daný problém a veľkosť jednca.

Merania boli vykonané na prístupe využívajúcom redundanciu, pričom miera mutácie bola stanovená na 12 a 15 génov.



(a) Mutácia 12 génov s redundanciou (b) Mutácia 15 génov s redundanciou

Obr. 5.18: Priebeh 20 reštartov CGP pri mutácií 12 (a) a 15 (b) génov genotypu.

5.4.7.1 Pozorovanie

Ako je možné vidieť na Obr. 5.18, ďalšie zvýšenie mutácie na 12 resp. 15 génov spôsobilo oproti predošlému meraniu mierny pokles efektívnosti. Pre nájdenie optimálneho riešenia bolo potrebné vygenerovať v priemere **660 596,65** jedincov pre 12 mutovaných génov a **966 880,8** jedincov pre mieru mutácie 15.

5.4.8 Zhodnotenie

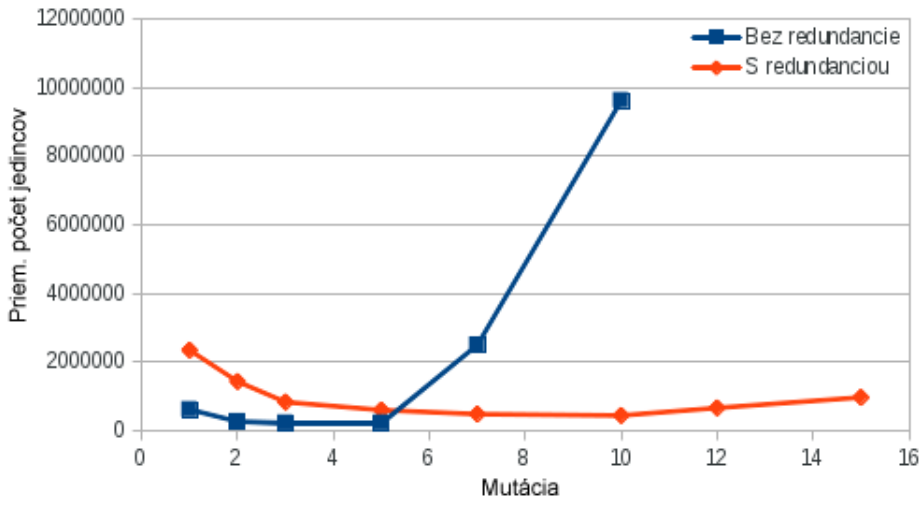
Výsledky meraní poukazujú na výrazný vplyv stanovenej miery mutácie na efektívnosť algoritmu. Kým mutácia jedného génu genotypu potrebuje

na komplexnejšiu zmenu vnútornej štruktúry relatívne väčší počet iterácií (Obr. 5.12), už pri malom zvýšení počtu mutácií je možné pozorovať zlepšenie v efektívnosti oboch použitých prístupov (obrázky 5.13 – 5.15).

Podobne ako príliš málo aplikovaných mutácií má na priebeh výpočtu nepriaznivý vplyv aj príliš veľká miera mutácie. Tá spôsobí, že pri generovaní jedincov generácie zmutuje náhodne príliš veľa génov rodiča. Pravdepodobnosť, že takto vzniknutý jedinec bude opäť funkčným obvodom sa tým radikálne znižuje.

Tento jav je možné pozorovať na veľmi výraznom zhoršení efektívnosti prístupu nevyužívajúceho redundanciu pri zvýšení mutácie na 7 a 10 génov (Obr. 5.19). Toto zhoršenie je spôsobené tým, že jedinec v priebehu výpočtu obsahuje veľmi málo redundantných uzlov (pri inicializácií žiadne), ktoré by mutácie absorbovali. Mutáciou je tak postihnutá priveľká časť aktívnych uzlov.

Naopak, pri využití redundancie sa zvýšenie mutácie ukázalo ako výhodné z hľadiska zvyšovania efektívnosti výpočtu. Najlepšiu efektívnosť sa podarilo dosiahnúť pri mutovaní 10 génov. Ďalším zvýšením mutície však začala efektivita opäť klesať, čo je možné pozorovať na zvyšovaní sa priemerného počtu vygenerovaných jedincov potrebného pre nájdenie oprimálneho riešenia (Obr. 5.19).



Obr. 5.19: Vývoj priemerného počtu vygenerovaných jedincov v závislosti na miere mutácie.

Toto zhoršenie však nie je také dramatické ako v prípade inicializácie algoritmu bez redundancie, a to vďaka absorbcii mutácií práve neaktívnymi resp. redundantnými uzlami obvodu.

Zhrnutie priemerného počtu vygenerovaných jedincov v jednotlivých experimentoch je zobrazené v tabuľke 5.5 (dátový základ pre Obr. 5.19).

Tabuľka 5.5: Priemerné počty vygenerovaných jedincov v priebehu výpočtu v závislosti na miere mutácie.

Mutácia	**Počet vygenerovaných jedincov**	
	Bez redundancie	**S redundanciou**
1	611 828,30	2 342 129,70
2	264 815,70	1 429 034,05
3	222 205,15	830 758,30
5	225 453,80	596 858,35
7	2 496 074,55	479 886,85
10	9 618 831,45	439 017,30
12	-	660 596,65
15	-	966 880,80

Na základe tohto merania je možné skonštatovať **priaznivý vplyv redundancie na efektivitu výpočtu**, najmä **v kombinácií s relatívne vyššou mierou mutácie**.

Výhoda tejto kombinácie spočíva v potenciáli objaviť riešenia vyžadujúce komplexnejšiu zmenu štruktúry obvodu (na rozdiel od prístupu s nízkou mierou mutácie bez využitia redundancie, ktorý ponúkal taktiež relatívne dobré výsledky – Obr. 5.12).

Ako najlepšie sa pre riešenie tejto úlohy osvedčilo nastavenie miery mutácie na 10 pri obvode inicializovanom s 200% redundanciou (104 redundantných uzlov). V ďalších experimentoch preto budú ako miery mutácie skúmané nasledujúce hodnoty:

- Absolútna hodnota miery mutácie 10,
- približne 6,4 – 6,5% veľkosti fenotypu obvodu,
- približne 9,5 – 9,6% počtu redundantných uzlov

Vzťahu optimálnej miery mutácie a veľkosti fenotypu sa bude podrobnejšie venovať časť 5.5.

5.5 Analýza závislosti optimálnej miery mutácie na veľkosti fenotypu

Pri meraní vplyvu miery mutácie na výpočet algoritmu (skúmaný v predošlej časti) bola ukázaná závislosť efektívnosti výpočtu algoritmu na stanovenej mutácii (Obr. 5.19). Ako optimálna miera mutácie pre danú veľkosť jedinca a redundanciu bola experimentálne určená hodnota 10 génov, čo v danom prípade zodpovedalo približne 6,5% veľkosti fenotypu jedinca, resp. približne 9,5% počtu jeho redundantných uzlov po inicializácií.

Na základe týchto výsledkov je v tejto časti skúmaný vzťah medzi mierou mutácie a veľkosťou fenotypu jedinca za účelom upresnenia a zovšeobecnenia poznatkov získaných v časti 5.4.

5.5.1 Inicializácia

Analýza bola vykonaná opäť použitím logickej funkcie **xor5**. Testované boli dve veľkosti fenotypu jedinca $2 \times 3n$ a $3 \times 3n$ (viď tabuľka 5.6).

Tabuľka 5.6: Prehľad testovacích obvodov pre testovanie vzťahu medzi mutáciou a veľkosťou fenotypu jedinca.

Topológia ($m_r \times m_c$)	**Počet uzlov**	**6,5%**	**Redundancia**		
			Miera	**Počet uzlov**	**9,5%**
$1 \times 3n$	156	**≈10**	200%	104	**≈10**
$2 \times 3n$	312	**≈20**	500%	260	**≈25**
$3 \times 3n$	468	**≈30**	800%	416	**≈40**

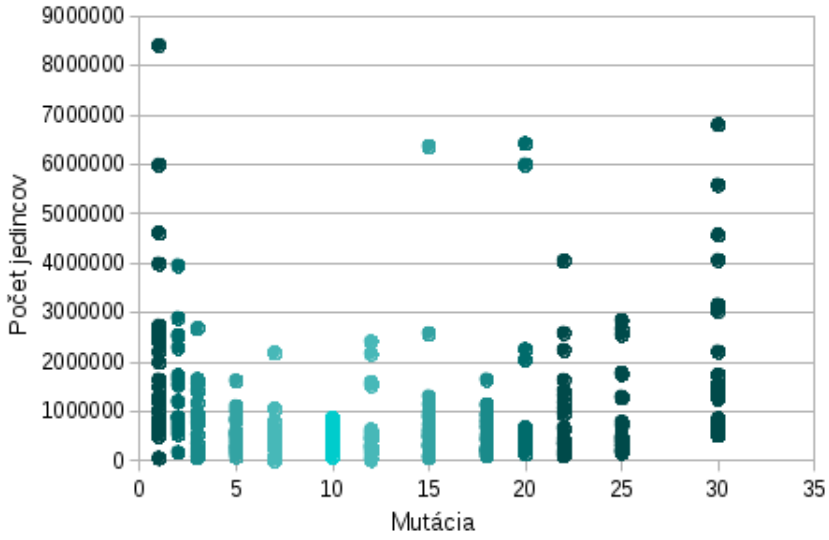
Tabuľka 5.7: Inicializácia vstupných parametrov CGP pre skúmanie vzťahu medzi veľkosťou jedinca a optimálnou mierou mutácie.

Veľkosť populácie	λ	1
Počet riadkov	m_r	**1/2/3**
Počet stĺpcov	m_c	$3n$
Previazanosť	l	l_{max}
Mutácia	m	(rôzne)
Pozícia aktívnych uzlov	pos	uprostred fenotypu
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu

Pre výpočet bola použitá evolučná stratégia $1 + 1$ a inicializácia bloku funkčných uzlov v strede fenotypu jedinca. Pre každú konfiguráciu vstupných premenných bolo vykonaných opäť spolu 20 reštartov.

Vstupné parametre experimentu sú zhrnuté v tabuľke 5.7.

5.5.2 Miera redundancie 200%



Obr. 5.20: Prehľad počtu vygenerovaných jedincov pre rôzne miery mutácie pri 200% redundancii.

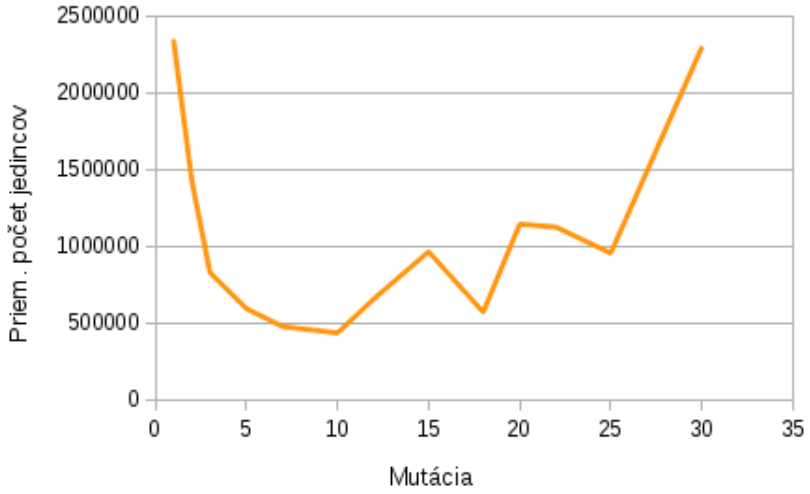
200% redundancia bola dosiahnutá doplnením riadku referenčného obvodu o redundantné uzly (topológia $1 \times 3n$). Ide vlastne o konfiguráciu skúmanú v časti 5.4 doplnenú o ďalšie merania.

Počty vygenerovaných jedincov v priebehu všetkých 20 reštartov pre každú skúmanú hodnotu mutácie znázorňuje Obr. 5.20.

Na Obr. 5.21 je znázornený vývoj priemerného počtu vygenerovaných jedincov v závislosti na miere mutácie.

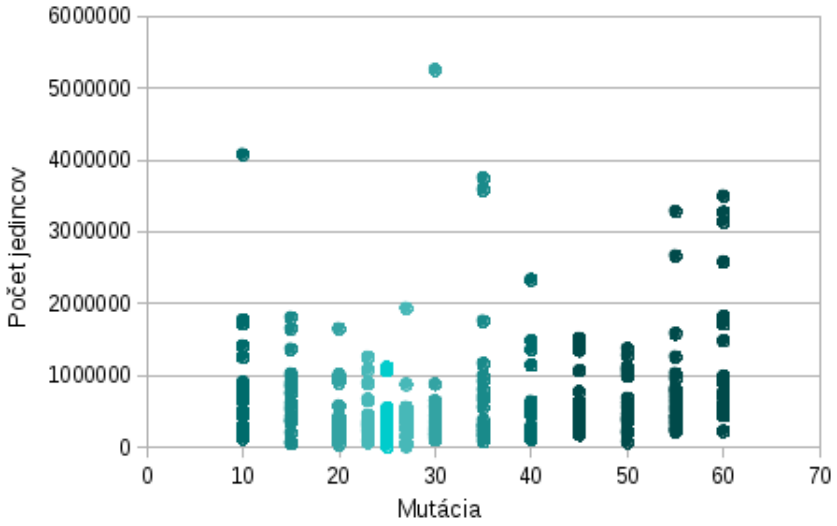
5.5.3 Miera redundancie 500%

Doplnením obvodu použitom v predošlom meraní o druhý riadok tvoria redundantné uzly celkom 5/6 celkového počtu uzlov obvodu (topológia $2 \times 3n$).

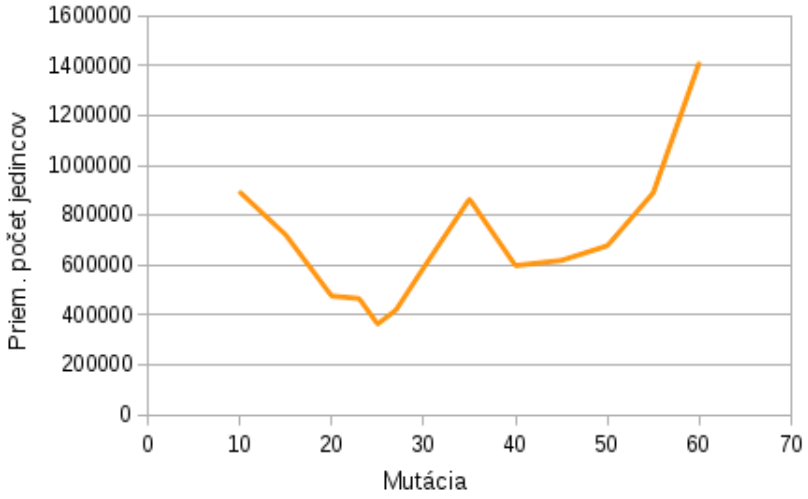


Obr. 5.21: Vývoj priemerného počtu vygenerovaných jedincov v závislosti na miere mutácie pri 200% redundancii.

Priebeh výpočtu je znázornený na Obr. 5.22 (počet vygenerovaných jedincov pre jednotlivé reštarty) a Obr. 5.23 (vývoj priemerného počtu vygenerovaných jedincov).



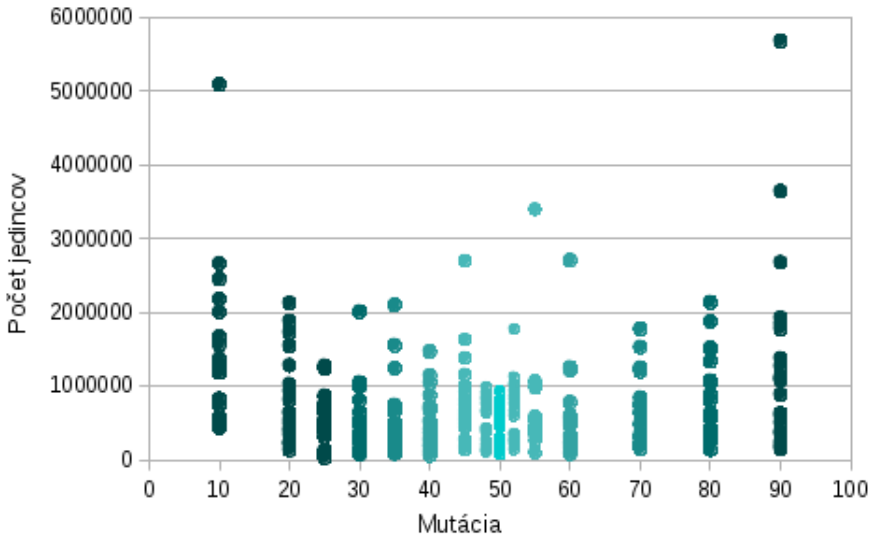
Obr. 5.22: Prehľad počtu vygenerovaných jedincov pre rôzne miery mutácie pri 500% redundancii.



Obr. 5.23: Vývoj priemerného počtu vygenerovaných jedincov v závislosti na miere mutácie pri 500% redundancii.

5.5.4 Miera redundancie 800%

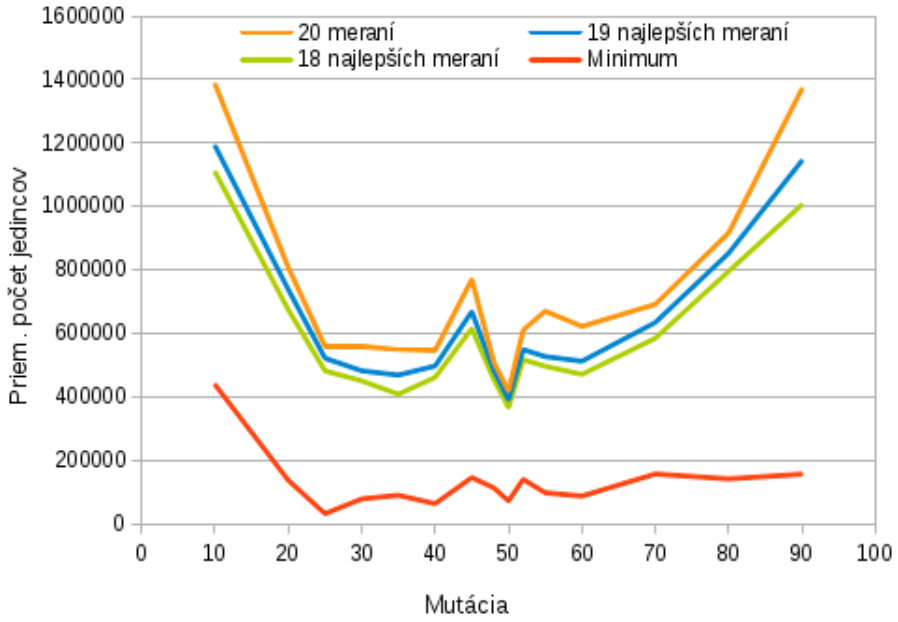
Pridaním tretieho riadku fenotypu jedinca tvoria redundantné uzly 8/9 všetkých uzlov fenotypu (topológia $3 \times 3n$).



Obr. 5.24: Prehľad počtu vygenerovaných jedincov pre rôzne miery mutácie pri 800% redundancii.

Obr. 5.24 opäť znázorňuje počty vygenerovaných jedincov v jednotlivých

reštartoch výpočtu. Na Obr. 5.25 je znázornený vývoj priemerného počtu vygenerovaných jedincov pre všetkých 20 reštarov jednotlivých meraní, ako aj pre 19 a 18 najlepších reštartov. V grafe je tiež znázornený minimálny počet vygenerovaných jedincov pre jednotlivé miery mutácie.



Obr. 5.25: Vývoj priemerného počtu všetkých 20 a zároveň 19 a 18 najlepších vygenerovaných jedincov v závislosti na miere mutácie pri 800% redundancii. Červeným grafom je znázornený vývoj minimálneho počtu vygenerovaných jedincov pre jednotlivé miery mutácie.

5.5.5 Zhodnotenie

Všetky vykonané merania naznačujú rovnakú tendenciu vývoja efektívnosti výpočtu algoritmu pri zmene miery mutácie – príliš nízka miera mutácie *nestačí* v genotypoch jedincov generovať dostatočné zmeny pre zabezpečenie relatívne efektívneho výpočtu. Stanovenie príliš veľkej mutácie spôsobí zníženie pravdepodobnosti, že zmutovaný jedinec bude opäť funkčný (znižuje sa pravdepodobnosť, že mutáciu „absorbujú“ redundantné uzly). Efektívnosť výpočtu tak opäť klesá. Tento trend je možné pozorovať vo všetkých získaných výsledkoch.

V ideálnom prípade by teda mal priemerný počet vygenerovaných jedincov klesať so zvyšovaním miery mutácie až do určitej optimálnej hodnoty,

pričom ďalšie zvýšenie mutácie by spôsobilo opätovné zvyšovanie počtu vygenerovaných jedincov, a tým zhoršenie efektívnosti výpočtu algoritmu.

Vo výsledkoch jednotlivých meraní však dochádza k neočakávaným zhoršeniam alebo zlepšeniam efektívnosti algoritmu (viď obrázky 5.21, 5.23 a 5.25). Tieto náhle „skoky“ v priemernom počte vygenerovaných jedincov sú spravidla spôsobené niekoľkými (približne 1–3) reštartami, ktoré dopadli výrazne horšie ako väčšina reštartov rovnakého merania (obrázky 5.20, 5.22 a 5.24). Príčinou tohto javu je zásadný vplyv náhody, ako hlavného princípu jediného evolučného faktora CGP – mutácie na výpočet algoritmu.

Neuvažovaním týchto krajných prípadov je vo výsledkoch do určitej miery pozorovateľný očakávaný vývoj efektívnosti algoritmu v závislosti na miere mutácie a je možné približne určiť rozsah, v ktorom sa nachádza optimálna miera mutácie pre daný problém. Výsledky meraní sú zhrnuté v tabuľkách 5.8 a 5.9).

Na Obr. 5.25 je zobrazený vývoj *minimálneho* počtu vygenerovaných jedincov zo všetkých reštartov pre jednotlivé miery mutácie (červený graf). Tento vývoj má podobnú tendenciu ako priemerný počet vygenerovaných jedincov za všetky vykonané reštarty, avšak na zmenu miery mutácie reaguje výrazne miernejšie. To opäť dokazuje významný vplyv náhody, ktorá relatívne vysokej efektivite výpočtu pri „neoptimálnej“ miere mutácie *nezabraňuje*, no znižuje jej pravdepodobnosť.

Tabuľka 5.8: Prehľad optimálnych mier mutácie pre jednotlivé merania.

Redundancia	**Najefektívnejší výsledok**	
	Mutácia	**Priem. počet jedincov**
200%	10	439 017,30
500%	25	364 107,60
800%	50	418 239,25

Tabuľka 5.9: Približný rozsah optimálnej miery mutácie pre jednotlivé merania.

Redundancia	**Optimálna mutácia**		
	Mutácia	**Nárast**	**% redundantných uzlov**
200%	≈10	10	≈9,6%
500%	≈25–30	15–20	≈9,6–11,5%
800%	≈48–50	18–20	≈11,5–12%

V tabuľke 5.9 je možné pozorovať zaujímavý vzťah medzi potenciálne optimálnou mierou mutácie a mierou redundancie: kým v prvom prípade

(200% redundancia) bol výpočet algoritmu najefektívnejší pri stanovení miery mutácie približne na 10 génov, opakovaným zvýšením redundancie o 300% na 500, resp. 800%, došlo k zvýšeniu potenciálne optimálnej miery mutácie o 15-20, resp. 18-20 génov. **Na každých 100% redundancie (52 uzlov) tak vo všetkých prípadoch pripadá mutácia približne 6 génov genotypu jedinca, čo je približne 11,5% počtu redundantných uzlov**. Preskúmaním univerzálnosti tohto vzťahu pri použití iných referenčných obvodov sa zaoberá časť 5.6.

Na základe získaných výsledkov je možné skonštatovať nasledujúce pozorovania:

- **Vhodným nastavením mutácie pri zvýšení redundancie je možné rádovo zachovať efektívnosť algoritmu** (tabuľka 5.8).
- **Nastavenie miery mutácie na príliš nízku alebo vysokú hodnotu má negatívny dopad na efektívnosť výpočtu algoritmu.**
- **Optimálna miera mutácie závisí na veľkosti fenotypu jedinca**.

5.6 Analýza závislosti výpočtovej sily a optimálnej mutácie na referenčnom obvode

V časti 5.5 bola skúmaná závislosť optimálnej miery mutácie na veľkosti fenotypu použitím referenčného obvodu *xor5*. V tejto časti sú získané výsledky otestované na iných problémoch z dôvodu preskúmania vývoja optimálnej miery mutácie nezávisle na použitom referenčnom obvode.

Pre účely experimentu boli použité obvody *mux* a *rd53*.

5.6.1 Testovacie problémy a inicializácia algoritmu

Výpočet bol vo všetkých prípadoch inicializovaný s 200% mierou redundancie (topológia $1 \times 3n$) a evolučnou stratégiou $1 + 1$. Pre každý referenčný obvod bola skúmaná efektívnosť algoritmu za použitia niekoľkých rôznych nastavení mutácie.

Pre obvod **mux** (referenčný obvod zložený zo 136 hradiel) bolo skúmaných spolu až 9 rôznych mier mutácie medzi hodnotami 5 a 35 (11,5% počtu redundantných uzlov ≈ 31). Pre každú hodnotu mutácie boli realizované celkom 3 reštarty s cieľom nájsť *optimálnu* realizáciu danej logickej funkcie, ktorá by mala pozostávať zo 46 hradiel.

Obvod **rd53** realizuje svoju logickú funkciu použitím 93 hradiel. Kvôli štruktúre referenčného obvodu je však pre nájdenie optimálneho riešenia (pozostávajúceho z 21 hradiel) potrebná komplexná zmena štruktúry celého obvodu. Schopnosť algoritmu poradiť si s takýmto problémom bola skúmaná použitím dvoch reštartov pre každú z piatich rôznych mier mutácie medzi hodnotami 5–25 (11,5% počtu redundantných uzlov ≈ 21). Meranie bolo zamerané na preskúmanie schopnosti algoritmu nájsť *akýkoľvek menší* ekvivalentný obvod pri vygenerovaní maximálne 10^6 jedincov.

Podrobné detaily inicializácie vstupných parametrov v tomto experimente zobrazuje tabuľka 5.10.

Tabuľka 5.10: Inicializácia vstupných parametrov CGP pre analýzu závislosti výpočtovej sily algoritmu a optimálnej mutácie na vstupných dátach.

Testovací obvod	-	**mux**	**rd53**
Veľkosť populácie	λ	1	
Počet riadkov	m_r	1	
Počet stĺpcov	m_c	$3n$	
Previazanosť	l	l_{max}	
Mutácia	m	5/10/12/15/18 20/25/30/35	5/10/15 20/25
Pozícia aktívnych uzlov	*pos*	uprostred fenotypu	
Max. vygenerovaných jedincov	-	10^7	10^6
Inicializácia redundantných génov	-	fragmenty ref. obvodu	

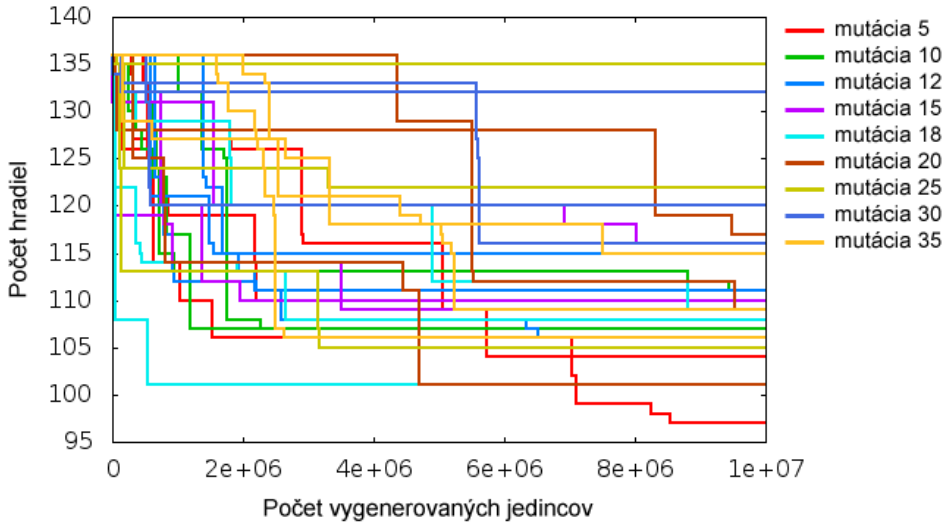
5.6.2 Experiment

Priebeh výpočtu pre obvod **mux** je zobrazený na Obr. 5.26. Obrázok 5.27 zobrazuje priemerné počty hradiel najlepších nájdených realizácií obvodu pre jednotlivé mutácie po 10^6 vygenerovaných jedincoch.

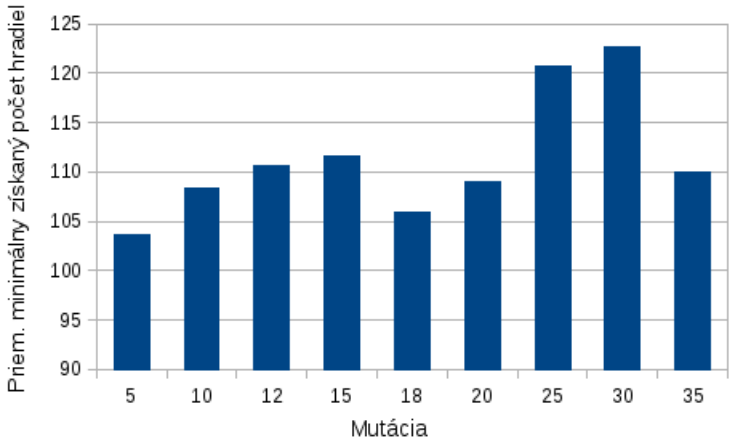
Výpočet s referenčným obvodom **rd53** nenašiel v priebehu 10^6 vygenerovaných jedincov oproti referenčnému obvodu *žiadne lepšie riešenie* v žiadnom z celkovo desiatich vykonaných reštartov (5-krát 2 reštarty).

5.6.3 Zhodnotenie

Pri optimalizácii obvodu *mux* sa nepodarilo v stanovenom limite 10^7 vygenerovaných jedincov nájsť optimálny obvod pozostávajúci zo 46 hradiel *ani v jednom* z vykonaných reštartov. Na Obr. 5.26 je možné pozorovať širokú škálu veľkostí najlepších nájdených obvodov po ukončení výpočtu:



Obr. 5.26: Priebeh jednotlivých reštartov pri optimalizácií obvodu *mux*. Farebne sú rozlíšené jednotlivé hodnoty mutácie, pričom pre každú hodnotu boli vykonané 3 reštarty.



Obr. 5.27: Priemerné počty hradiel obvodu *mux* po ukončení výpočtu pre jednotlivé mutácie.

od obvodu tvoreného 97 hradlami (mutácia 5, zlepšenie o 39 hradiel), až po obvod tvorený 135 hradlami (mutácia 25, zlepšenie o 1 hradlo). Algoritmus teda v stanovenom limite nestihol skonvergovať.

Na základe vývoja efektívnosti optimalizácie obvodu *xor5* v závislosti na veľkosti fenotypu (časť 5.5) bola optimálna miera mutácie v prípade obvodu *mux* predpokladaná blízko hodnoty 30. Tento predpoklad sa však

nepotvrdil, čo je vidieť na Obr. 5.27, kde sú zobrazené priemerné veľkosti najlepších nájdených riešení za všetky reštarty výpočtov pre jednotlivé mutácie.

Najlepšie riešenie bolo nájdené pri *najmenšej* aplikovanej mutácii (mutácia piatich génov genotypu jedinca), pričom veľkosť najlepších riešení nájdených pri ostatných skúmaných mutáciách (až na tri prípady – 18, 20 a 35) so zvyšovaním mutácie rastie. Z tohto pozorovania však nie je možné vyvodzovať objektívne vzťahy medzi mutáciou a výpočtovou silou algoritmu, nakoľko v žiadnom z vykonaných reštartov nebolo nájdené optimálne riešenie daného problému.

Pri optimalizácii referenčného obvodu *rd53* bola veľkosť fenotypu jedinca oproti predošlému meraniu približne o 31,6% menšia. Z toho dôvodu by sa dal očakávať efektívnejší výpočet algoritmu ako pri obvode *mux*. Pri meraní však žiadny z vykonaných reštartov pre jednotlivé mutácie (5, 10, 15, 20 a 25) nenašiel oproti referenčnému obvodu v priebehu 10^6 vygenerovaných jedincov *žiadny menší* ekvivalentný obvod. Príčinou tohto výsledku je pravdepodobne štruktúra referenčného obvodu, ktorá pre nájdenie optimálneho riešenia vyžaduje komplexnú reorganizáciu. Pravdepodobnosť nájdenia optimálneho riešenia sa tak blíži pravdepodobnosti *náhodnému vygenerovaniu* optimálneho obvodu v neaktívnej časti fenotypu.

Výsledky získané týmto experimentom teda poukazujú na **závislosť výpočtovej sily, resp. efektívnosti algoritmu na vnútornej štruktúre referenčného obvodu**.

5.7 Vplyv pozície aktívnych uzlov vo fenotype jedinca na výpočet algoritmu

Pri generovaní prvého jedinca sú jeho uzly inicializované hradlami referenčného obvodu. V prípade inicializácie s redundanciou, kedy je počet uzlov jedinca väčší ako počet hradiel referenčného obvodu, je možných viacero spôsobov rozmiestnenia aktívnych uzlov vo fenotype jedinca. Vplyv rôznych stratégií rozmiestnenia bude skúmaný v tejto časti.

Algoritmus CGP vymedzuje jednotlivým uzlom obmedzenia vyplývajúce z ich umiestnenia v určitom stĺpci mriežky fenotypu. Umiestnenie uzla v konkrétnom riadku daného stĺpca však na výpočet nemá žiadny vplyv. Všetky uzly daného stĺpca, a teda aj celé riadky fenotypu, sú si z hľadiska výpočtu navzájom rovnocenné a zameniteľné. Z toho dôvodu sú aktívne uzly po inicializácii rozmiestnené vždy do prvého riadku fenotypu jedinca. Navyše, tieto uzly sú vždy susedné a len v jednom riadku fenotypu. Alter-

natíva prekladania aktívnych a neaktívnych uzlov je odsimulovateľná pridaním ďalšieho riadku redundantných (neaktívnych) uzlov.

Pre analýzu vplyvu pozície aktívnych uzlov *vrámci riadku* fenotypu jedinca sa porovnávali tri možnosti umiestnenia aktívnych uzlov: **na začiatok**, **do stredu** a **na koniec** riadku fenotypu (Obr. 5.28).

(a) (b) (c)

Obr. 5.28: Znázornenie umiestnenia aktívnych uzlov (tmavý štvorec) vo fenotype jedinca po inicializácií algoritmu – na začiatku (a), uprostred (b) a na konci (c) riadku jedinca. Prázdne štvorce reprezentujú skupiny neaktívnych (redundantných) uzlov.

Meranie bolo vykonané spolu na troch rozmeroch mriežky fenotypu: $1 \times 3n$, $2 \times 3n$ a $3 \times 3n$, kde n označuje počet hradiel referenčného obvodu. Nastavenie ďalších vstupných parametrov výpočtu popisuje tabuľka 5.11. Pre každú konfiguráciu bol výpočet spustený v celkovo 20 reštartoch.

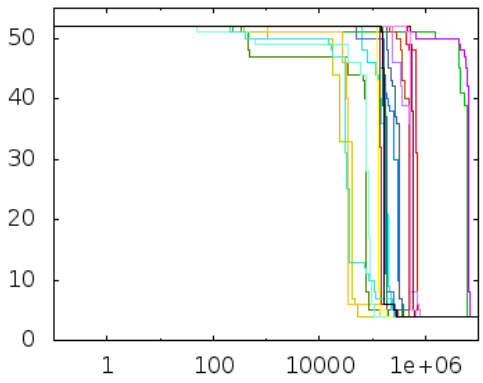
Tabuľka 5.11: Inicializácia vstupných parametrov CGP pre testovanie vplyvu umiestnenia aktívnych uzlov vo fenotype jedinca na efektivitu výpočtu algoritmu.

Veľkosť populácie	λ	1
Počet riadkov	m_r	**1/2/3**
Počet stĺpcov	m_c	156
Previazanosť	l	156
Mutácia	m	10
Pozícia aktívnych uzlov (od stĺpca)	pos	**0/52/104**
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu

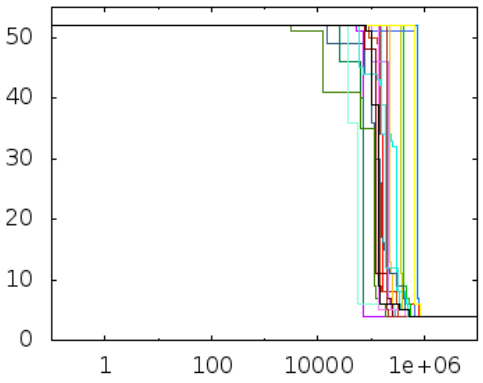
5.7.1 Fenotyp $1 \times 3n$

Uzly jedinca sú organizované v mriežke fenotypu do jedného riadku, ktorého veľkosť je 300% veľkosti referenčného obvodu. Po inicializácií tak zostávajú neaktívne 2/3 uzlov jedinca.

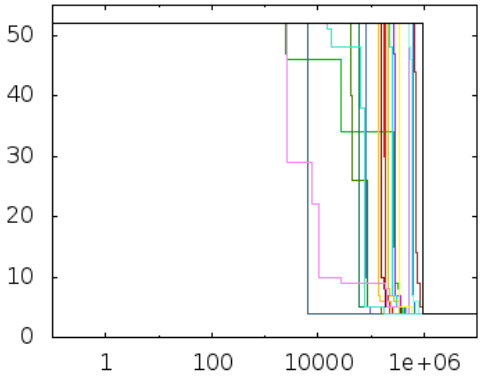
Priebeh algoritmu pre umiestnenie bloku aktívnych uzlov na začiatok, do stredu[20] a na koniec riadku mriežky zobrazuje Obr. 5.29.



(a) Umiestnenie bloku aktívnych uzlov na začiatok fenotypu



(b) Umiestnenie bloku aktívnych uzlov do stredu fenotypu



(c) Umiestnenie bloku aktívnych uzlov na koniec fenotypu

Obr. 5.29: Priebeh 20 reštartov CGP pri rôznom umiestnení aktívnych uzlov – na začiatok (a), do stredu (b) a na koniec mriežky (c) fenotypu veľkosti $1 \times 3n$ (logaritmická mierka na osi x).

[20]Rovnaký experiment ako testovanie vplyvu miery mutácie 10 pri využití redundancie

Tabuľka 5.12: Priemerný počet vygenerovaných jedincov pri veľkosti fenotypu $1 \times 3n$.

Umiestnenie aktívnych uzlov	Priemerný počet jedincov
na začiatok	1 007 928,1
do stredu	439 017,3
na koniec	443 795,5

5.7.1.1 Pozorovanie

Z grafov priebehu meraní pre jednotlivé inicializácie aktívnych uzlov nie je medzi jednotlivými experimentami viditeľný výrazný rozdiel (Obr. 5.29). Pri inicializácií aktívnych uzlov na začiatku mriežky fenotypu bolo však v dvoch reštartoch z 20 nájdené optimálne riešenie až po viac ako $6 \cdot 10^6$ vygenerovaných jedincoch. To spôsobilo nárast priemerného počtu vygenerovaných jedincov tohto prístupu na **1 007 928,1**.

Pre nájdenie optimálneho riešenia bolo v prípade umiestnenia aktívnych uzlov v strede fenotypu potrebných priemerne **439 017,3** vygenerovaných jedincov. Pri umiestnení na koniec fenotypu to bolo v priemere **443 795,5** vygenerovaných jedincov (tabuľka 5.12).

5.7.2 Fenotyp $2 \times 3n$

Druhý experiment bol vykonaný s rovnakou mierou mutácie (10). Obvod bol však doplnený o jeden redundantný riadok oproti predošlému experimentu, čím dosahuje 500% redundanciu (5/6 redundantných uzlov – viď Obr. 5.28).

Priebeh algoritmu je znázornený na Obr. 5.30.

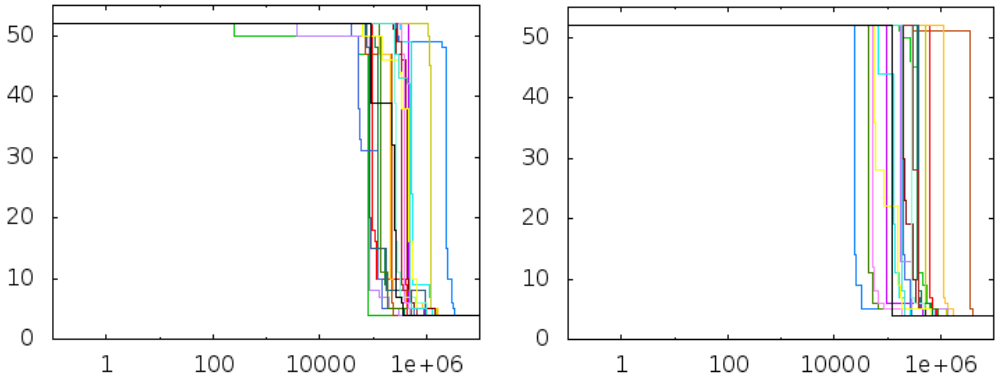
Tabuľka 5.13: Priemerný počet vygenerovaných jedincov pri veľkosti fenotypu $2 \times 3n$.

Umiestnenie aktívnych uzlov	Priemerný počet jedincov
na začiatok	906 964,2
do stredu	893 590,8
na koniec	751 024,5

5.7.2.1 Pozorovanie

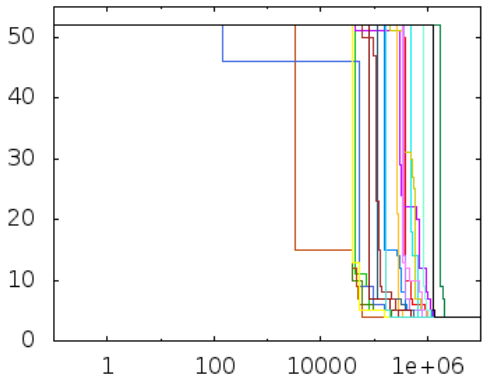
Výsledky získané týmto experimentom už vykazujú pozorovateľnejšie rozdiely v efektivite výpočtu pre skúmané inicializácie algoritmu. Experimenty naznačujú zvyšovanie efektivity výpočtu pri umiestňovaní aktívnych uzlov pri

inicializácií algoritmu bližšie ku koncu obvodu. Rozdiely v priemernom počte vygenerovaných jedincov počas výpočtu je možné vidieť v tabuľke 5.13.



(a) Umiestnenie bloku aktívnych uzlov na začiatok fenotypu

(b) Umiestnenie bloku aktívnych uzlov do stredu fenotypu



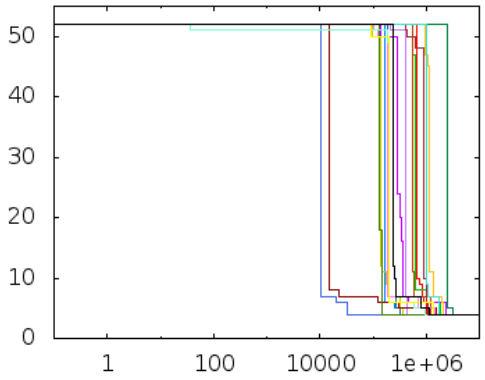
(c) Umiestnenie bloku aktívnych uzlov na koniec fenotypu

Obr. 5.30: Priebeh 20 reštartov CGP pri rôznom umiestnení aktívnych uzlov – na začiatok (a), do stredu (b) a na koniec mriežky (c) veľkosti $2 \times 3n$ (logaritmická mierka na osi x).

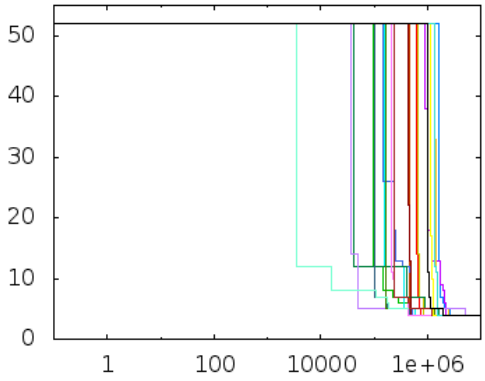
5.7.3 Fenotyp $3 \times 3n$

Pridaním ďalšieho riadku redundantných uzlov do obvodu sa zvyšuje redundancia na 800% (redundantné uzly tvoria 8/9 všetkých uzlov obvodu).

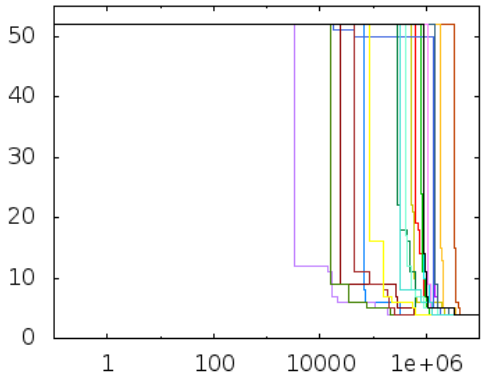
Priebeh algoritmu pre toto nastavenie je znázornený na Obr. 5.31.



(a) Umiestnenie bloku aktívnych uzlov na začiatok fenotypu



(b) Umiestnenie bloku aktívnych uzlov do stredu fenotypu



(c) Umiestnenie bloku aktívnych uzlov na koniec fenotypu

Obr. 5.31: Priebeh 20 reštartov CGP pri rôznom umiestnení aktívnych uzlov – na začiatok (a), do stredu (b) a na koniec mriežky (c) veľkosti $3 \times 3n$ (logaritmická mierka na osi x).

5.7.3.1 Pozorovanie

Zhrnutie priemerného počtu vygenerovaných jedincov je uvedené v tabuľke 5.14.

Na rozdiel od predošlého experimentu sa tentokrát ako najefektívnejšie riešenie ukázalo umiestnenie bloku aktívnych uzlov pri inicializácii algoritmu *na začiatok* fenotypu jedinca.

Tabuľka 5.14: Priemerný počet vygenerovaných jedincov pri veľkosti fenotypu $3 \times 3n$.

Umiestnenie aktívnych uzlov	**Priemerný počet jedincov**
na začiatok	1 202 924,70
do stredu	1 386 721,15
na koniec	1 648 094,60

5.7.4 Zhodnotenie

Pri umiestnení aktívnych uzlov na koniec fenotypu vzniká teoreticky viac možností pre ich mutáciu, nakoľko sa pred nimi nachádza väčší počet stĺpcov, ku ktorých génom by mohli byť pripojené ich vstupy. Menej stĺpcov „za“ blokom aktívnych uzlov (v krajnom prípade žiadne[21]) však neumožňuje pripojenie fragmentu obvodu vygenerovaného evolúciou na koniec aktívnej časti obvodu.

Narozdiel od tohto prístupu, umiestnením aktívnych uzlov do stredu fenotypu je možné k aktívnej časti obvodu pripájať vygenerované fragmenty z ľubovoľnej strany. Táto inicializácia je teda všeobecnejšia, avšak pre zapojenie uzlov v stĺpcoch „za“ aktívnou časťou je nevyhnutná vhodná mutácia výstupu obvodu (pri mutácii ľubovoľného aktívneho uzla tieto uzly nie sú dostupné). Oproti predošlému prístupu tým vzniká viac uzlov, ktoré majú relatívne menšiu pravdepodobnosť aktívne sa podieľať na výslednej logickej funkcii obvodu.

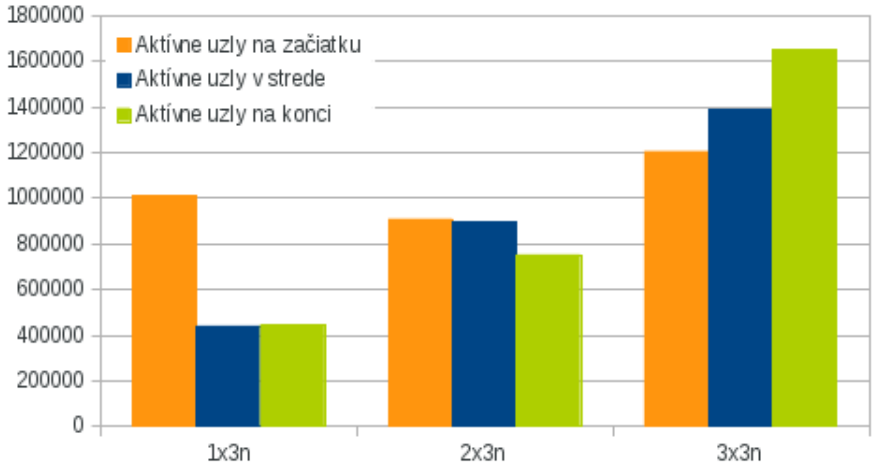
Ďalším extrémnym prípadom je umiestnenie bloku aktívnych uzlov na začiatok fenotypu. Možnosti evolúcie môžu byť v takom prípade do istej miery obmedzené, nakoľko pri mutácii je napr. na vstup prvého aktívneho uzla, nachádzajúceho sa v prvom stĺpci fenotypu, možné pripojiť jedine vstup obvodu.

Súhrnné výsledky merania sú zobrazené na Obr. 5.32 a 5.33, kde je znázorné rozloženie počtu vygenerovaných obvodov pri jednotlivých meraniach.

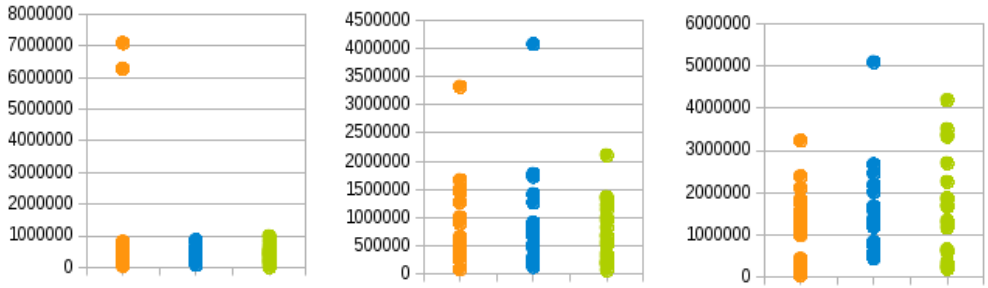
Získané výsledky však nepotvrdili žiadny výrazný dopad niektorého z vyššie uvedených teoretických predpokladov. Zatiaľ čo pri najmenšom obvode s jedným riadkom fenotypu bola efektívnosť výpočtu pri umiestnení aktívnych uzlov do stredu a na koniec fenotypu veľmi podobná, pri umiestnení na začiatok fenotypu bola výrazne horšia (spôsobené dvomi výrazne neefektívnymi reštartami – viď 5.33 (a)).

Pri rozšírení obvodu o ďalší riadok redundantných uzlov, už bol medzi priemernými počtami vygenerovaných jedincov pri inicializácii aktívnych uzlov v strede a na konci fenotypu výraznejší rozdiel oproti predošlému

[21] Umiestnenie aktívnych uzlov na koniec mriežky fenotypu



Obr. 5.32: Závislosť priemerného počtu vygenerovaných jedincov v priebehu výpočtu algoritmu na umiestnení aktívnych uzlov vo fenotype jedinca.



(a) $1 \times 3n$ (b) $2 \times 3n$ (c) $3 \times 3n$

Obr. 5.33: Rozloženie počtu vygenerovaných jedincov (os y) v jednotlivých reštartoch pri rôznych pozíciách aktívnych uzlov vo fenotype. **Oranžovou** farbou sú zobrazené výsledky pri umiestnení aktívnych obvodov na začiatok, **modrou** do stredu a **zelenou** na koniec fenotypu.

meraniu. Získaný výsledok naznačuje zlepšovanie efektívnosti výpočtu inicializáciou aktívnych uzlov bližšie ku koncu fenotypu.

Tomuto pozorovaniu ale odporuje tretie meranie vykonané na obvode s tromi riadkami uzlov, kde zase prevláda presne opačný trend vývoja efektívnosti v závislosti na pozícií aktívnych uzlov vo fenotype jedinca – ako najefektívnejšie sa javí umiestnenie aktívnych uzlov na začiatku fenotypu.

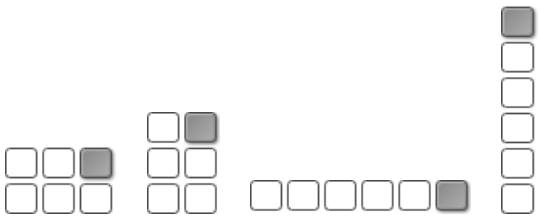
Zo získaných výsledkov teda **nie je možné dokázať jednoznačný vplyv pozície inicializácie aktívnych uzlov vo fenotype jedinca na efektívnosť výpočtu algoritmu**.

Pomerne výrazné je všeobecné zvýšenie počtu vygenerovaných jedincov pri zväčšení mriežky fenotypu jedinca. Vzťah medzi mierou mutácie a veľkosťou fenotypu je skúmaný v časti 5.5.

5.8 Vplyv vnútornej štruktúry fenotypu na výpočet algoritmu

Pri pevne stanovenom počte uzlov jedinca je možné zostrojiť niekoľko fenotypov, ktoré sa môžu navzájom líšiť organizáciou uzlov do riadkov a stĺpcov (topológiou). V tejto časti sa skúma vplyv topológie mriežky fenotypu na výpočtovú silu resp. efektívnosť algoritmu.

Pre experimenty bol stanovený počet uzlov jedinca na 6-násobok počtu uzlov referenčného obvodu. Prehľad testovaných topológií je zobrazený na Obr. 5.34.



Obr. 5.34: Topológie testovacích obvodov pre analýzu vplyvu topológie obvodu na výpočet algoritmu. Štvorce označujú vždy rovnako veľký blok uzlov. Plným štvorcom je označený blok aktívnych uzlov (inicializované hradlami referenčného obvodu).

Aktívne uzly v testovaných topológiách vystupujú ako pevný blok uzlov umiestnených za sebou v jednom riadku. Delenie aktívnych uzlov do viacerých riadkov sa neuvažuje, nakoľko nie je univerzálne aplikovateľné[22].

[22]Rozdelenie aktívnych uzlov do viacerých riadkov tak, že v jednom stĺpci je nanajvýš jeden aktívny uzol, nemá z hľadiska výpočtu žiadny význam (viď 5.7). Umiestnenie viac ako jedného aktívneho uzla do riadkov jedného stĺpca zase nemusí byť vždy možné, keďže uzly v rovnakom stĺpci nemôžu byť navzájom prepojené.

Počet stĺpcov testovacích obvodov sa v jednotlivých experimentoch stanovuje na násobky počtu hradiel referenčného jedinca výhradne kvôli prehľadnosti. Aktívne uzly sú umiestnené na koniec riadku fenotypu a mutácia stanovená na 10 génov genotypu. Optimalita nastavenia týchto parametrov je diskutovaná v podkapitolách 5.4 a 5.7.

Súhrn inicializácie vstupných parametrov je popísaný v tabuľke 5.15[23].

Tabuľka 5.15: Inicializácia vstupných parametrov CGP pre testovanie vplyvu topológie fenotypu jedinca na efektivitu výpočtu algoritmu.

Veľkosť populácie	λ	1
Počet riadkov	m_r	**1/2/3/6**
Počet stĺpcov	m_c	$6n/3n/2n/1n$
Previazanosť	l	l_{max}
Mutácia	m	10
Pozícia aktívnych uzlov	pos	na konci riadku
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu

5.8.1 Experiment

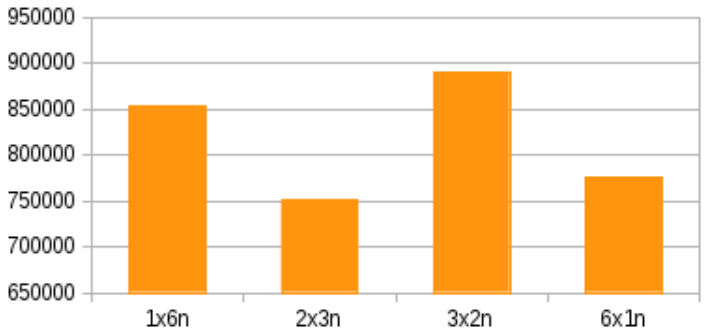
Pre každú z testovaných topológií bolo vykonaných, rovnako ako v predošlých meraniach, celkom 20 reštartov. Ich priebeh zobrazuje Obr. 5.37. Na Obr. 5.36 je znázornené rozloženie počtu vygenerovaných jedincov v jednotlivých reštartoch výpočtu.

Priemerný počet vygenerovaných jedincov pre jednotlivé topológie je zhrnutý v tabuľke 5.16 a na obrázku 5.35.

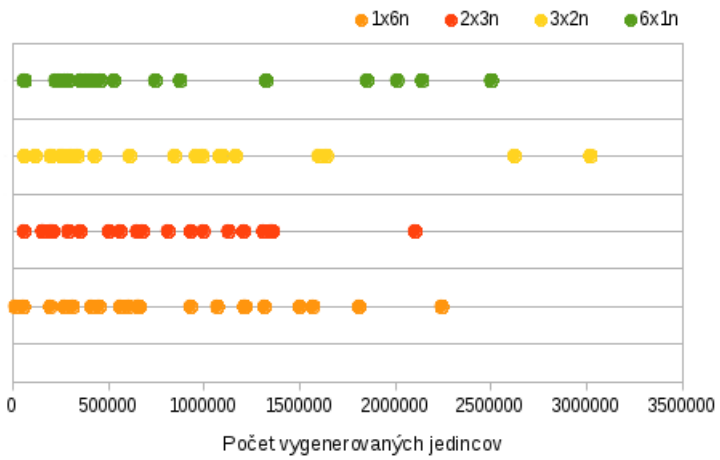
Tabuľka 5.16: Priemerný počet vygenerovaných jedincov pri rôznych topológiách fenotypu jedinca.

Topológia fenotypu ($m_r \times m_c$)	**Priemerný počet jedincov**
$1 \times 6n$	853 654,15
$2 \times 3n$	751 024,50
$3 \times 2n$	889 829,05
$6 \times 1n$	775 202,50

[23]Parameter n označuje počet hradiel referenčného jedinca.



Obr. 5.35: Priemerný počet vygenerovaných jedincov pri použití rôznych topológií fenotypu.

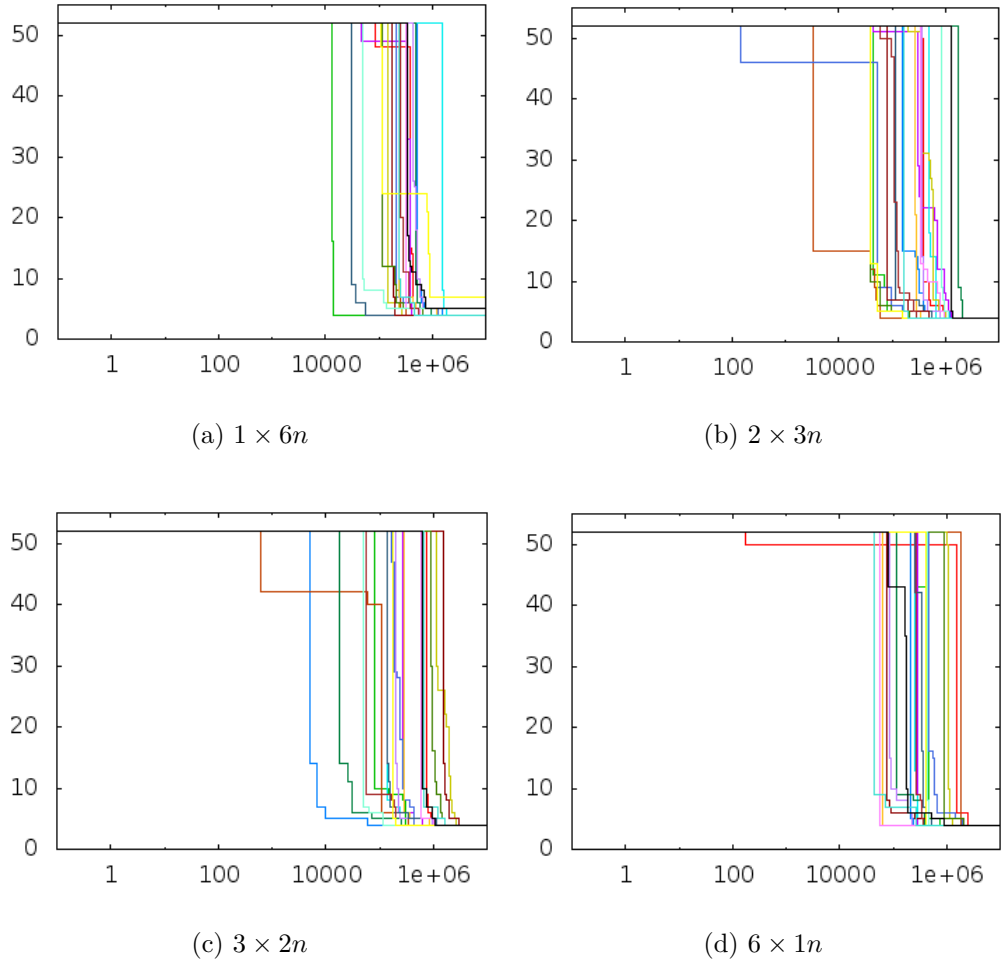


Obr. 5.36: Rozloženie počtu vygenerovaných jedincov pre rôzne topológie fenotypu jedinca.

5.8.2 Zhodnotenie

Na obrázku 5.35 je možné pozorovať rozdiely v priemernom počte vygenerovaných jedincov počas výpočtu algoritmu s rôznymi topológiami fenotypu. Nezdá sa však, žeby tento vývoj priamo súvisel s počtom riadkov resp. stĺpcov fenotypu.

Ako najefektívnejšia sa ukázala topológia s dvomi riadkami a trojnásobným počtom stĺpcov ako je počet hradiel referenčného obvodu. Naopak, najmenej efektívny výpočet bol pri použití mriežky s rozmermi $3 \times 2n$. Táto topológia potrebovala pre nájdenie optimálneho riešenia v priemere o **138 804,55** vygenerovaných jedincov viac, čo je približne 18.5% počtu vygenerovaných jedincov najefektívnejšieho výpočtu.



(a) $1 \times 6n$ (b) $2 \times 3n$

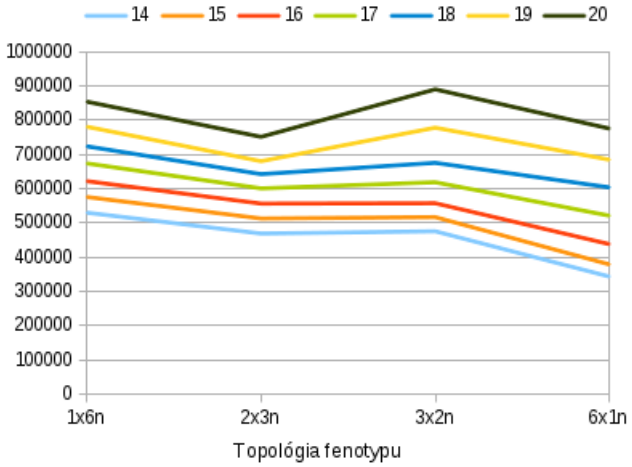
(c) $3 \times 2n$ (d) $6 \times 1n$

Obr. 5.37: Priebeh algoritmu pri použití rôznych topológií fenotypu (logaritmická mierka na osi x).

V rozložení počtu vygenerovaných jedincov všetkých reštartov (Obr. 5.36) je možné pozorovať „zhluky“ dát, v ktorých je sústredená väčšina výsledkov pre danú topológiu. Pre topológiu $1 \times 6n$ je to približne do $2 \cdot 10^6$, pre $2 \times 3n$ približne do $1,5 \cdot 10^6$, pre $3 \times 2n$ približne do $1,3 \cdot 10^6$ a pre $6 \times 1n$ približne do 10^6 vygenerovaných jedincov. Tieto zhluky by mohli naznačovať priaznivý vplyv zavádzania redundantných riadkov do fenotypu.

V meraniach sa však vyskytujú reštarty, ktoré do takto zvolených „zhlukov“ nepatria, pričom ich počet rastie spolu s pridávaním riadkov, resp. **odoberaním stĺpcov** fenotypu. Pre topológiu $6 \times 1n$ je to približne päť reštartov, čo je až štvrtina všetkých reštartov vykonaných na danej topológii.

Na Obr. 5.38 začína byť viditeľné relatívne zlepšovanie efektívnosti výpočtu s pridávaním riadkov už pri „orezaní“ štyroch najhorších reštartov každého merania.



Obr. 5.38: Priemerný počet vygenerovaných jedincov pri uvažovaní n najlepších reštartov s rôznymi topológiami fenotypu.

Na základe tejto analýzy je možné *predpokladať* priaznivý vplyv využívania redundantných riadkov vo fenotype jedinca, avšak v kombinácií s využitím redundancie v stĺpcoch – teda priaznivý vplyv redundancie ako takej, čo už bolo overené aj predošlými meraniami (napr. 5.4.8).

Prezentované výsledky však **nevykazujú priamu závisloť efektívnosti výpočtu na topológii, resp. usporiadaní uzlov fenotypu jedinca**.

5.9 Analýza vplyvu spôsobu inicializácie redundantných uzlov na efektívnosť výpočtu

Vo všetkých predošlých meraniach boli redundantné uzly vo fenotype prvého jedinca inicializované fragmentami referenčného obvodu (viď časť 4.2.1 a Obr. 4.4). Účelom tohto merania je porovnať efektívnosť tohto prístupu s prístupom využívajúcim **náhodnú inicializáciu** redundantných uzlov (Obr. 4.3).

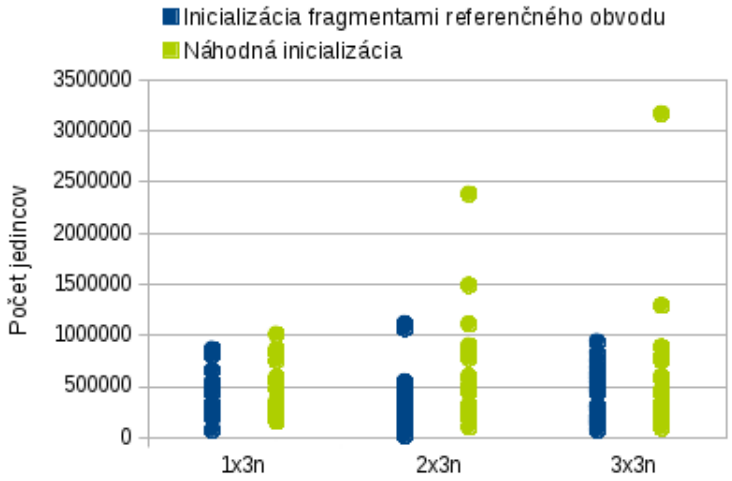
Meranie bolo vykonané 20 reštartami pre každú z troch veľkostí fenotypu jedinca (podobne ako v predošlom meraní): $1 \times 3n$, $2 \times 3n$ a $3 \times 3n$. Inicializáciu všetkých vstupných parametrov výpočtu zobrazuje tabuľka 5.17.

Tabuľka 5.17: Inicializácia vstupných parametrov CGP pre testovanie vplyvu inicializácie redundantných uzlov na efektivitu výpočtu algoritmu.

Veľkosť populácie	λ	1
Počet riadkov	m_r	1/2/3
Počet stĺpcov	m_c	$3n$
Previazanosť	l	l_{max}
Mutácia	m	10/25/50
Pozícia aktívnych uzlov	pos	uprostred fenotypu
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	**náhodne**

5.9.1 Experiment

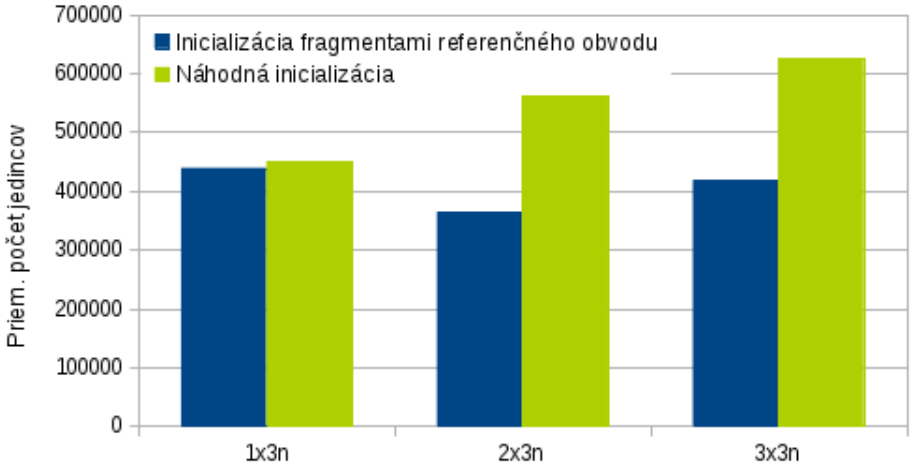
Porovnanie počtu vygenerovaných jedincov v priebehu výpočtu algoritmu pre všetky reštarty a porovnanie celkového priemerného počtu vygenerovaných jedincov je zobrazené na Obr. 5.39, resp. Obr. 5.40.



Obr. 5.39: Prehľad počtu vygenerovaných jedincov pre rôzne spôsoby inicializácie redundantných uzlov (20 reštartov).

5.9.2 Zhodnotenie

V porovnaní priemerného počtu vygenerovaných jedincov na obrázku 5.39 je vidieť vo všetkých troch prípadoch lepšiu efektívnosť (resp. menší priemerný počet vygenerovaných jedincov) výpočtu algoritmu pri inicializácií redundantných uzlov prvého jedinca **fragmentami** referenčného obvodu.



Obr. 5.40: Priemerný počet vygenerovaných jedincov pri použití rôznych spôsobov inicializácie redundantných uzlov.

Pri inicializácií redundantných uzlov *náhodne* je relatívne malá pravdepodobnosť, že by tieto uzly po zapojení do aktívnej časti obvodu realizovali „zmysluplnú“ logickú funkciu. „Náskok“ uzlov inicializovaných fragmentami referenčného obvodu sa však hneď po začatí evolúcie stráca: mutácia neaktívnych uzlov fenotypu jedinca nie je nijak kontrolovaná alebo obmedzovaná, nakoľko pri generovaní jedincov nemajú mutácie neaktívnej časti obvodu na výslednu funkčnosť, resp. nefunkčnosť jedinca žiadny vplyv. Dá sa preto očakávať, že v priebehu niekoľkých generácií bude štruktúra neaktívnej časti fenotypu jedinca (prepojenie a logické funkcie pridelené jednotlivým uzlom) rovnako náhodná, ako by to bolo v prípade náhodnej inicializácie redundantných uzlov na začiatku výpočtu.

Toto tvrdenie potvrdzuje Obr. 5.39, na ktorom je možné pozorovať veľmi podobné počty vygenerovaných jedincov pri väčšine reštartov skúmaných prístupov. Priemerný počet vygenerovaných prístupov pri náhodnej inicializácií redundantných uzlov zhoršuje len niekoľko málo (2) reštartov, ktoré potrebovali pre nájdenie optimálneho riešenia výrazne viac vygenerovaných jedincov ako väčšina reštartov rovnakého merania. Tento jav je však spôsobený vplyvom náhody a nie je nijak špecifický pre použité nastavenie vstupných parametrov výpočtu.

Na základe výsledkov tohto merania je možné skonštatovať, že inicializácia redundantných uzlov fragmentami referenčného obvodu *môže mať* na efektívnosť výpočtu mierne priaznivý vplyv (najmä zo začiatku výpočtu, kedy ešte fragmenty referenčného obvodu neboli náhodne zmutované), **vo všeobecnosti však nemá na výpočet žiadny zásadný vplyv**.

5.10 Porovnanie výpočtovej sily evolúcie a náhodného generovania jedincov

Ako bolo diskutované v predošlých experimetoch, efektívnosť CGP závisí vo veľkej miere na náhode. Dôvodom je netrivialita kríženia a stanovenie mutácie ako jediného (náhodného) evolučného faktora (podrobnejšie v časti 3.2.2).

Účelom tohto merania je preto porovnanie efektívnosti resp. výpočtovej sily takejto relatívne obmedzenej evolúcie, ktorá bola využitá vo všetkých predošlých experimentoch, a „náhodného strieľania“, pri ktorom sú kandidátne obvody konštruované úplne náhodne.

5.10.1 Algoritmus

Z algoritmu CGP, použitého v predošlých experimentoch, bola pre účely tohto merania **úplne odstránená mutácia**.

Predok prvej generácie je opäť inicializovaný referenčným obvodom, všetci ďalší jedinci sú však v priebehu výpočtu generovaní úplne náhodne nezávisle na predkovi.

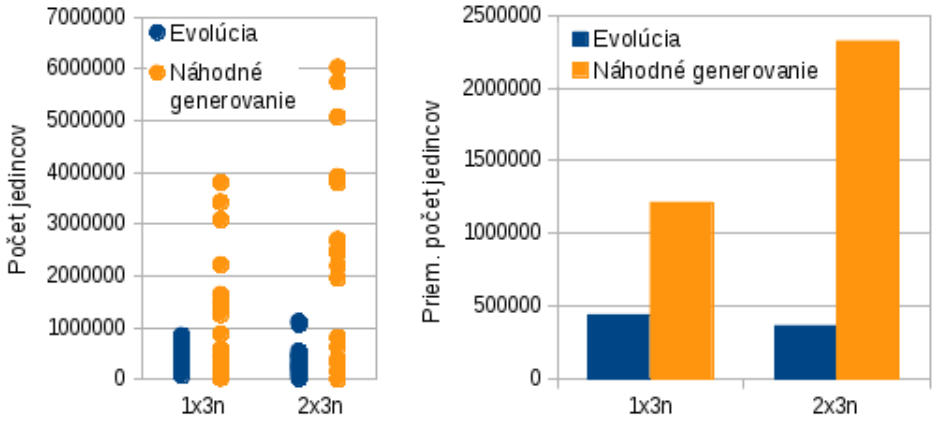
5.10.2 Experiment

Testovanie prebiehalo na dvoch veľkostiach fenotypu jedinca, a to $1 \times 3n$ a $2 \times 3n$, pričom výsledky boli porovnané s výsledkami predošlých experimentov, kde boli použité fenotypy rovnakých rozmerov. Pre každú z veľkostí fenotypu bolo vykonaných spolu 20 reštartov výpočtu.

Konfiguráciu vstupných parametrov zobrazuje tabuľka 5.18.

Tabuľka 5.18: Inicializácia vstupných parametrov CGP pre testovanie efektívnosti náhodného generovania jedincov algoritmu.

Veľkosť populácie	λ	1
Počet riadkov	m_r	1/2
Počet stĺpcov	m_c	$3n$
Previazanosť	l	l_{max}
Mutácia	m	10/25/bez mutácie
Pozícia aktívnych uzlov	pos	uprostred fenotypu
Max. vygenerovaných jedincov	-	10^7
Inicializácia redundantných génov	-	fragmenty ref. obvodu



(a) Počet vygenerovaných jedincov v jednotlivých reštartoch

(b) Priemerný počet vygenerovaných jedincov

Obr. 5.41: Porovnanie evolučného algoritmu a náhodného generovania jedincov.

Porovnanie počtu vygenerovaných jedincov v jednotlivých reštartoch a celkového priemerného počtu vygenerovaných jedincov zobrazuje Obr. 5.41. Porovnanie minimálneho počtu vygenerovaných jedincov dosiahnutého v jednotlivých meraniach je uvedené v tabuľke 5.19.

Tabuľka 5.19: Porovnanie minimálneho počtu vygenerovaných jedincov pri evolúcii a náhodnom generovaní jedincov.

Topológia	**Evolúcia**	**Náhodné generovanie**	**Rozdiel**
$1 \times 3n$	73 627	30 090	-59,13%
$2 \times 3n$	15 512	1 992	-87,16%

5.10.3 Zhodnotenie

Náhodným generovaním jedincov sa rovnako podarilo objaviť optimálne riešenie daného problému, a to vo všetkých vykonaných reštartoch. V tabuľke 5.19 je dokonca vidieť **rekordne nízky** počet vygenerovaných jedincov pre nájdenie optimálneho riešenia dosiahnutý vypustením evolúcie pri obidvoch rozmeroch fenotypu.

Táto relatívne vysoká efektívnosť mohla byť dosiahnutá vďaka možnosti generovania ľubovoľne komplexných štruktúr bez obmedzení a v každom

okamihu výpočtu. Vygenerovanie optimálneho jedinca je teda pri použití úplne náhodného prístupu vždy *rovnako pravdepodobné* nezávisle na celkovom počte vygenerovaných jedincov, resp. generácií. To dokazuje aj rovnomernejšie rozloženie počtu vygenerovaných jedincov medzi najlepším a najhorším reštartom oproti predošlým experimentom využívajúcim evolúciu (Obr. 5.41 (a)).

Napriek lepším najefektívnejším reštartom náhodného generovania jedincov bola celková priemerná efektívnosť (počet vygenerovaných jedincov) tohto prístupu **výrazne horšia** oproti evolučnému prístupu (Obr. 5.41 (b)). Ďalšie zhoršovanie priemernej efektívnosti sa dá očakávať pri optimalizácii komplexnejšieho problému (optimálny obvod testovacej logickej funkcie – *xor5* je tvorený len 4 hradlami).

Z týchto dôvodov sa **náhodné generovanie kandidátnych jedincov nedá považovať za vhodnú metódu optimalizácie logických obvodov.**

Záver

Kartézske genetické programovanie ako implementácia *assemble-and-test* metódy návrhu logických obvodov nie je evolučným algoritmom v pravom slova zmysle: hľadanie vzorov v genotypoch jeho jedincov – logických obvodov alebo určovanie relatívnej „vzdialenosti“ medzi nimi je netriviálny problém. Navyše, jedinci sú až príliš hákliví na zmenu svojho genotypu, pričom účinok takejto zmeny nemusí byť vždy predvídateľný. Z týchto dôvodov sa v CGP neuvažuje kríženie (jeden z dvoch evolučných operácií genetického programovania) a evolúcia je zabezpečovaná výhradne mutáciou. Algoritmus sa tým stáva vo zásadnej miere závislý na náhode.

Z uvedených dôvodov sa komplikuje aj regulácia selekčného tlaku, ktorý je ďalším významným parametrom evolučných algoritmov. Možnosti jeho ovplyvnenia, ako aj vplyv konfigurácie vstupných premenných na priebeh algoritmu bol predmetom skúmania tejto práce.

Významným atribútom CGP je *redundancia*, v ktorej spočíva zásadná časť potenciálu algoritmu. Redundancia dáva mutácii priestor pre náhodné generovanie neaktívnych fragmentov logického obvodu bez vplyvu na aktívne uzly obvodu, vďaka čomu je algoritmus potenciálne schopný komplexnejšej prestavby referenčného obvodu.Priaznivý vplyv redundancie na priebeh algoritmu bol aj experimentálne potvrdený.

Pre optimálnu efektívnosť výpočtu CGP je však nevyhnutné nájsť rovnováhu medzi redundanciou (veľkosťou fenotypu) a stanovením miery mutácie: meraním sa ukázala silná závislosť efektívnosti algoritmu práve na zvolenej mutácií pre danú veľkosť fenotypu. Vhodným nastavením bolo dokonca možné rádovo zachovať priemerný počet vygenerovaných jedincov počas výpočtu aj pri zväčšení fenotypu jedinca. Ako alternatíva **selekčného tlaku** by sa teda dala považovať práve stanovená miera mutácie. Ani tá však nedokáže kvôli vysokej závislosti na náhode s istotou garan-

tovať relatívne efektívny priebeh algoritmu alebo vôbec nájdenie optimálneho riešenia. Zásadný vplyv náhody je však možné v niektorých prípadoch čiastočne potlačiť viacerými reštartami experimentu a štatistickým zhodnotením získaných výsledkov.

Algoritmus CGP sa pri optimalizácii „vhodných“ obvodov ukázal *regulovateľný* pomocou vstupných parametrov a v konečnom dôsledku *schopný* optimalizácie, pričom v efektivite výrazne predbieha iný automatizovaný prístup, ktorý na náhode závisí ešte viac – algoritmus hľadajúci optimálne riešenie generovaním kandidátnych obvodov úplne náhodne. Svojím potenciálom prevyšuje aj konvenčný návrh logických obvodov, ktorý je, na rozdiel od CGP, zase zaťažený ľudským faktorom.

Zásadným úskalím tohto algoritmu je však okrem náhody aj experimentálne potvrdená závislosť na štruktúre referenčného obvodu. Pri optimalizácii obvodu vyžadujúceho kompletnú reorganizáciu vnútornej štruktúry síce redundancia umožňuje vygenerovanie a zapojenie ľubovoľne zložitého fragmentu obvodu, no pravdepodobnosť úspechu sa v takom prípade blíži pravdepodobnosti úspechu náhodného generovania. Dôležitým faktorom je aj veľkosť hľadaného optimálneho riešenia.

Ďalší výskum

Predmetom ďalšieho výskumu tejto problematiky by mohlo byť ešte hlbšie preskúmanie vzťahu optimálnej miery mutácie a veľkosti fenotypu jedincov, resp. redundancie, a taktiež preskúmanie možností zvýšenia univerzálnosti použitia algoritmu a nezávislosti jeho výpočtovej sily na použitom referenčnom obvode.

Literatúra

[1] A. Harrison, M.: *Introduction to Switching and Automata Theory.* McGraw-Hill, 1965, 499 s.

[2] Berkeley Logic Synthesis and Verification Group: A System for Sequential Synthesis and Verification. September 2012. Dostupné z: `http://www.eecs.berkeley.edu/~alanmi/abc/`

[3] D. Hachtel, G.; Somenzi, F.: *Logic Synthesis and Verification Algorithms.* Boston, MA: Kluwer Academic Publishers, 1996, 564 s.

[4] F. Miller, J.: *Cartesian Genetic Programming.* Natural Computing Series, Springer, 2011, ISBN 978-3-642-17309-7.

[5] F. Miller, J.; Job, D.; K. Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*, , č. 1, 2000: s. 7–35.

[6] F. Miller, J.; Thomson, P.: Cartesian Genetic Programming. V *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000). LNCS*, ročník 1802, Springer-Verlag, 2000, s. 121–132.

[7] Fišer, P.; Schmidt, J.; Vašíček, Z.; ai.: On logic synthesis of conventionally hard to synthesize circuits using genetic programming. V *Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, IEEE 13th International Symposium, Apríl 2010, s. 346–351.

[8] Fišer, P.: BOOM – The Boolean Minimizer (C++ library). Dostupné z: `http://ddd.fit.cvut.cz/prj/BOOM/`

[9] Hassoun, S.; Sasao, T.: *Logic Synthesis and Verification.* Boston, MA: Kluwer Academic Publishers, 2002, 454 s.

[10] Laboratory of Mathematical Logic at PDMI: DIMACS CNF format. Apríl 2013. Dostupné z: `http://logic.pdmi.ras.ru/~basolver/dimacs.html`

[11] Poli, R.; Langdon, W. B.; McPhee, N. F.: *A field guide to genetic programming*. Publikované cez `http://lulu.com`, voľne dostupné na adrese `http://www.gp-field-guide.org.uk`, 2008.

[12] S. Tseitin, G.: On the complexity of derivation in propositional calculus. V *A. Slisenko (Ed.), Studies in Constructive Mathematics and Mathematical Logic*, Steklov Mathematical Institute, 1968, s. 115–125.

[13] SAT Research Group, Princeton University: SAT Research at Princeton. Apríl 2013. Dostupné z: `http://www.princeton.edu/~chaff/index.html`

[14] University of California, Berkeley: *Berkeley Logic Interchange Format (BLIF)*. Júl 1992.

[15] Vašíček, Z.; Sekanina, L.: On Area Minimization of Complex Combinational Circuits Using Cartesian Genetic Programming. V *WCCI 2012 IEEE World Congress on Computational Intelligence*, Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Jún 2012.

Dodatok **A**

Zoznam použitých skratiek

CGP Cartesian Genetic Programming – Kartézske genetické programovanie

CNF Conjunctive normal form – Konjunktívna normálna forma

SAT Boolean satisfiability problem – Problém splniteľnosti pravdivostnej formuly

Dodatok **B**

Obsah priloženého CD

- `abc`
  - `2-gates.genlib` ..... definícia použitých hradiel
  - `abc.exe` ..... nástroj pre kontrolu ekvivalencie logických obvodov
  - `readme.txt` ..... návod na použitie nástroja abc
- `circuits`
  - `reference` ..... referenčné obvody použité v tejto práci
  - `evolved` .. obvody získané ako výstupy implementovaného algoritmu
    - `mux`
    - `xor5`
- `readme.txt` ..... informácie o CD a jeho obsahu
- `doc`
  - `usage.txt` ..... prehľad vstupných parametrov aplikácie
- `run`
  - `GNU-Linux-x86` ..... spustiteľné súbory aplikácie pre OS Linux
- `src` ..... zdrojové kódy aplikácie
- `text`
  - `DP_Pelak_Stanislav_2013.pdf` ..... text práce vo formáte PDF
  - `DP_Pelak_Stanislav_2013.tex` ..... text práce vo formáte TEX