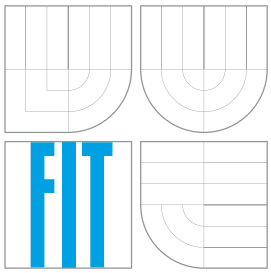
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁSTROJ PRO VIZUÁLNÍ ANALÝZU EVOLUCE OBVODŮ

A TOOL FOR VISUAL ANALYSIS OF CIRCUIT EVOLUTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JANA STAUROVSKÁ
AUTHOR

VEDOUCÍ PRÁCE — prof. Ing. LUKÁŠ SEKANINA, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Cílem diplomové práce je zpracovat studii o kartézském genetickém programování se zaměřením na použití v oblasti evoluce obvodů a vytvořit návrh konceptu vizualizace této evoluce. Následně je cílem vytvořit program umožňující vizualizovat evoluci obvodů kartézského genetického programování, její jednotlivé generace, stejně tak i jednotlivé chromozomy, dále umožňující zobrazovat změny mezi generacemi a chromozomy a porovnávat více chromozomů najednou. Pro výsledný program bylo rovněž zpracováno několik příkladů použití.

Abstract

The main goal of the master's thesis is to compose a study on cartesian genetic programming with focus on evolution of circuits and to design a concept for visualisation of this evolution. Another goal is to create a program to visualise the circuit evolution in cartesian genetic programming, its generations and chromosomes. The program is capable of visualising the changes between generations and chromosomes and comparing more chromosomes at once. Several user cases had been prepared for the resulting program.

Klíčová slova

genetika, genetické programování, kartézské genetické programování, hradlo, chromozom, evoluce, evoluční algoritmus

Keywords

genetics, genetic programming, cartesian genetic programming, gate, chromosome, evolution, evolutionary algorithm

Citace

Jana Staurovská: Nástroj pro vizuální analýzu
evoluce obvodů, diplomová práce, Brno, FIT VUT v Brně, 2012

Nástroj pro vizuální analýzu evoluce obvodů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením profesora Lukáše Sekaniny.

.....
Jana Staurovská
21. května 2012

Obsah

Kapitola 1

Úvod

Čoraz viac princípov z prírody preniká do technológií. Jedným z nich je aj Darwinova teória evolúcie a jej moderní následníci, teórie neodarwinizmu. Zjednodušene povedané, tieto teórie postulujú, že najlepšie prispôsobení jedinci majú najviac potomkov, ktorí potom vytvoria ďalšiu, potenciálne lepšiu generáciu. Tento základný princíp bol prenesený do technológií s takmer úplným zachovaním terminológie. Vznikla nová trieda prehľadávacích algoritmov, ktoré dnes poznáme pod názvom evolučné algoritmy.

Cieľom tejto diplomovej práce je uviesť čitateľa do problematiky kartézskeho genetického programovania (ďalej len CGP, skrátené z anglického názvu Cartesian Genetic Programming), ktoré sa v návrhu logických obvodov často používa, navrhnúť vhodné prostredie a užívateľské rozhranie pre vizualizáciu výsledkov CGP, tj. generácií chromozómov a chromozómov samotných, s využitím výsledkov mojej bakalárskej práce [10] a implementovať ho. Vznikne tak komplexný program, ktorý bude umožňovať vizuálnu kontrolu výsledkov evolúcie CGP, umožní ich manipuláciu a úpravy pre ďalšie použitie, napríklad skrytie nepoužitých hradiel, simuláciu chromozómu, zobrazenie histórie evolúcie chromozómu, porovnanie rozdielov dvoch chromozómov a podobne.

Tento text je štrukturovaný do ôsmych kapitol. Prvou kapitolou je úvod, ktorý práve čítate. Ďalšou je popis CGP z obecného hľadiska. Potom nasleduje kapitola zameraná na využitie CGP pri evolučnom návrhu obvodov. Vo štvrtej kapitole sa nachádza výčet a porovnanie rôznych nástrojov pre prácu a vizualizáciu CGP. V ďalšej kapitole je navrhnutý koncept nástroja pre vizualizáciu evolúcie CGP. V šiestej kapitole je popísaná implementácia programu podľa návrhu z piatej kapitoly. Predposlednou kapitolou sú príklady použitia s názornými ukážkami, ktoré slúžia na otestovanie programu. Poslednou kapitolou je záver, kde je zhrnutá práca, ktorá bola dosiaľ urobená a prípadné rozšírenia.

Kapitola 2

Kartézske genetické programovanie

2.1 Evolučné algoritmy

Evolučné algoritmy vychádzajú z Darwinovej teórie prirodzeného výberu. Tento termín zastrešuje rôzne stochastické prehľadávacie algoritmy s týmito rysmi [8]:

- Používajú populáciu (množinu) kandidátnych riešení, ktorá umožňuje paralelný prístup k prehľadávaniu.
- K vytváraniu nových kandidátnych riešení používajú biológiou inšpirované operátory, ktoré kombinujú kandidátne riešenia existujúce v danom čase.

Pre porovnanie základné rysy klasického darwinizmu sú [7]:

- Populácie sú geneticky variabilné, premenlivosť je náhodná vzhľadom k prostrediu.
- Populácie majú neobmedzenú kapacitu rastu, ale potravné a priestorové zdroje sú obmedzené, a preto dorastá iba časť zygot na jedincov schopných rozmnožovania – medzi jedincami musí existovať "boj o prežitie".
- Potomkov plodia hlavne najlepšie vybavení jedinci, a tým prenášajú svoje genetické dispozície vo zvýšenej miere do ďalších generácií – zastúpenie vlastností vhodných pre dané prostredie sa tak stále zvyšuje.
- Vďaka tomuto prírodnému výberu sa druhy prispôsobujú prostrediu a dlhodobým pôsobením výberu je možné vysvetliť celú evolúciu.

Inými slovami môžeme povedať, že biologická evolúcia je progresívna zmena genetického obsahu populácie v priebehu mnohých generácií. Obsahuje tieto tri komponenty [7]:

1. **Prirodzený výber** je proces, v ktorom jedinci vysoko adaptovaní na prostredie (angl. *with high fitness*) vstupujú s väčšou pravdepodobnosťou do procesu reprodukcie než ostatní jedinci.
2. **Náhodný genetický drift**, v ktorom náhodné udalosti v živote jedincov ovplyvňujú populáciu. Takýmito udalosťami sú napr. náhodná mutácia genetického materiálu alebo náhodná smrť jedinca s vysokou fitness hodnotou predtým, než mal možnosť účastniť sa produkčného procesu. Náhodné efekty genetického driftu sú významné hlavne pre malé populácie.

3. **Reprodukčný proces**, v rámci ktorého sa z rodičov vytvárajú potomkovia. Genetická informácia potomkov je vytvorená vzájomnou výmenou genetickej informácie rodičov. Najčastejšie tento proces prebieha tak, že z genetickej informácie dvoch jedincov sa náhodne vyberú časti chromozómu – informácie, z ktorých je potom zostavená genetická informácia nového jedinca – potomka (tzv. sexuálna reprodukcia).

Pri evolučných algoritmoch je riešenie nejakej úlohy prevedené na proces evolúcie populácie náhodne vygenerovaných riešení. Každé riešenie je zakódované do reťazca symbolov (parametrov) a ohodnotené tzv. fitness funkciou, ktorá vyjadruje kvalitu riešenia – čím je hodnota väčšia tým je dané riešenie perspektívnejšie a častejšie vstupuje do reprodukčného procesu, behom ktorého sú generované nové riešenia. Populácia riešení sa bežne nazýva populáciou individuí alebo chromozómov. Reprodukčný proces je založený na dvoch hnacích silách: variačné operátory kríženia a mutácie, ktoré prinášajú rozmanitosť populácie/diverzitu a selekcia, ktorá uprednostňuje kvalitných jedincov. Kombinácia variácie a selekcie prispieva obecne k zlepšeniu fitness funkcie jedincov v novo sa tvoriacej populácii. V procese kríženia jedincov, tak ako v živej prírode sú nové indivíduá získavané krížením väčšinou dvojíc rodičovských indivíduí. Všetky komponenty evolučného procesu sú stochastické – častejšie napr. vstupujú do reprodukčného procesu páry s lepšou fitness funkciou, ale aj slabší jedinci majú šancu vstúpiť do reprodukčného procesu.

Evolučné algoritmy pozostávajú z nasledujúcich komponent:

- Reprezentácia riešení/indivíduí – spôsob zakódovania,
- Funkcia ohodnotenia kvality riešenia – fitness funkcia,
- Populácia,
- Techniky výberu rodičovských indivíduí,
- Variačné operátory selekcie a mutácie a
- Obnova populácie.

Konkrétna špecifikácia jednotlivých komponent je odlišná pre rôzne typy evolučných algoritmov. Majú však spoločný obecný algoritmus 2.1.1 [7].

Algoritmus 2.1.1.

```
begin
  t := 0;                    // nastavenie počiatočného času
  initpopulation P(t);       // inicializácia, náhodné generovanie populácie
  evaluate P (t);            // ohodnotenie počiatočných jedincov populácie
  while not finished do      // test na ukončovacie kritérium
    t := t + 1;              // inkrementácia čísla populácie
    P' := selectpar P (t);   // selekcia rodičov
    recombine P' (t);        // rekombinácia vybraných rodičov
    mutate P' (t);           // náhodná mutácia potomkov
    evaluate P' (t);         // ohodnotenie novovzniknutých potomkov
    P := survive P,P' (t);   // obnova populácie pre d'alšiu generáciu
  od
end
```

Zo začiatku boli evolučné algoritmy používané iba na optimalizáciu, tj. k nájdeniu optimálnych hodnôt parametrov optimalizovaného systému. V posledných rokoch sa však začínajú používať aj na návrhy systémov, kde okrem parametrov je hľadaná aj štruktúra výsledného systému. Nachádzajú uplatnenie všade tam, kde klasické analytické či heuristické metódy nepostačujú, teda hlavne pri riešení zložitých nelineárnych úloh. Používajú sa napr. pri numerickej a kombinatorickej optimalizácii, modelovaní a identifikácii modelu, plánovaní a riadení, inžinierskom návrhu, dolovaní dát, strojovom učení a umelej inteligencii [7]. Tieto úlohy majú podobné vlastnosti:

- Priestor riešení je príliš rozsiahly a chýba expertná znalosť, ktorá by umožnila zúžiť priestor sľubných riešení.
- Nie je možné urobiť matematickú analýzu problému.
- Tradičné metódy zlyhávajú.
- Ide o úlohy s mnohočetnými extrémami, kritériami a obmedzujúcimi podmienkami.

Evolučné algoritmy však majú aj svoje nevýhody. Kvalitu riešenia je možné hodnotiť iba relatívne, nie je možné otestovať či sa jedná o globálne optimum. Pre mnohé úlohy je typická veľká časová náročnosť, pre príliš rozsiahle úlohy poskytuje riešenia príliš vzdialené od optima. Ukončenie optimalizácie je explicitné na základe časového limitu alebo stagnácie kriteriálnej funkcie.

2.2 Genetické programovanie

Genetické programovanie (GP) pracuje s tzv. spustiteľnými štruktúrami. Najčastejšie sa jedná o programy, ktoré sú reprezentované stromami. Tieto štruktúry majú typicky premennú dĺžku. Ďalším príkladom spustiteľnej štruktúry je elektronický obvod, mechanický systém alebo optický systém. Genetické operátory pracujú nad týmito spustiteľnými štruktúrami. Okrem bežných operátorov, ako sú kríženie a mutácia, existuje aj rada pokročilých operátorov umožňujúcich generovať programy s podprogramami, modulmi a podobne. V rámci zistenia fitness hodnoty je spustený kód kandidátneho programu pre definovanú množinu vstupov a sú vyhodnotené získané výsledky.

Pred začatím riešenia úlohy pomocou GP je potrebné uskutočniť päť prípravných krokov: definovať množinu terminálov, definovať množinu funkcií, definovať spôsob výpočtu fitness, definovať parametre GP (veľkosť populácie, počet generácií a podobne), definovať spôsob určenia výsledku a spôsob ukončenia evolúcie.

Tradičné GP používa generácie bez prekrývania, tj v GP existujú presne definované a oddelené populácie. Každá generácia je reprezentovaná celou populáciou jedincov. Nová populácia je vytváraná zo starej populácie, ktorú nahradzuje. Algoritmus prebieha v piatich krokoch [7]:

1. Inicializácia populácie.
2. Vyhodnotenie jedincov v existujúcej populácii a priradenie fitness hodnoty každému jedincovi.
3. Kým nie je nová populácia plne obsadená, opakujeme nasledujúce kroky. Vyberieme jedinca alebo niekoľko jedincov zvoleným selekčným algoritmom, s vybranými jedincami urobíme genetické operácie a výsledok genetických operácií vložíme do novej populácie.

4. Ak je splnené kritérium pre ukončenie, pokračujeme nasledujúcim krokom. Inak nahradíme existujúcu populáciu novou populáciou a pokračujeme krokom 2.

5. Jedinec s najvyššou hodnotou fitness je výsledkom algoritmu.

Množina terminálov reprezentuje vstupy do programu, ktorý je vyvíjaný pomocou GP, konštanty a funkcie bez argumentov s vedľajším účinkom. Pojem terminál sa používa preto, že premenné, konštanty a funkcie bez argumentov tvoria v stromovej štruktúre terminálne symboly alebo tiež listy stromu. Keď sú terminály spracovávané, vrátia programu určitú hodnotu. Jednou z úloh, ktorú terminál zastáva, je vstup do programu – premenná. Premenné hrajú dôležitú úlohu v procese učenia, kedy sa práve cez ne dostávajú do programu data z trénovacej množiny. Pomocou terminálov je možné do programu zaviesť aj konštanty. Jednou možnosťou je zvolenie množiny konštánt (napr. určitá podmnožina reálnych čísel), ktoré si zachovajú svoju hodnotu počas celého behu GP a pri generovaní stromu, keď sa pridáva konštanta, je vytvorená kombináciou týchto konštánt a aritmetických funkcií. Druhou možnosťou je generovať konštanty celkom náhodne [8].

Množina funkcií môže byť špecifická pre oblasť, v ktorej je GP použité, alebo môže byť obecná. Je možné použiť ľubovoľnú programovú konštrukciu známu z programovacích jazykov. Je však potrebné brať ohľad na to, aby funkcia nebola príliš zložitá a jej vyčíslenie nebolo príliš časovo náročné. V GP je nutné používať tzv. *chránené varianty* niektorých funkcií, ktoré nie sú definované pre určité hodnoty. Typickým príkladom je delenie nulou alebo odmocnina záporného čísla. V takýchto prípadoch funkcia musí vrátiť nejakú bezpečnú hodnotu, napr. nulu alebo veľmi vysoké číslo. Množina funkcií teda typicky obsahuje štandardné aritmetické a logické funkcie, ďalej konštrukcie známe z programovacích jazykov (napr. cykly, skoky a podobne) a funkcie z konkrétnej aplikačnej domény (napr. z oblasti robotiky: prečítaj hodnotu i-tého senzoru alebo otoč ľavé koleso o x stupňov vľavo) [7].

Funkcie a terminály pre GP by mali byť volené tak, aby bolo pomocou nich možné čo najlepšie reprezentovať riešenie daného problému. Ak použijeme malú množinu funkcií, nebudeme schopní riešiť príliš zaujímavé problémy. Naopak tiež nie je vhodné použiť príliš veľkú množinu funkcií, pretože sa tým zväčšuje prehľadávací priestor a nemusíme byť schopní nájsť požadované riešenie. Množina funkcií by teda mala byť vyvážená s ohľadom na riešený problém.

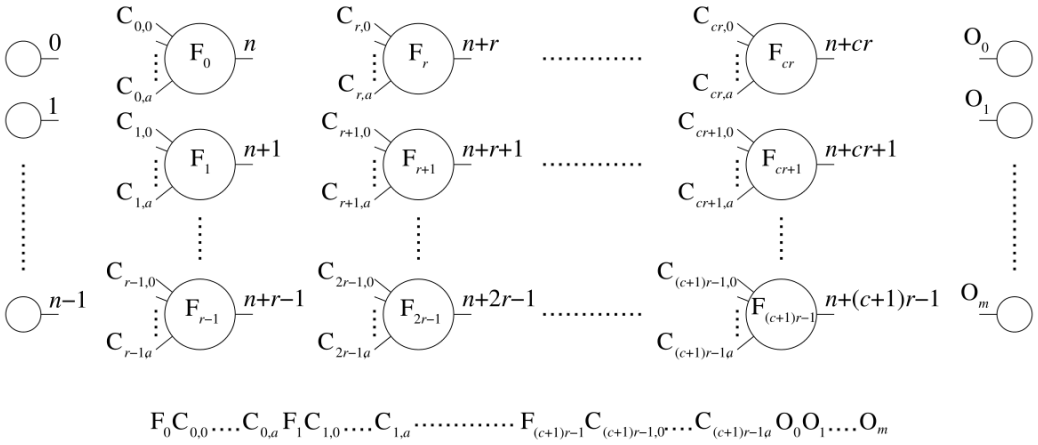
Programy v GP sú štruktúry pozostávajúce z funkcií a terminálov. Ďalej sú definované pravidlá určujúce, kedy a ako budú jednotlivé funkcie a terminály použité. Základnými troma štruktúrami používanými v GP sú stromy, lineárne štruktúry a obecné grafové štruktúry. Vo väčšine stromovo orientovaných GP systémov genotyp a fenotyp jedinca splývajú. Priechodom stromu dôjde k vykonaniu programu, ktorý strom reprezentuje. Lineárna reprezentácia je tvorená postupnosťou inštrukcií, ktorá je podobná zápisu programu v jazyku symbolických inštrukcií. Obecné grafy sa v genetickom programovaní začali hojne používať až v poslednej dobe. Príkladom systému s grafovou reprezentáciou je CGP [8].

2.3 CGP

CGP predstavil najprv J. Miller vo svojom článku Cartesian Genetic Programming v roku 1999 a potom spoločne s P. Thomsonom v roku 2000 [4]. Volá sa "kartézske", pretože ako reprezentáciu problému používa dvojdimenzionálnu mriežku prvkov, nazývanú aj matica. Program v CGP je reprezentovaný ako orientovaný acyklický graf, takže nie je povolená

spätná väzba, s pevným počtom uzlov. Gény, ktoré tvoria genotyp v CGP, sú celé čísla, ktoré reprezentujú, kde uzly získajú svoje dáta, ktoré operácie na nich vykonajú a kde je možné získať výstupné dáta. Ak sa genotyp dekóduje, niektoré uzly môžu byť vynechané, pretože sa nepoužijú pri výpočte výsledku. Takéto uzly a ich gény sa nazývajú nekódové. Výsledok dekódovania genotypu sa nazýva fenotyp. Vzhľadom na možnú existenciu nekódových uzlov v genotype môže mať fenotyp rôzny počet uzlov, od žiadnych uzlov po celkový počet uzlov z genotypu. Mapovanie genotyp–fenotyp použité v CGP je jedna z jeho definujúcich vlastností [5].

Každý prvok je reprezentovaný viacerými génmi. Má definovanú funkciu pomocou *funkčného génu* a vstupy pomocou *spájacích génov*. Funkcia je definovaná v tabuľke od užívateľa a spracováva vstupy na výstupy. Datové výstupy prvkov sú číslované sekvenčne stĺpec za stĺpcom a tieto čísla sa potom používajú v spájacích génoch. Výstupy celého genotypu fungujú na rovnakom princípe, sú to spájacie gény pripojené na koniec genotypu. Obecná forma programu v CGP je na obrázku 2.1 prevzatom z [5]. Je to mriežka uzlov, ktorých funkcie sú vyberané z množiny primitívnych funkcií, ktoré zadáva užívateľ. Mriežka má n_c stĺpcov a n_r riadkov. Z tohto je možné vypočítať maximálny povolený počet prvkov $L_n = n_c n_r$.Počet programových vstupov je n_i a počet výstupov je n_o. U každého uzlu je predpokladané, že používa toľko vstupov, koľko je maximálna arita a funkcie. Každý dátový vstup a uzlový výstup je označený sekvenčne, začínajúc nulou, čo dáva unikátnu dátovú adresu, ktorá špecifikuje, kde sú dostupné vstupné dáta alebo výstupná hodnota uzlu.



Obrázek 2.1: Obecná forma CGP (prevzaté z [5]).

Užívateľ definuje pred spustením počet stĺpcov, riadkov a l-back parameter. Posledný parameter určuje, o koľko stĺpcov smerom k primárnym vstupom je možné pripojiť vstup uzlu. Pokiaľ $l = 1$, tak prvok môže mať na vstupe iba primárny vstup alebo výstupy prvkov v najbližšom stĺpci naľavo od neho. Maximálna možná hodnota pre l je počet stĺpcov.

Hodnoty, ktoré môžu gény nadobudnúť, sa nazývajú alely. V prípade CGP sú to celé čísla, ktoré sú obmedzené podľa svojej funkcie. Funkčné gény musia nadobúdať iba hodnoty z tabuľky funkcií. Alely spájacích génov musia rešpektovať veľkosť genotypu a l-back parameter. Najprv musia alely funkčných génov f_i obsahovať platné adresy vo vyhľadávacej tabuľke primitívnych funkcií. Nech n_f reprezentuje počet povolených primitívnych funkcií uzlu. Potom f_i musí spĺňať podmienku

$$0 \leq f_i < n_f.$$

Uvažujme uzol v stĺpci j. Hodnoty zo spájacích génov C_{ij} všetkých uzlov v stĺpci j sú tieto: ak $j \geq l$, tak platí

$$n_i + (j - l)n_r \leq C_{ij} \leq n_i + jn_r.$$

Ak $j < l$, tak platí

$$0 \leq C_{ij} \leq n_i + jn_r.$$

Výstupné gény O_i sa môžu pripojiť na ktorýkoľvek uzol alebo vstup za týchto podmienok:

$$0 \leq O_i < n_i + L_n.$$

2.4 Evolučný algoritmus CGP

Algoritmus CGP je založený na evolučnej stratégii (ES), konkrétne na variante $(1 + \lambda)$. Pri tomto spôsobe tvorí novú populáciu λ novovzniknutých mutantov najlepšieho rodiča aj ich rodič. Veľkosť populácie je teda $\lambda + 1$ a na základe skúseností sa λ volí okolo 4 [14]. Ako operátor CGP sa používa iba mutácia, ktorá pracuje tak, že náhodne zmení hodnotu génu na inú. Samozrejme s ohľadom na to, aké hodnoty sú v tomto géne prípustné. Tento operátor je riadený parametrom, ktorý udáva percento mutovaných génov, v našom prípade je jeden gén jedna celočíselná hodnota.

Celý algoritmus sa dá popísať pomocou niekoľkých krokov [14] :

1. Náhodné vygenerovanie inicializačnej populácie
2. Ohodnotenie všetkých jedincov populácie pomocou fitness funkcie (viď kapitola 3.3)
3. Výber najlepšie ohodnoteného jedinca do novej populácie
4. Vygenerovanie λ potomkov mutácií nájdeného najlepšieho jedinca
5. Pokiaľ nie je splnená podmienka pre ukončenie, pokračuje sa krokom 2

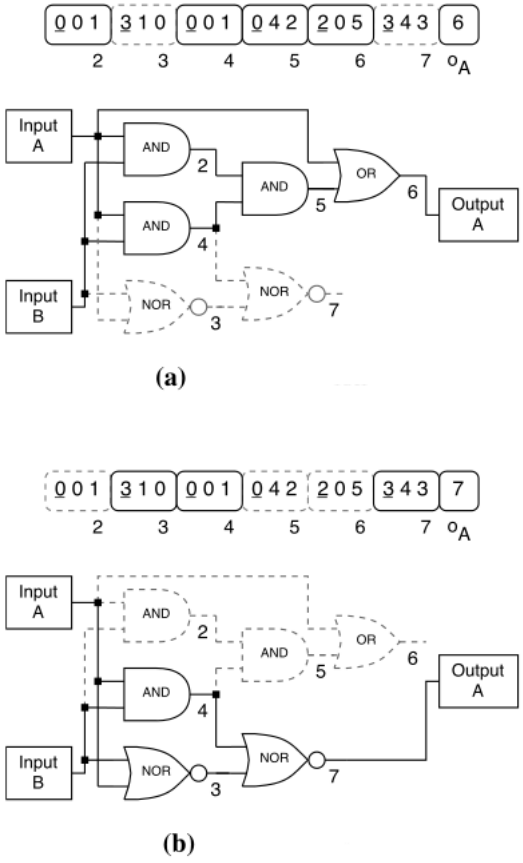
Tretí krok je kritický. Je potrebné vybrať vždy rôzneho jedinca od toho pôvodného, aj keď majú rovnakú fitness, aby nedochádzalo k degradácii a spomaleniu evolúcie, pretože by v populácii nebol prítomný žiaden nový genetický materiál [14].

2.5 Mutácia

Mutácia použitá v CGP je bodová. Znamená to, že alela na náhodnej génovej pozícii je zmenená na inú náhodnú, ale validnú, hodnotu. V prípade funkčného génu je to hodnota z tabuľky funkcií, v prípade spájacieho génu je to validné čislo výstupu niektorého z predchádzajúcich prvkov, prípadne primárneho vstupu.

Užívateľ určuje počet mutácii v jednej mutačnej operácii. Ovplyvňuje to rýchlosť zmien a mutácie. Zvykne sa udávať v percentách z počtu génov v genotype.

Na obrázku 2.2 je vidieť ako môže jediná mutácia v genotype urobiť veľkú zmenu vo fenotype. Príklad je prevzatý z [5].



Obrázek 2.2: Ukážka mutácie v CGP. Obrázok označený a) je pred mutáciou, b) je po mutácii. Zmena nastala iba v spájacom géne výstupu genotypu, avšak úplne to zmenilo vzhľad obvodu a okrem jedného hradla sa všetky kódové hradlá zmenili na nekódové a naopak. Nekódové sú vyznačené prerušovanou čiarou.

2.6 Rekombinácia

Zistilo sa, že rekombinačný bodový operátor má nepriaznivé účinky na podgrafy v chromozóme, ako aj na výkon CGP. Práca Cleggovej a kol. [2] skúmala použitie kríženia v CGP pomocou kríženia s desatinnou čiarkou, podobného tomu v evolučnom programovaní, a pridania ďalšej vrstvy zakódovania genotypu, avšak ani tento pokus nebol príliš úspešný a na vhodné použitie rekombinácie sa ešte stále čaká.

2.7 Parametre CGP

Nastavenie parametrov pre CGP vyžaduje experimentáciu v závislosti na probléme, avšak existujú nejaké rady, čo použiť. Četnosť mutácie závisí na veľkosti genotypu a odporúčané je 1% mutácie, ak je použitých maximálne 100 prvkov v genotype [5]. Ak má každý prvok 3 gény a v genotype je ich presne 100, tak mutovať budú maximálne 3 gény. Pri vyšších počtoch prvkov by frekvencia mutácie mala byť aj nižšia než 1%, zato pri nižších počtoch prvkov môže byť aj vyššia frekvencia.

Ďalším dôležitým parametrom je počet stĺpcov a riadkov v genotype. Ak nie je požadova-

ná obdĺžniková štruktúra, tak je odporúčané použiť jeden riadok a adekvátny počet stĺpcov. V niektorých prípadoch, napr. u CGP v evolučných obvodoch, zvykne byť užitočné použiť rovnaký počet stĺpcov aj riadkov [5]. Všetko sú to však len odporúčania, žiadna detailná práca na túto tému nebola publikovaná.

Veľkosť populácie je ďalším z parametrov. V CGP sa používajú malé populácie, napr. päť jedincov, avšak vzhľadom na to sa dá očakávať veľa generácií. Priemerný počet výpočtov fitness je porovnateľný s inými prístupmi ku genetickému programovaniu [5].

2.8 Seba-modifikujúce CGP

V biológii vidíme, že čas je dôležitým aspektom mapovania genotypu na fenotyp. Najjednoduchšie sa to predstavuje na mnohobunkových organizmoch. V tomto prípade fenotyp obsahuje veľa buniek, aj keď používa jeden genotyp. Tým pádom je fenotyp konštruovaný v čase, ovplyvňovaný nielen génmi, ale aj okolím. Tieto princípy využíva seba-modifikujúce CGP. Povolením nových funkcií zavedieme čas aj do CGP. Znamená to, že kód bude môcť meniť aj sám seba a tak budú vznikať jedinci s rôznymi genotypmi, napríklad pribudne nejaký primárny vstup alebo výstup a môžu tak vznikať nové riešenia. Každé použitie nových funkcií sa nazýva iterácia [3].

Proces konsštrukcie fenotypu prebieha vo fázach. Najprv sa vytvorí presná kópia genotypu a nazve sa fenotyp v iterácii 0. Potom sú aplikované seba-modifikujúce funkcie, ktoré vytvoria fenotyp ďalšej iterácie. Vypočíta sa zlepšenie fitness a tento cyklus sa opakuje až kým nedosiahne zadaný počet iterácií alebo kým má k dispozícii nejaké seba-modifikačné funkcie. Na záver sa vypočíta fitness fenotypu. Tento postup predpokladá, že užívateľ chce vyriešiť množinu problémov (napr. vypočítanie parity pre rôzny počet vstupov, pričom počet hradiel sa môže v priebehu evolúcie zmeniť), nie iba jeden problém ako je obvyklé pre CGP.

2.9 Kódovanie prepojenia elementov (chromozóm)

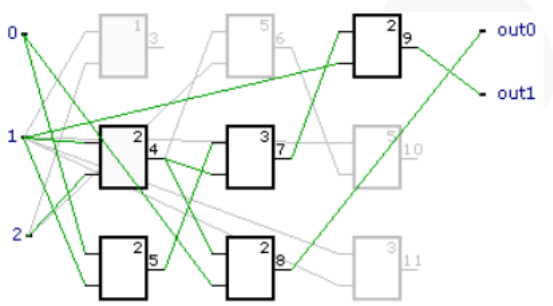
Zakódované kandidátne riešenia problému sú v našom prípade chromozómy. Jeden chromozóm tvorí $n_c \times n_r \times (n_a + 1) + n_o$ celočíselných hodnôt, kde n_a je počet vstupov elementu a n_o je počet výstupov celého obvodu. Pre jednoduchú orientáciu majú všetky výstupy elementov a vstupy kombinačného obvodu svoje jednoznačné očíslovanie. Číslуje sa po stĺpcoch zľava doprava — najprv vstupy kombinačného obvodu, potom výstupy jednotlivých elementov, až po výstupy kombinačného obvodu.

Hodnoty génov udávajú čísla výstupov privedených na vstupy konkrétneho elementu, posledné číslo špecifikuje logickú funkciu, ktorú element realizuje (napr. AND, OR, XOR...). Na konci chromozómu sú ďalšie hodnoty — pre každý výstup kombinačného obvodu určujú, výstup ktorého elementu je naňho napojený.

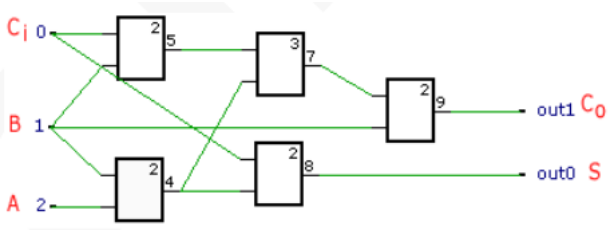
Príklad 2.9.1. Ukážkový chromozóm 3×3 (prevzatý z [14]):
(1,2,1)(1,2,2)(0,1,2)(4,2,5)(5,4,3)(4,0,2)(7,1,2)(1,6,5)(1,1,3)(8,9)

V príklade 2.9.1 je zápis ukážkového chromozómu, ktorý tvoria elementy s dvoma vstupmi a s dvoma primárnymi výstupmi kombinačného obvodu. Tento chromozóm je tvorený maticou 3×3, má tri vstupy kombinačného obvodu a teda index výstupu prvého elementu je 3. Elementy, ktoré sa nepodieľajú na riešení, sú vykreslené šedou farbou na obrázku 2.3.

Na obrázku 2.4 je rovnaké prepojenie, avšak hradlá sú inak umiestnené (podľa oneskorenia) — sú vynechané tie, ktoré boli šedo vykreslené v predchádzajúcom obrázku a sú dodané významy vstupov a výstupov. Tu už je vidieť, že ide o štandardné zapojenie jednobitovej úplnej sčítačky, ktorá využíva štyri hradlá typu XOR (funkcia 2) a jedno hradlo typu AND (funkcia 3).



Obrázek 2.3: Dekódovaný chromozóm [14]



Obrázek 2.4: Prekreslené schéma zapojenia [14]

Kapitola 3

CGP v evolučnom návrhu obvodov

Evolučný prístup k návrhu obvodov je výhodný v tom, že umožňuje prehľadať väčší stavový priestor než konvečný prístup k návrhu, ktorý vyžaduje veľa analytickej práce, skúseností a času. Pomocou evolučných algoritmov je možné dosiahnuť nielen optimalizáciu nejakého už existujúceho obvodu, ale aj úplne nové funkčné riešenie. Pretože v evolučnom návrhu nie sú kladené prísne obmedzenia na typ komponent a množinu transformácií, ako to robia konvenčné metódy, môže prístup založený na metóde "generuj a testuj"pracovať s ľubovoľnými komponentami, o ktorých predpokladáme, že sú pre riešenie daného problému vhodné. Okrem hradiel je možné priamo pracovať aj so zložitejšími logickými obvodmi, napr. so sčítačkami, násobičkami a pod. To je nemysliteľné pre konvenčné minimalizačné metódy [8].

CGP je vďaka tvaru svojich jedincov veľmi vhodné pre evolučný návrh digitálnych obvodov. Konkrétne použitie a špecifikácia sa nachádzajú v ďalších podkapitolách.

3.1 Logické obvody

Logické obvody sú také obvody, ktoré pracujú s dvoma diskrétnymi hodnotami signálu. Každý signál je nastavený buď na logickú nulu alebo logickú jedničku. Takéto obvody sú vytvárané pomocou základných logických členov – hradiel, ktoré vykonávajú logické operácie nad svojimi vstupmi, napr. dvojvstupový logický člen AND, ktorý realizuje logický súčin nad svojimi vstupmi [8].

Logické signály majú presne definované úrovne, napr. logická nula môže byť reprezentovaná napätím 0-2V a logická jednička napätím 3-5V. Pokiaľ je na výstupe hradla nameraná iná hodnota (napr. 2-3V), je to chápané ako nedefinovaná hodnota. Pre logické členy je charakteristické určité oneskorenie, s čím súvisí maximálna frekvencia, na ktorej môžu pracovať. Pre svoju činnosť vyžadujú určitý príkon. Existujú aj obmedzenia na počet iných logických členov, ktoré je možné pripojiť na vstup, prípadne výstup, logického členu s istotou, že obvod bude správne pracovať. Do istej miery sú logické členy tolerantné k výkyvom napájacieho napätia, teploty, elektromagnetického rušenia a podobne. Všetky parametre logických členov sú uvedené v katalógu dodávanom výrobcom a návrhár by ich mal rešpektovať.

Logické obvody môžu byť modelované pomocou výrazov booleovskej algebry, takže vytváranie logických obvodov priamo odpovedá vytváraniu logických výrazov pomocou prostriedkov Booleovej algebry. Členom AND, OR, NOT odpovedajú operácie $\cdot$, $+$ a^{-}(komplement), prvky 0 a 1 majú význam logickej nuly a logickej jedničky.

3.2 Kombinačné obvody

Logické obvody rozlišujeme podľa ich správania na sekvenčné a kombinačné. Pre sekvenčné obvody platí, že výstupné hodnoty závisia na kombinácii vstupných hodnôt a tiež na určitej postupnosti (sekvencii) predchádzajúcich vstupných hodnôt. Pre kombinačné obvody platí, že hodnoty výstupných premenných sú iba funkciou vstupných premenných.

Kombinačný obvod s n vstupmi a m výstupmi najčastejšie špecifikujeme a modelujeme pomocou logických funkcií $f_1, ..., f_m$ tvaru $f_i : \{0,1\}^n \rightarrow \{0,1\}$, kde $i = 1, ..., m$ [8]. Logická funkcia je definovaná booleovským výrazom, ktorý môže obsahovať logické premenné, logické spojky AND a OR, unárnu operáciu negácie (NOT) a logickú 0 a 1. Najčastejšie sa používa dvojstupňová reprezentácia, kde sú premenné kombinované pomocou jedného operátoru a vzniknuté termy sú skombinované pomocou operátoru druhého. Pokiaľ je výraz zapísaný ako suma súčinov, každý zo súčinov obsahuje všetky vstupné premenné (buď priamo alebo v negácii) a všetky súčiny reprezentujú pravdivé ohodnotenie f_i, potom je f_i zapísaná v kanonickom tvare, ktorý nazývame *úplna normálna disjunktná forma* [8]. Obdobne je možné definovať úplnú normálnu konjunktnú formu. Existujú aj iné spôsoby špecifikácie, napríklad pomocou pravdivostnej tabuľky, logickej mapy, Vennovho diagramu, hyperkocky alebo binárneho rozhodovacieho diagramu. Cieľom návrhu kombinačného obvodu je pre zadanú špecifikáciu vytvoriť takú implementáciu, ktorá bude nielen plniť požadované kombinačné chovanie, ale súčasne bude aj splňovať celú radu ďalších obmedzení. Typicky je požadovaná implementácia vyžadujúca minimálny počet logických členov (minimalizácia plochy na čipe), s minimálnym príkonom, s minimálnym oneskorením a podobne. Často sú tieto požiadavky protichodné a musí sa hľadať vhodný kompromis.

Minimalizačné metódy predpokladajú, že je zadaná funkcia určitým spôsobom reprezentovaná. Tiež definujú konečnú množinu pravidiel, ktoré je možné použiť k transformácii jedného tvaru funkcie na iný tvar funkcie. Pre každú metódu existuje algoritmus, ktorého prostredníctvom prebehnú vhodné transformácie tak, aby bola získaná čo najvýhodnejšia implementácia.

Algebraické minimalizačné metódy sú založené na aplikácii axiómov a teorémov Booleovej algebry, kedy sa snažíme minimalizovať počet výskytov premenných a počet logických operátorov vo vstupnom výraze, ktorý je obvykle zadaný v úplnej normálnej forme. Na počítači sa najčastejšie implementujú algoritmické minimalizačné metódy, ktoré používajú takú reprezentáciu logickej funkcie, ktorá umožní jednoducho odhaliť *booleovskú susednosť* (booleovská susednosť umožňuje eliminovať premennú z logického výrazu, napr. $xy + x\overline{y} = x$). Typickými príkladmi minimalizačných metód sú metódy Quine-McCluskey a Espresso, ktoré sú založené na reprezentácii logickej funkcie pomocou tabuľky. Problémy minimalizácie patria obecne do triedy NP a preto sa pri veľkom počte premenných používajú rôzne heuristiky umožňujúce nájsť dostatočne kvalitné suboptimálne riešenie v rozumnom čase [8].

Všetky uvedené minimalizačné metódy majú jednu dôležitú vlastnosť – pomerne mnoho predpokladajú o tvare výsledného riešenia. Ak pracujeme s disjunktnou normálnou formou, výsledný výraz musí byť zapísaný ako suma súčinov. Existujú však logické obvody, ktoré nie je možné pomocou zvolenej metódy optimalizovať. Typickým príkladom je paritný obvod, ktorý dáva na výstup logickú jedničku, ak je počet jedničiek na vstupe nepárny. Pre paritný obvod s tromi vstupmi a, b, c dostávame úplnu normálnu disjunktnú formu v tvare $f_p = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}b\overline{c} + abc$. Tento výraz už nie je možné zjednodušiť, pretože v ňom neexistuje booleovská susednosť. Pre implementáciu, ktorá by vychádzala z tohto výrazu, by sme potrebovali tri invertory, štyri trojvstupové členy AND a jeden štvorvstupový člen OR. Tento obvod je však

možné implementovať aj lacnejšie pomocou dvoch dvojvstupových logických členov XOR, kedy každý z nich realizuje funkciu $x \oplus y = x\overline{y} + \overline{x}y$. Pretože však výsledok očakávame v tvare sumy súčinov a metóda nevie priamo pracovať s logickou funkciou neekvivalencie, nebude táto výhodná implementácia nájdená. Tento problém rieši evolučný návrh obvodov, ktorý neočakáva určitý definovaný tvar výsledného riešenia a dokáže teda nájsť aj veľmi výhodné riešenia, ktoré sú konvenčnými metódami nedosiahnuteľné. CGP je konkrétnym prípadom evolučného návrhu obvodov. Pri pohľade na obecnú formu CGP, uvedenú v predchádzajúcej kapitole, je vidieť, že pripomína obecný obvod (mriežka hradiel so vstupmi a výstupmi), takže obecné zakódovanie je veľmi jednoducho použiteľné aj pre kombinačné obvody.

3.3 Fitness funkcia pre kombinačné obvody

Vzhľadom na to, že počas evolúcie je potrebné určiť ako dobré je určité zapojenie (chromozóm), je potrebná funkcia, ktorá dokáže spravodlivo ohodnotiť vzniknutých jedincov. Táto funkcia sa volá **fitness funkcia**.

Výpočet fitness hodnoty sa robí postupným nastavovaním všetkých možných vstupných kombinácií podľa pravdivostnej tabuľky. Pre tieto kombinácie sa potom podľa zapojenia vypočítajú výstupy, tj. simuluje sa činnosť kombinačného obvodu a porovnávajú sa očakávané výsledky so simulovanými. V prípade úplnej definície sa jedná o 2^n kombinácií, kde n je počet vstupov [14].

Hodnota fitness je potom rovná počtu zhôd simulovaných výsledkov s výsledkami očakávanými, vzorec 3.1, kde b je počet správnych bitov vo výsledkoch všetkých možných priradení na vstupoch, z je počet hradiel použitých v konkrétnom kandidátnom obvode. Ak máme napríklad 4 vstupy a 2 výstupy, tak počet vstupných kombinácií je $2^4 = 16$. Maximálny fitness teda môže byť $16 * 2 = 32$ zhôd [14].

$$fit1 = \begin{cases} b, & b < n_o 2^{n_i} \\ b + (n_c n_r - z), & inak \end{cases} \tag{3.1}$$

Toto samozrejme nie je jediná možnosť fitness funkcie. Môže sa porovnávať počet použitých elementov, spotreba, oneskorenie a pod., avšak je nutné brať do úvahy problémy so škálovateľnosťou. Čas evaluácie fitness totiž rastie exponenciálne s počtom vstupov [9].

Kapitola 4

Existujúce nástroje pre vizualizáciu CGP

4.1 Implementácie CGP

Zhrnutie existujúcich implementácii CGP je v [6]. Sú to tieto:

- Implementácia v programovacom jazyku C, autor Julian Miller, dostupné na https://sites.google.com/site/julianfrancismiller/professional.
- Implementácia v programovacom jazyku C, autori Zdeněk Vašíček a Lukáš Sekanina, dostupné na http://www.fit.vutbr.cz/research/viewproduct.php.en?id=61&notitle=1.
- Implementácia v programovacom jazyku Java, autor David Oranchak, dostupné na http://oranchak.com/cgp/contrib-cgp-18.zip, dokumentácia na http://oranchak.com/cgp/doc/. Tento program je možné použiť na tieto problémy: symbolická regresia, klasifikácia dúhovky a rakoviny pŕs, rôzne paritné problémy.
- Implementácia v programe MATLAB, autor Jordan Pollack, dostupné na https://sites.google.com/site/julianfrancismiller/publications. Použiteľné na problémy symbolickej regresie.
- Implementácia v programovacom jazyku Java, autor Laurence Ashmore, dostupné na http://www.emoware.org/evolutionaryart.asp. Zamerané na evolučné umenie pomocou CGP.

4.2 Nástroje pre vizualizáciu

Pre vizualizáciu rôznych druhov evolučných algoritmov existuje veľké množstvo programov. Tu sú vybrané príklady (máj 2012):

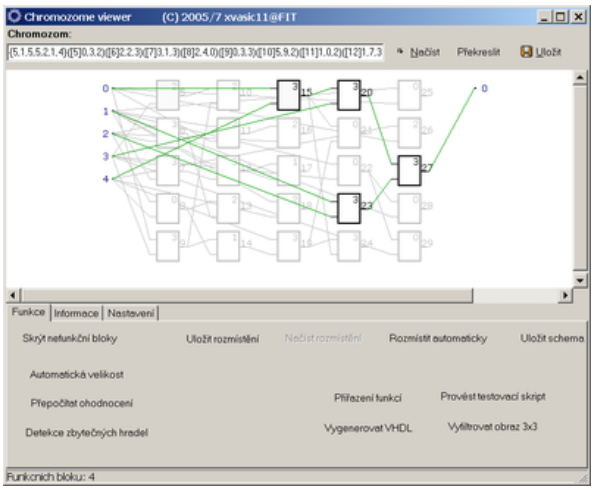
- Vizualizácia v programovacom jazyku Java, autor Johannes Sarg, dostupné na https://www.ads.tuwien.ac.at/raidl/tspga/TSPGA.html. Zamerané na problém obchodného cestujúceho (angl. Traveling Salesman problem, TSP).
- Vizualizácia v programovacom jazyku Python, autori Balint Seeber a Janice Leung, dostupné na http://spench.net/drupal/software/geneticteapot. Zobrazuje robota, ktorý nasleduje steny (angl. wall-following robot).

- Vizualizácia v programovacom jazyku C, autori Tanja Dabs a Jochen Schoof, viac informácii na http://joscho.de/PUB/tr098.pdf.
- Vizualizácia v programovacom jazyku C, autor Ariel Dolan, dostupné na http://www.aridolan.com/ga/gaa/gaa.html.
- Vizualizácie v programovacom jazyku Lisp, autor: George P. W. Williams, dostupné na http://common-lisp.net/project/geco/.

4.3 Nástroje pre vizualizáciu CGP

Nástroj na vizualizáciu CGP, tj. všetkých generácií, s možnosťou manipulácie a zvýrazňovania jednotlivých častí sa nepodarilo nájsť. Pravdepodobne vôbec neexistuje. Avšak aspoň pre zobrazenie jednotlivých chromozómov existujú dva programy. Názov majú rovnaký – cgpviewer – avšak ich funkcie sa trochu líšia.

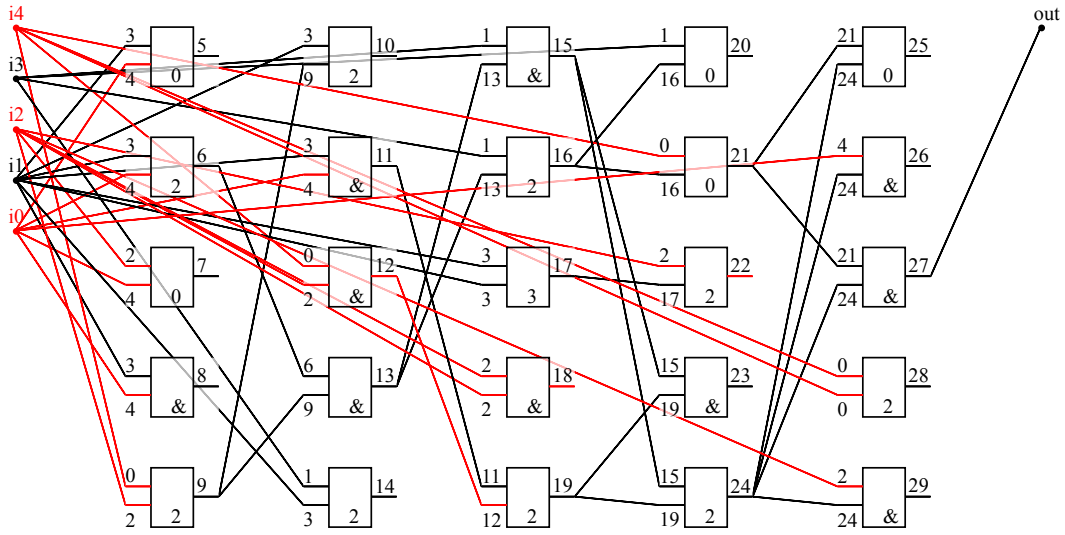
Prvý cgpviewer [13], viď obrázok 4.1, má možnosť skryť nefunkčné bloky, čo v tomto prípade znamená, že zošednú. Má však aj možnosť automatického rozmiestnenia podľa oneskorenia hradiel, ktoré všetky prebytočné bloky skryje a zobrazí len tie, ktoré sa podieľajú na výsledku. Je to optimalizácia kvôli prehľadnosti. Spoje medzi hradlami sa správajú adekvátne k súčasnému zobrazeniu, tj. môžu tiež zošednúť alebo sa skryť. Jednotlivým blokom je možné priradiť funkcie (napr. AND, NOR, XOR) a vďaka tomu ich potom nahradiť schématickou značkou vybraného hradla. Logické obvody je tiež možné simulovať. Čo sa týka možností exportu, tento program dokáže vygenerovať VHDL a DOT, uložiť obrázok do formátu BMP a počas práce s programom je schopný si zapamätať jedno rozmiestnenie hradiel (napr. ak chcete vyskúšať nejaké iné zobrazenie a radi by ste sa potom prípadne vrátili k pôvodnému).



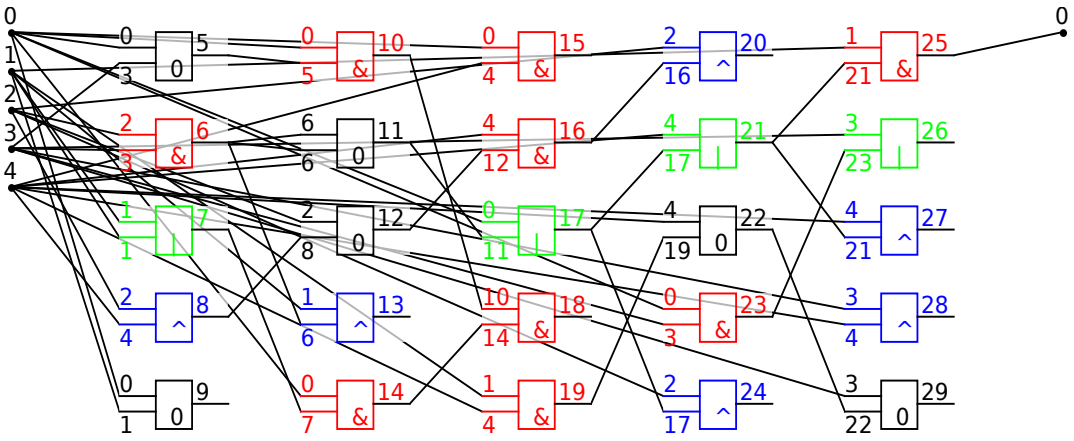
Obrázek 4.1: Vzhľad programu cgpviewer [13].

Druhý cgpviewer vznikol ako bakalárska práca [10]. Je napísaný v Jave kvôli prenositeľnosti na rôzne operačné systémy. Základom jeho funkcionality je zobrazenie chromozómu s možnosťou nechať nepoužité hradlá zošednúť alebo ich úplne skryť. Pre zobrazenie komplikovanejších chromozómov je k dispozícii možnosť priblížiť alebo oddialiť chromozóm. Pri zobrazovaní väčšieho chromozómu určite príde vhod možnosť vypnúť anti-

aliasing a tým zrýchliť vykresľovanie. Taktiež je možnosť posúvať jednotlivé hradlá v chromozóme, prípadne si ich opätovne nechať zarovnať na mriežku. Ďalšou funkciou je simulácia, ktorú je možné vidieť na obrázku 4.2. Simuluje sa podľa užívateľom zadaných funkcií, pričom musí zadať výstup pre každý možný vstup. Program počíta s hradlami, ktoré majú dva vstupy a jeden výstup, pričom nevadí ak sa použije iba jeden vstup (čím je možné vytvoriť aj hradlo NOT). S týmto súvisí aj farbenie hradiel podľa ich funkcie, ako je možné vidieť na obrázku 4.3. Farbu zadáva užívateľ. Aby bolo možné sa k rozpracovanému chromozómu vrátiť aj po čase, je k dispozícii export do interného formátu, ktorý vie uložiť všetky potrebné parametre pre opätovné zobrazenie. Taktiež je k dispozícii export do formátu SVG, čo je rozšírený vektorový formát. Z neho je možné samozrejme vytvoriť aj bitmapový obrázok v externom editore (napr. Gimp).



Obrázek 4.2: Simulácia. Červenou sú spoje, na ktorých je logická hodnota jedna. Čierne obsahujú nulu.



Obrázek 4.3: Zobrazenie chromozómu hneď po otvorení a zafarbení.

Kapitola 5

Analýza a návrh konceptu nástroja pre vizuálnu analýzu evolúcie obvodov

Oproti nástroju `cgpviewer` [13], ktorý zobrazuje len jedno kandidátne riešenie, je cieľom vizualizácie v tejto diplomovej práci získať prehľad o celom behu evolúcie, mať možnosť pozorovať zmeny v chromozómoch a umožniť ich preskúmanie. Vzhľadom na to, že žiaden vhodný program na takýto prehľad neexistuje, vzniklo zadanie tejto diplomovej práce. Pri návrhu treba brať do úvahy obmedzený priestor na ploche monitora počítača, dostupný výpočetný výkon pre zobrazovanie, veľkosť informácií o jednom behu evolúcie a podobne. Z toho sa dajú odvodiť niektoré obmedzenia.

5.1 Formát evolúcie a vstupné dáta

Na FIT existuje implementácia CGP pracujúca ako konzolová aplikácia [12], ktorá zachycuje priebeh evolúcie, pričom ten je možné si uložiť do súboru. Vstupné dáta pre vizualizáciu budú získavané z tohto programu po úprave. V tomto programe je potrebné zmeniť parameter počtu behov CGP na jedna a povoliť vypisovanie chromozómov v každej generácii. Takto upravený program však vypisuje svoj výstup na štandardný výstup, pre program je však potrebné, aby sa nachádzal v súbore. V operačnom systéme Linux sa dá veľmi jednoducho presmerovať výstup programu do súboru, čo je ideálne riešenie tohto problému. Vznikne nám súbor, v ktorom sa nachádzajú informácie o jednom behu CGP obecne a jednotlivé generácie, viď príklad 5.1.1.

Príklad 5.1.1.

```
<22, 0><5,1, 5,5, 2,1,0>([5]3,4,0)([6]0,2,2)([7]4,3,0)([8]1,0,2)([9]1,4,2)
([10]6,3,1)([11]2,9,0)([12]0,6,2)([13]5,4,0)([14]2,7,1)([15]4,12,2)
([16]0,2,3)([17]3,13,1)([18]12,11,1)([19]14,0,1)([20]19,18,2)([21]19,0,0)
([22]1,0,0)([23]17,17,2)([24]19,0,1)([25]3,21,0)([26]22,21,0)([27]4,24,1)
([28]0,21,3)([29]3,23,1)(0)
<18, 4><5,1, 5,5, 2,1,4>([5]2,2,0)([6]3,4,0)([7]4,2,0)([8]3,4,3)([9]3,4,0)
([10]3,0,1)([11]0,9,1)([12]5,8,2)([13]5,5,2)([14]0,3,3)([15]0,2,3)
([16]4,1,0)([17]14,2,3)([18]10,1,3)([19]11,14,2)([20]3,2,2)([21]2,4,1)
([22]17,1,2)([23]0,16,3)([24]16,18,2)([25]20,23,2)([26]24,20,1)
([27]24,21,0)([28]4,1,1)([29]22,0,0)(25)
```

```
<16, 1><5,1, 5,5, 2,1,1>([5]0,1,2)([6]2,2,3)([7]1,4,1)([8]1,2,0)([9]2,1,2)
([10]0,1,1)([11]9,6,2)([12]8,2,2)([13]4,4,3)([14]1,6,3)([15]14,1,1)
([16]10,0,0)([17]0,11,0)([18]13,10,2)([19]13,14,1)([20]3,19,1)([21]0,18,1)
([22]16,18,0)([23]15,2,2)([24]16,16,0)([25]0,20,3)([26]3,0,1)([27]22,2,1)
([28]22,0,2)([29]0,1,0)(13)
<22, 3><5,1, 5,5, 2,1,3>([5]4,3,2)([6]1,2,0)([7]4,2,0)([8]4,2,2)
([9]1,3,2)([10]8,0,1)([11]1,2,2)([12]1,6,3)([13]8,9,2)([14]8,0,2)
([15]2,11,3)([16]0,4,1)([17]0,13,2)([18]11,0,1)([19]2,11,0)([20]4,16,0)
([21]17,19,0)([22]19,0,2)([23]17,1,3)([24]19,1,3)([25]20,3,1)([26]4,21,1)
([27]21,23,2)([28]2,20,0)([29]22,24,0)(25)
<22, 1><5,1, 5,5, 2,1,1>([5]1,4,1)([6]3,0,1)([7]0,1,3)([8]2,4,3)([9]3,0,2)
([10]0,6,0)([11]4,2,1)([12]2,2,3)([13]6,9,3)([14]3,9,3)([15]13,10,1)
([16]3,0,1)([17]13,10,3)([18]11,14,2)([19]14,4,0)([20]3,4,2)([21]15,1,2)
([22]17,3,3)([23]18,0,2)([24]2,3,1)([25]3,2,2)([26]24,20,1)([27]2,21,2)
([28]1,4,0)([29]1,3,0)(11)
```

Príklad 5.1.1 ukazuje záznam piatich chromozómov v jednej generácii. Pre každý z nich sú uvedené nasledujúce informácie: prvé číslo v špicatých zátvorkách je fitness, tj. ohodnotenie, daného chromozómu, druhé číslo je počet použitých blokov. V ďalších špicatých zátvorkách sú údaje v tomto poradí: počet vstupov obvodu, počet výstupov obvodu, počet stĺpcov, počet riadkov, počet vstupov hradla, l-back parameter, počet použitých blokov. Ostatné prvky popisujú jednotlivé elementy — v hranatých zátvorkách je uvedené číslo výstupu tohto elementu, za ním sú uvedené čísla výstupov elementov, ktoré sú privedené na vstupy tohto elementu a posledné je číselné označenie funkcie elementu. Posledným prvkom je zoznam hradiel, ktoré sú pripojené k výstupom. Tieto hradlá sú od seba oddelené čiarkami.

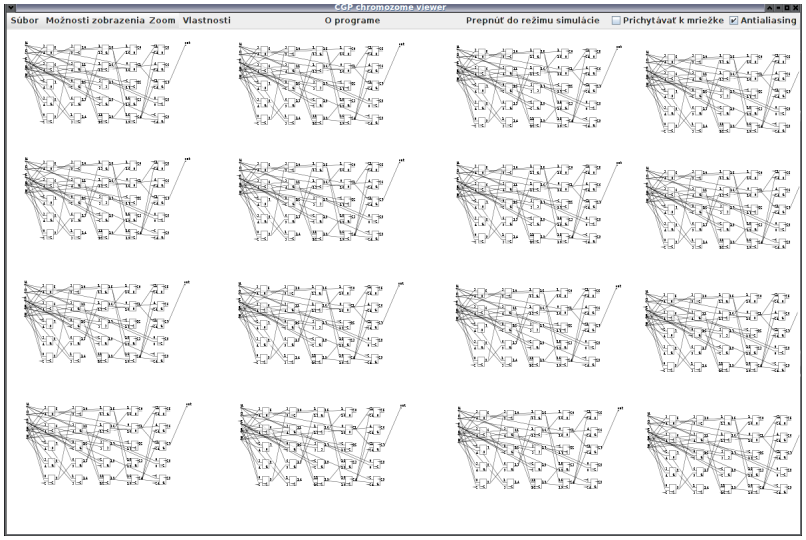
5.2 Návrh hlavného okna

Výsledný program má vedieť zobraziť celú evolúciu CGP, tj. históriu všetkých generácií a chromozómov. Celú históriu by bolo vhodné zobraziť v stĺpcoch, pričom jeden stĺpec by reprezentoval jednu generáciu. Aby takéto zobrazovanie bolo ešte rozumné, je vhodné obmedziť počet chromozómov v jednej generácii napríklad na päť. Každý evolučný beh sa skladá z množstva generácií, takže by malo byť možné presúvať sa medzi zobrazením rôznych generácií, viď obrázok 5.1. V tomto okne by bolo vhodné zobraziť aj fitness jednotlivých chromozómov, prípadne ďalšie dôležité informácie.

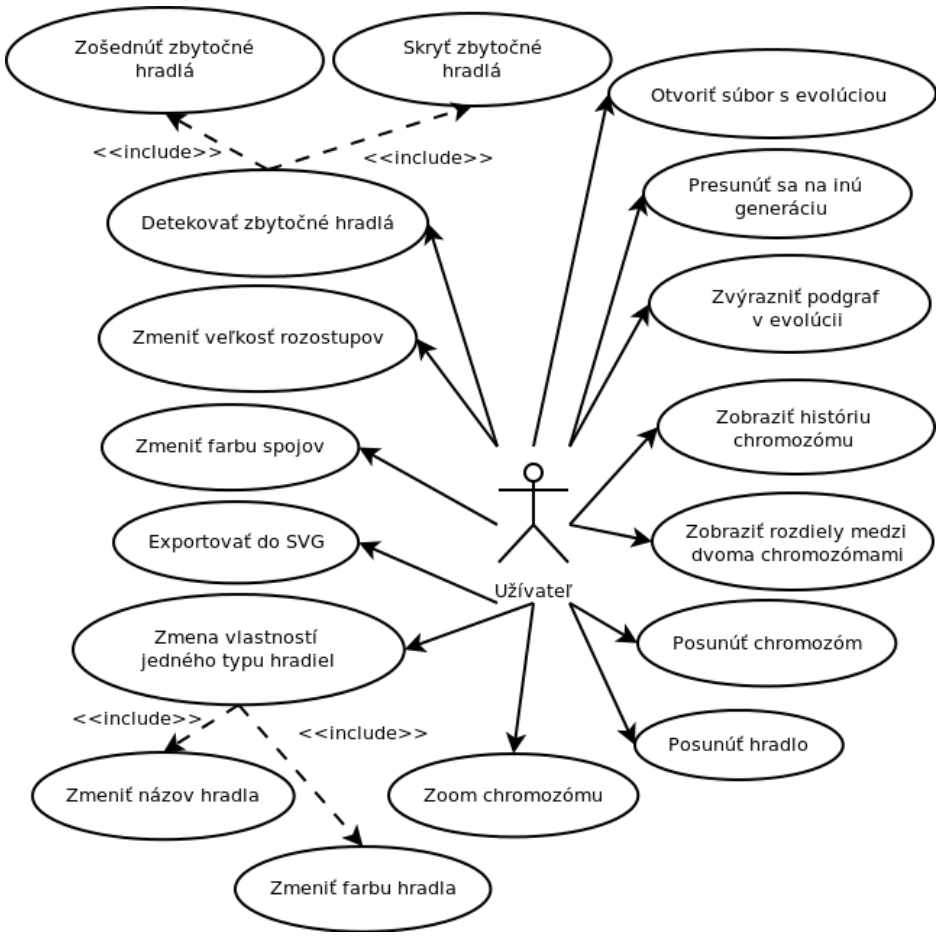
5.3 Základné funkcie

Základný pohľad na požadovanú funkčnosť poskytuje diagram prípadov užitia na obrázku 5.2.

V bakalárskej práci [10] bol vytvorený program pre vizualizáciu jedného chromozómu. Pretože vizualizácia jedného chromozómu bude základom programu pre vizualizáciu priebehu evolúcie, je možné prevziať radu už existujúcich funkcií do vytváraného programu. Všetky takto prevzaté funkcie bude potrebné upraviť pre nový dátový model a nové objekty, ktoré určite vzniknú. Určite to bude možnosť označiť nepotrebné hradlá a buď ich farebne označiť (zošednúť) alebo ich nechať skryť. Takisto pre lepšiu prehľadnosť ostane



Obrázek 5.1: Náčrt vzhľadu hlavného okna aplikácie. Jeden stĺpec reprezentuje jednu generáciu, v tomto prípade má generácia 4 jedincov.



Obrázek 5.2: Diagram prípadov užitia.

zachovaná možnosť posúvať jednotlivé hradlá aj celý chromozóm a možnosť približovať aj odďaľovať celú zobrazovaciu plochu. V možnostiach naďalej bude možnosť nastaviť farebnosť hradiel podľa ich funkcie a vlastná farba spojov, ktoré zadá užívateľ. Ďalej nebude chýbať možnosť zarovnať zobrazenie na mriežku, ale taktiež možnosť využiť prichytávanie hradiel pri presúvaní na zobrazenú mriežku. Akékoľvek úpravy chromozómu bude možné uložiť do vektorového formátu SVG, avšak pravdepodobne neostane možnosť uložiť si rozpracovaný chromozóm do interného formátu, pretože by to vyžadovalo nejaký spôsob, akým by sa takýto súbor dal otvoriť a takýto postup do celkovej koncepcie nezapadá. Naďalej však ostane možnosť jednoduchej simulácie obvodu.

S ohľadom na využitie hotovej bakalárskej práce[10] som sa rozhodla vytvoriť program pre vizualizáciu v jazyku Java. Je síce pomalší oproti kompilovaným jazykom (napr. C), ale oproti nim má veľkú výhodu v prenositeľnosti na iné operačné systémy. Taktiež sa v ňom výborne a rýchlo programuje, obzvlášť grafické rozhrania. V tomto jazyku existuje aj veľké množstvo knižníc, ktoré sa dajú použiť, napr. knižnica Batik SVG Toolkit[1], ktorú použijem pre export do formátu SVG.

5.4 Nové funkcie

Základom vizualizátoru evolúcie bude hlavné okno, ako bolo popísané vyššie. V tomto okne bude možné presúvať sa medzi generáciami buď pomocou posuvníka alebo metódou *drag'n'drop* ("chyť a posuň"), ktorú využíva posúvanie hradiel a chromozómu z bakalárskej práce. Tiež by malo byť možné označiť podgraf v chromozóme a sledovať jeho zmeny v priebehu evolúcie. Zobrazovanie je možné vyriešiť buď zvýraznením farbou alebo použitím popisujúcich textov. S ohľadom na bakalársku prácu je vhodnejšie použitie farieb, pretože to bude pôsobit prehľadnejšie.

Veľmi dôležitá je možnosť vybrať si jeden alebo dva chromozómy a zobraziť si ich detail. Tu príde na rad prepracovaný kód bakalárskej práce obohatený o ďalšie funkcie. V oboch prípadoch by mala pribudnúť možnosť zobrazenia histórie chromozómu, tj. označenie generácií, v ktorých nastala posledná zmena, ktorá viedla k súčasnému stavu. Toto je opäť možné riešiť ako farbami, tak aj textovo. V zobrazení chromozómu však už budú označené jednotlivé hradlá, vstupy aj výstupy textom, takže pre prehľadnosť bude určite lepšie použiť farby. Mali by vytvárať plynulý prechod z jednej farby do druhej, aby bolo možné ľahko posúdiť, kedy aká zmena nastala.

Pri zobrazení dvoch chromozómov by nemalo chýbať zobrazenie porovnania. Opäť bude určite praktické použiť farby na rozlíšenie rozdielov. Zafarbovať sa bude buď text alebo časti hradla samotné. Táto funkcia je vhodná pre sledovanie zmien v priebehu evolúcie.

5.5 Obmedzenia

Vzhľadom na použitie bakalárskej práce [10] existujú určité obmedzenia. Jedným z nich je obmedzenie na počet vstupov a výstupov hradiel. Hradlo môže mať maximálne dva vstupy a jeden výstup. S ohľadom na to, že Java je relatívne pomalá vo vykonávaní kódu a náročnejšia na výkon, bude určite existovať obmedzenie na počet populácií, prípadne na veľkosť súboru so záznamom behu CGP. Užívateľ je tiež obmedzený veľkosťou svojej obrazovky, čomu by mal prispôsobiť počet stĺpcov a riadkov zobrazovanej matice hradiel, alebo by mal akceptovať, že pri veľkom obvode zobrazovanie nebude ideálne.

Kapitola 6

Implementácia

Táto kapitola popisuje detaily implementácie v Jave. Pri implementácii som sa snažila využiť všetky výhody, ktoré Java poskytuje, napríklad objektovo orientovaný prístup, grafické rozhranie Swing, jednoduché používanie externých knižníc a podobne.

Vzhľadom na to, že táto práca naväzuje na výsledky bakalárskej práce, sú tu popísané aj tieto. Konkrétne algoritmy som priamo neupravovala, avšak aby bol program bakalárskej práce použiteľný pre potreby tejto práce, bolo potrebné na ňom urobiť dosť veľa menších zmien a úprav.

6.1 Úprava programu cgp

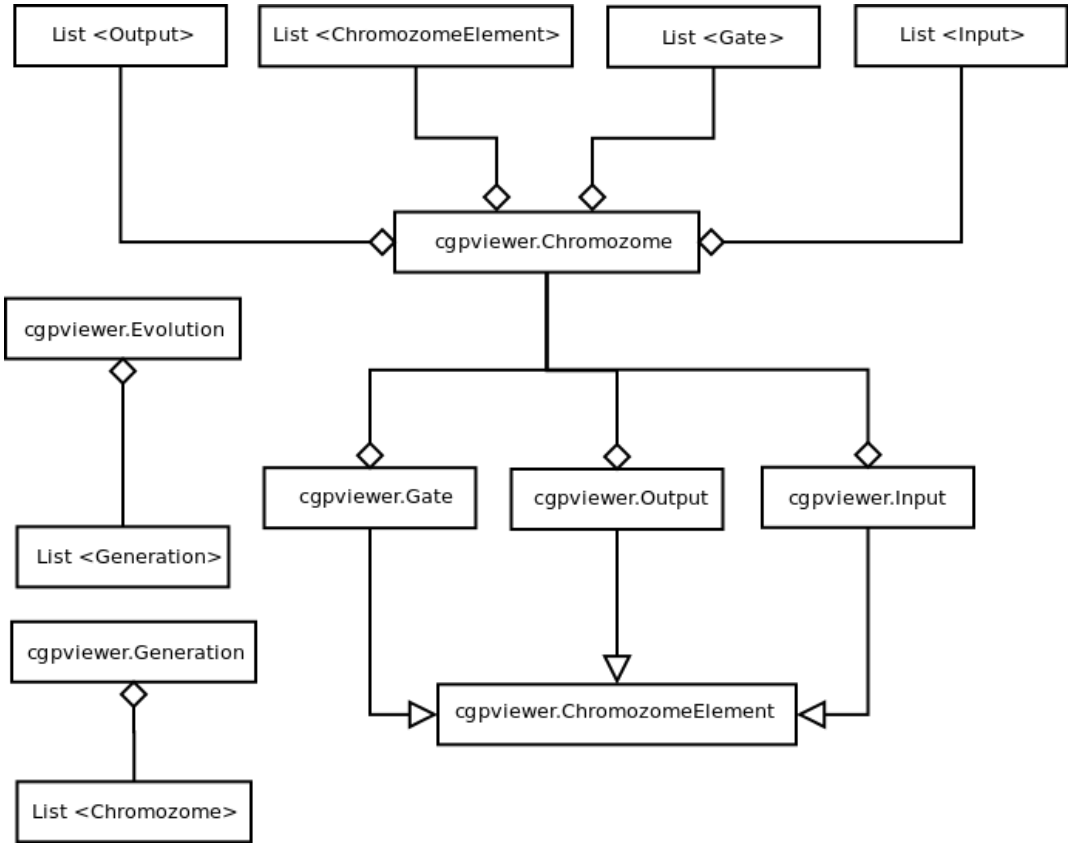
Najprv je potrebné získať dáta, ktoré sa budú zobrazovať. Pre tento účel bol upravený program cgp [12]. Upravený program je možné nájsť v prílohe, avšak uvediem tu aj postup, akým som k nemu dospela. Postačí zakomentovať štyri riadky kódu, aby sa vypisovali všetky chromozómy každej generácie. Konkrétne riadky sú vypísané v algoritme 6.1.1, v kóde programu sú to riadky 377, 378, 384 a 385. Samozrejme je možné vypisovať každú desiatu generáciu a podobne, podstatné je, aby v každej generácii boli nejaké chromozómy a aby boli v tom správnom tvare.

Algoritmus 6.1.1.

```
#ifdef PERIODIC_LOG
   if (param_generaci % PERIODICLOGG == 0) {
       ...
   }
#endif
```

6.2 Hlavné triedy aplikácie

Základom tohto programu je trieda `Evolution`. Je to singleton, čo znamená, že jediná inštancia tohto objektu existuje vrámci programu iba jeden krát a je dostupná zo všetkých ostatných objektov. Táto vlastnosť je dôležitá pre používanie rôznych premenných, ktoré majú byť v celom programe jednotné. Takisto je vďaka tejto vlastnosti jednoduchý prístup ku všetkým generáciám a ich chromozómom, čo je využité pri zobrazovaní histórie v zobrazení jedného alebo dvoch chromozómov. Usporiadanie tried je prehľadne zobrazené na obrázku 6.1, podrobnejší popis je ďalej v tejto sekcii.

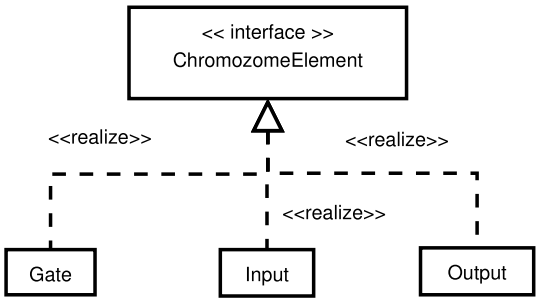


Obrázek 6.1: Diagram tried pre triedy obsahujúce dáta.

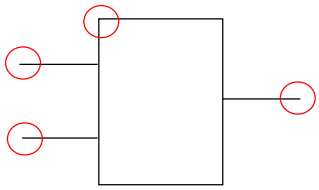
Táto trieda je postavená hlavne na zozname generácií, pričom každá položka, tj. generácia, je objektom, ktorý je tvorený zoznamom jedincov, tj. chromozómov. Na reprezentáciu chromozómu som zvolila triedu z [10], ktorú som upravila zo singletonu na obyklú, viackrát inštanciovateľnú triedu. V nej je opäť zoznam objektov, tentokrát tvorený hradlami, vstupmi a výstupmi, pričom každý z nich má vlastnú triedu. Vzhľadom na to, že jednotlivé elementy (vstupy, výstupy a hradlá) majú určité vlastnosti spoločné a niektoré zasa rozdielne, bolo potrebné vytvoriť abstraktné rozhranie, na obrázku 6.2, ktoré tieto elementy potom implementujú. Vďaka tomuto postupu sú uľahčené niektoré programové konštrukcie a práca s týmito elementami, napríklad vykresľovanie. Aby však bolo možné využiť aj vlastnosti, v ktorých sa elementy líšia, sú vytvorené aj samostatné zoznamy objektov pre rôzne druhy elementov. Vo všetkých zoznamoch sú len odkazy na objekty, takže je možné mať jeden objekt vstupu a pristupovať k nemu cez rôzne zoznamy rôznym spôsobom.

Každé hradlo obsahuje informácie o svojej funkcii, vstupoch, výstupe, rovnako ako súradnice ľavého horného rohu, koncových bodov vstupov a výstupu, viď obrázok 6.3. Súradnice sa používajú pri vykresľovaní, detekcii miesta kliknutia užívateľa a adekvátne sa prepočítavajú. U vstupov a výstupov je to podobne, taktiež majú svoje poradové číslo, súradnice a podobne.

Objekt evolúcie sa vytvára po načítaní záznamu behu CGP, ktorý má tvar popísaný v predchádzajúcej kapitole. Veľkosť tohoto súboru je obmedzená na 10MB kvôli zisteným problémom so stabilitou pri väčších súboroch.



Obrázek 6.2: Diagram tried pre triedy vstupu obvodu, hradla a výstupu obvodu.



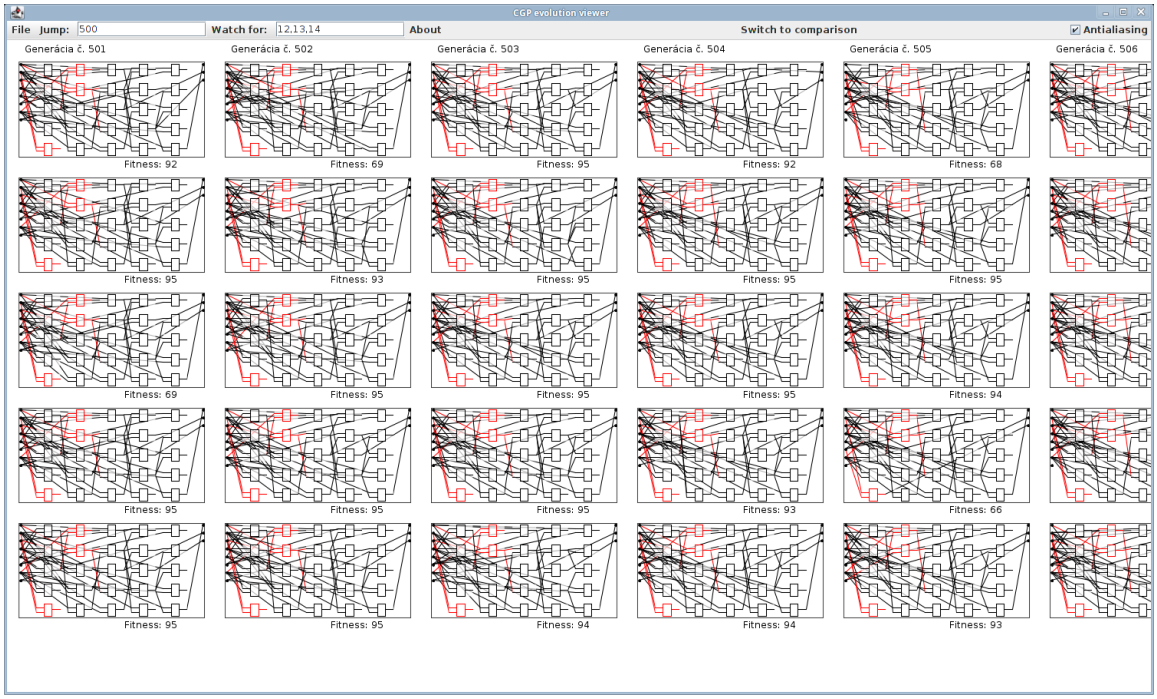
Obrázek 6.3: Vykreslené hradlo. Zakrúžkované sú ukladané súradnice.

6.3 Vykresľovanie

Po načítaní súboru s behom CGP sa vypočítajú súradnice pre všetky chromozómy v tejto populácii a zobrazia sa. Výpočet ich umiestni tak, aby chromozómy v jednej generácii boli vždy pod sebou v jednom stĺpci, pričom každý stĺpec ma poradové číslo, ktoré určuje číslo generácie. Pri vykresľovaní je dbané na rovnaké rozostupy medzi chromozómami. Zároveň s chromozómami sa ku každému vypíše aj jeho fitness hodnota a pre pochopiteľnejšie grafické rozhranie je každý chromozóm ohraničený obdĺžnikom, viď obrázok 6.4. Ohraničenie uľahčuje orientáciu a zdôrazňuje, kam má užívateľ kliknúť, ak si chce zobraziť detail chromozómu, prípadne porovnanie dvoch chromozómov, a kam má kliknúť, ak sa chce presunúť na inú generáciu. Obdĺžnik ohraničuje plochu, ktorá reaguje na kliknutie tým, že zobrazí nové okno s detailami o jednom alebo dvoch chromozómoch. Detekované kliknutie mimo tieto obdĺžniky spôsobí "chytenie"pracovnej plochy a jej posunutie o posun myši užívateľom. Reakcia na kliknutie na chromozóm je určená prepínačom. Možnosti manipulácie s vybraným chromozómom sú u oboch možností rovnaké, avšak v porovnávacom režime, keď sa zobrazia dva chromozómy, je možné zapnúť porovnanie týchto dvoch chromozómov.

Ďalšia funkcia v tomto okne je možnosť vypnúť a zapnúť antialiasing. Ten spôsobuje vyhladenie hrán ak je zapnutý, takže obraz vyzerá krajšie, avšak je náročnejší na zobrazenie, preto je posúvanie sa v evolúcii relatívne pomalé. Aby si užívateľ dokázal rýchlejšie zobraziť ďalšie generácie, je k dispozícii funkcia s názvom **Jump**, ktorá preskočí toľko generácii, koľko užívateľ do prázdneho poľa zadá. Týmto spôsobom je veľmi jednoduché presúvať sa dopredu aj dozadu v priebehu evolúcie. V programe je to urobené tak, že sa k súradniciam každého chromozómu pripočíta adekvátny počet pixelov, ktorý môže byť samozrejme aj záporný. Užívateľom zadané číslo sa vynásobí celkovou veľkosťou jedného chromozómu v pixeloch a o toto číslo sa posunú všetky potrebné súradnice.

Pre sledovanie zmien v podgrafoch chromozómu je dostupná možnosť zadať do poľa **Watch for** poradové čísla hradiel oddelené čiarkou, ktoré sa majú zvýrazniť v zobrazení



Obrázek 6.4: Okno zobrazujúce šesť generácií jedného behu CGP. Červenou sú zvýraznené dôležité hradlá spolu so spojmi vedúcimi do nich a z nich. Ako dôležité ich označuje užívateľ.

celej evolúcie. Spolu so zadanými hradlami sa zvýraznia aj všetky spojenia, ktoré vedú do a z týchto hradiel, ukážka na obrázku 6.4. Pokiaľ užívateľ nepozná presné čísla hradiel, ktoré by ho zaujímali, môže si ich zistiť tak, že si nechá zobraziť detail chromozómu, kde sú tieto čísla zobrazené. Takto môže vizuálne sledovať zmeny a ak zaznamená niečo, čo chce podrobnejšie preskúmať, zobrazí si detail konkrétneho chromozómu. Pre prehľadnosť v tomto zobrazení nie je zvýraznených viac zmien, prípadne iných detailov. Z pokusov totiž vyplýva, že každý chromozóm je tvorený veľkým množstvom čiar, ktoré sa prekrývajú. Vypisovanie rôznych čísel, prípadne textov, do chromozómu v zobrazení evolúcie by spôsobilo nečitateľnosť a ešte viac neprehľadných čiar. Na vyznačovanie zmien sú teda vhodné farby, avšak ani s nimi to nie je správne preháňať, pretože to opäť spôsobí neprehľadnosť.

Základné okno slúži len na prehľadné zobrazenie evolúcie. Pre ďalšiu manipuláciu s chromozómami je nutné si jeden vybrať a zobraziť si jeho detail, prípadne si zobraziť porovnanie dvoch chromozómov. V ďalších častiach budú popísane funkcie a možnosti týchto volieb.

6.3.1 Zošednutie a skrytie hradiel

Pre tieto funkcie je potrebné detekovať, ktoré hradlá sa podieľajú na celkovom výsledku, pretože zošednú, prípadne skryjú sa, tie hradlá, ktoré nemajú žiaden vplyv na hodnotu na výstupe. Táto detekcia sa deje tak, že všetkým hradlám sa najprv nastaví viditeľnosť na `GREY` (hodnota vlastného vymenovaného typu viditeľnosti hradiel) a prechádza sa obvod od výstupov k vstupom, pričom sa použité hradlá menia na `VISIBLE`. Pri prechádzaní sa použije rekurzia, pretože metóda na detekciu využíva to, že čísla vstupov označujú čísla výstupov iných hradiel, ktoré do nich vedú. U výstupov nastáva špeciálny prípad, pretože do výstupov celého obvodu vedie vždy len jeden vstup. Tieto teda majú vlastnú metódu, ktorá ale nakoniec robí to isté, ale iba pre jeden vstup. V algoritme 6.3.1 je metóda pre

prechádzanie výstupov chromozómu. Ako je vidieť, vždy sa zavolá metóda z algoritmu 6.3.2, ktorá vykonáva rekurziu.

Algoritmus 6.3.1.

```
public void findUnnecessary() {
        // skryť všetky hradlá
        for (int i = 0; i < listOfGates.size(); i++) {
                listOfGates.get(i).visible = GREY;
        }
        // skontrolovať každý výstup
        for (int i = 0; i < number_of_out; i++) {
                // volanie rekurzívnej metódy
                int number = listOfOuts.get(i).number;
                listOfGates.get(number).visible = VISIBLE;
                setVisible(listOfGates.get(number).in1,
                 listOfGates.get(number).in2);
        }
}
```

Algoritmus 6.3.2.

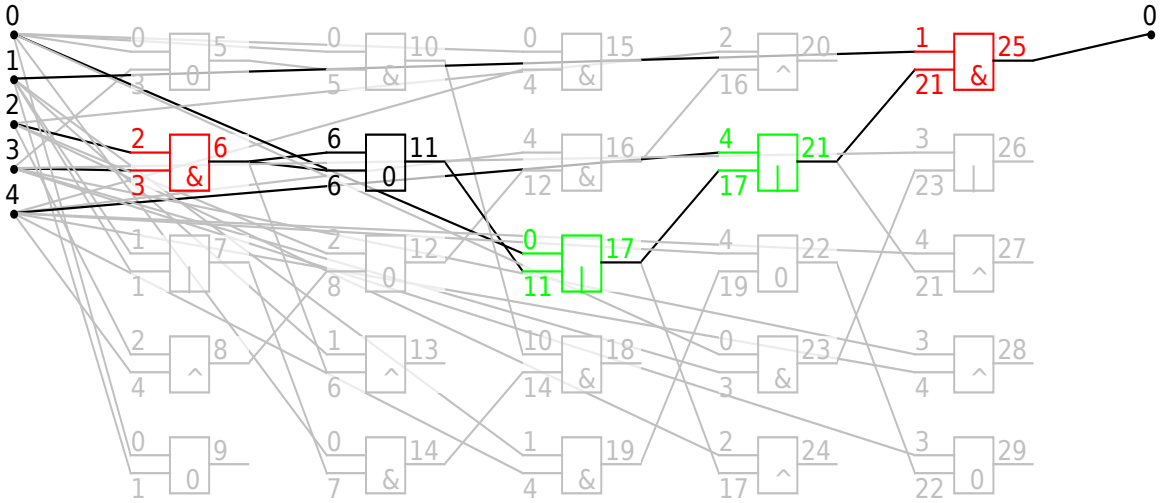
```
private void setVisible(int x, int y) {
        if (x >= number_of_in) {
                listOfGates.get(x).visible = VISIBLE;
                setVisible(listOfGates.get(x).in1,
                 listOfGates.get(x).in2);
        }
        if (y >= number_of_in) {
                listOfGates.get(y).visible = VISIBLE;
                setVisible(listOfGates.get(y).in1,
                 listOfGates.get(y).in2);
        }
}
```

Po spustení týchto algoritmov môže dôjsť k samotnému zošednutiu alebo skrytiu nepotrebných hradiel. Pre zošednutie hradiel postačí nastavenie viditeľnosti na `GREY` a ošetrenie pri vykresľovaní, aby bola použitá správna farba – svetlošedá ako na obrázku 6.5. Pri skrytí nepotrebných hradiel je však ešte potrebné počítať s tým, že tie hradlá by nemali umožňovať žiadnu interakciu, čo znamená že nemajú reagovať na kliknutie vo svojej oblasti. Prepínač viditeľnosti sa nastaví na `INVISIBLE` a pri vykresľovaní aj detekovaní sa používa ako kontrolná podmienka. Vďaka tomu funguje vykreslenie aj detekcia správne, ukážka na obrázku 6.6 a po zarovnaní na obrázku 6.7. Obe tieto funkcie fungujú rovnako v oboch režimoch zobrazenia – detail aj porovnanie.

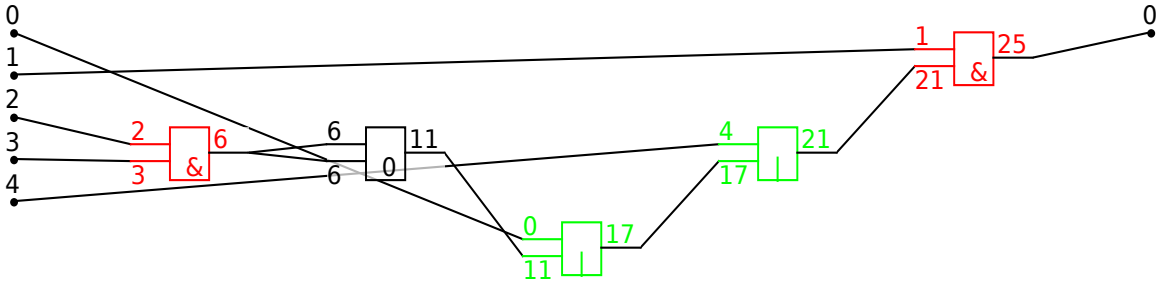
V prípade viditeľných hradiel je farba určená ich funkciou, viď obrázok 6.8. Pri spustení programu má každá funkcia nastavenú čiernu farbu, ktorá sa dá jednoducho zmeniť v nastaveniach alebo nahraním súboru s nastaveniami.

6.3.2 Posúvanie prvkov a kresliacej plochy

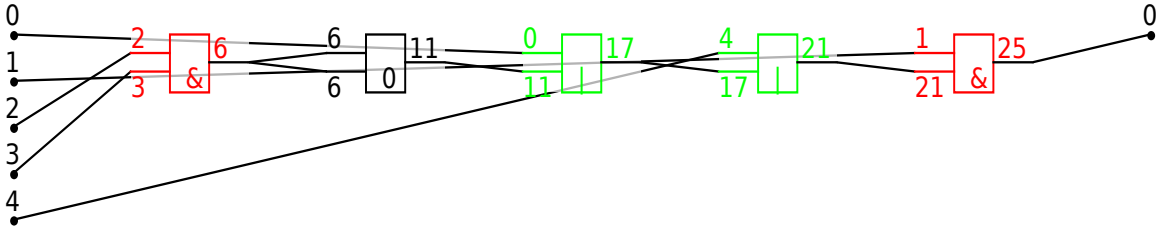
Všetko vykresľovanie sa deje v objekte `canvas` (v Jave je to trieda `java.awt.Canvas`), čo v preklade znamená kresliace plátno. Tento objekt obsahuje metódu `paint()`, ktorá sa za-



Obrázek 6.5: Zobrazenie chromozómu po zošednutí zbytočných hradiel.



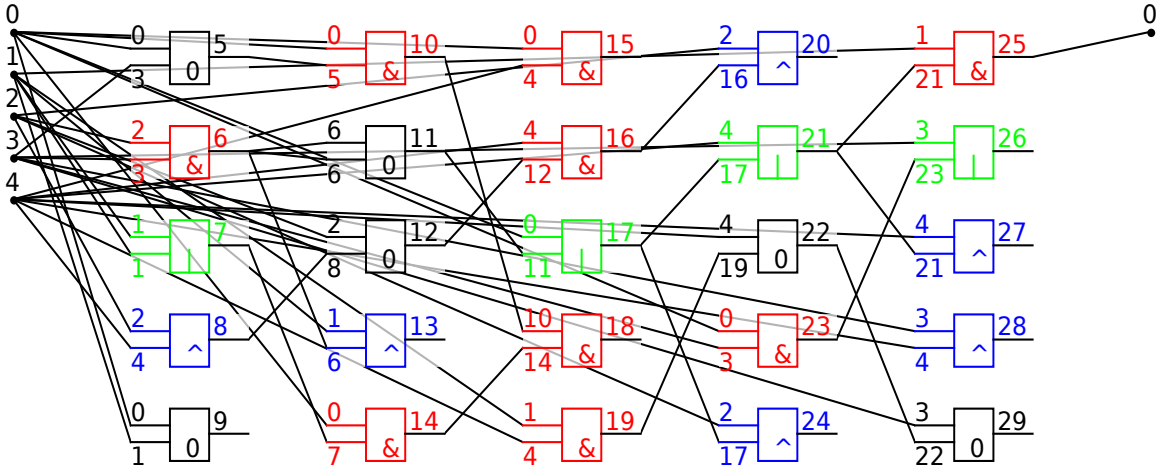
Obrázek 6.6: Zobrazenie chromozómu so skrytými hradlami.



Obrázek 6.7: Zobrazenie chromozómu bez zbytočných hradiel zarovnané na mriežku.

volá automaticky pri vykreslení, takže jej telom je postup toho, čo má byť vykreslené. Pre umožnenie detekcie kliknutia a tiahnutia je potrebné v tejto triede implementovať rozhrania `MouseListener` a `MouseMotionListener`. Tieto rozhrania obsahujú metódy, ktoré sa volajú pri aktivite myši – stlačenie, uvoľnenie stlačenia, ťahanie a podobne. Práve tieto tri pomenované metódy sú využívané v tomto programe. Najprv je detekované, že užívateľ niekam klikol. V závislosti na tom, kam klikol, sa do premenných uložia potrebné údaje. Potom sa uskutoční požadovaný posun hradlom alebo celým plátnom. Rozdiel medzi súradnicami stlačenia a uvoľnenia myši tvorí veľkosť posunu.

Pri posúvaní hradiel a celého plátna sa využíva metóda `MouseDragged()`, ktorá spôsobuje, že posúvaný prvok sa stále vykresľuje a tak je vždy vidno jeho aktuálnu polohu, čo napomáha lepšej orientácii. Bohužiaľ to má nepríjemný vedľajší efekt — obraz "bliká". Neustálym prekresľovaním sa vždy vymaže aktuálny obraz, vypočíta nový a potom sa vykreslí,



Obrázek 6.8: Zobrazenie chromozómu hneď po otvorení a zafarbení.

takže vo fáze počítania je obraz prázdny. Riešením je použitie dvoch zásobníkov (tzv. double buffering). V jednom momente sa zobrazuje iba jeden, do druhého sa zatiaľ počítajú nové hodnoty. Potom sa tieto dva zásobníky vzájomne vymenia, takže predpočítaný sa zobrazí. Pre urýchlenie vykresľovania sa pri posúvaní hradla vypne antialiasing, ktorý je výpočetne náročnejší a zapne sa až po ukončení posúvania, čo je dôležité pre plynulé posúvanie.

S posúvaním hradiel pri zobrazení detailu a porovnania chromozómov súvisí aj prichytávanie k mriežke. Po zaškrtnutí tejto možnosti v aplikácii sa pod chromozómom vykreslí svetlošedá mriežka, ku ktorej sa budú prichytávať hradlá. Samotné prichytávanie je založené na zaokrúhľovaní súradníc k súradniciam mriežky. Ak by zaokrúhlenie prebiehalo vždy pri spustení detekovacích metód, tak by zmena polohy bola vždy príliš malá a zaokrúhľovalo by sa vždy na to isté číslo, takže hradlo by sa nikam neposunulo. Preto v programe existujú dvoje súradnice, jedny zaokrúhlené, druhé nie. Nezaokrúhlené súradnice sa používajú pri presúvaní hradla a zaokrúhľujú sa až pri vykreslení a výpočte ostatných súradníc hradla.

Pri posune celého kresliaceho plátna som sa rozhodla využiť princíp známy hlavne z dotykových obrazoviek – "chytím plochu a posuniem"– v tomto prípade s použitím myši. Tento postup vyzerá efektne a navyše aj sprehľadňuje kód tohto programu. Aj v tomto prípade však je potrebné brať do úvahy použitie prichytávania na mriežku. Aby pri takomto posune celý chromozóm zvláštne "neposkakoval"po obrazovke, posúva sa aj celá mriežka. Takýto pohyb je prirodzenejší a plynulejší.

V prípade zobrazenia celého priebehu evolúcie je detekcia kliknutí podobná, avšak trochu obmedzenejšia. Pri tomto zobrazení totiž nie je možné posúvať jednotlivé chromozómy, vždy je možné iba posunúť celé plátno alebo si vybrať chromozóm pre zobrazenie detailu alebo porovnania. Ako bolo spomenuté vyššie, každý chromozóm je vizuálne ohraničený obdĺžnikom, aby bolo zjavné, ktorá oblasť plátna zareaguje akým spôsobom a nedochádzalo k nedorozumeniam. V tomto zobrazení nedáva zmysel prichytávanie k mriežke, čo trochu zjednodušilo výpočet súradníc pri posune celého plátna. Pri veľkom počte generácií je zobrazovanie dosť náročné, takže som ho optimalizovala tak, aby sa vykresľovali iba tie chromozómy, ktoré sú na ploche viditeľné. Každý chromozóm teda obsahuje podmienku $minX < width$ a zároveň $maxX > 0$, čo znamená, že jeho minimálna súradnica na ose x musí byť menšia ako šírka kresliaceho plátna a maximálna súradnica na osi x musí byť väčšia ako nula. Osa x má svoj počiatok v ľavom hornom rohu kresliaceho plátna a jej maximálna súradnica má hodnotu šírky plátna. Posúvanie plátnom v tomto prípade už nie

je tak plynulé, avšak malo by to zrýchliť prácu s programom.

6.3.3 Zrušenie posuvov a zmien

Jednou možnosťou ako rušiť posuvy a zmeny je zobrazenie hradiel na mriežku, čo znamená, že všetky momentálne viditeľné hradlá dostanú pokyn na prepočítanie svojich súradníc a ich upravenie na mriežku. Táto mriežka však nesúvisí priamo s mriežkou z prechádzajúcej časti, pretože v tomto prípade sa uskutoční rovnaký výpočet ako pri vytvorení okna s detailom chromozómu, avšak s tým rozdielom, že bude brať do úvahy, že niektoré hradlá sa nemajú zobraziť, takže ich ani nezahrnie do výpočtu. Toto je vhodné pre prehľadnosť a rýchle minimalistické usporiadanie hradiel.

Druhou možnosťou je zrušiť všetky urobené zmeny ohľadom zobrazovania hradiel a vyresetovať celé zobrazenie. Najprv sa všetkým hradlám nastaví viditeľnosť na `VISIBLE`, a potom si všetky hradlá prepočítajú súradnice a chromozóm sa zobrazí v strede kresliaceho plátna.

6.3.4 Zoom

Objekt `Graphics` poskytuje možnosť využiť transformačnú maticu, ktorá manipuluje s niektorými vlastnosťami obrazu, z ktorých je pre tento program podstatný práve zoom. Táto matica je použiteľná vo všetkých triedach odvodených od `Graphics`, ako je `canvas` a export do SVG. Jej použitie je veľmi jednoduché, postačí na to riadok kódu.

```
g.setTransform(new AffineTransform(zoom, 0, 0, zoom, 0, 0));
```

Pri vytváraní transformačnej matice je dôležitá premenná zoom, ktorá sa mení pri kliknutí užívateľa na príslušné tlačítka, prípadne stlačením klávesovej skratky. Mení sa však aj pri centrovaní chromozómu do stredu plátna tak, aby chromozóm vypĺňal čo najväčšiu plochu a zároveň nikde neprechádzal za hranice okna. Tohto je dosiahnuté tým, že sa vypočíta pomer šírky chromozómu k šírke plátna a pomer výšky chromozómu k výške plátna. Najmenšie z týchto čísel sa použije.

6.3.5 Nastavovanie vlastností zobrazovania

Všetky zafarbiteľné prvky sa pri spustení programu nastavia na čiernu farbu. Pre prehľadnosť používania má užívateľ možnosť nastaviť väčšine prvkov vlastnú farbu. K tomuto slúži položka **Properties** v menu, ktorá otvorí okno s nastaveniami, viď obrázok 6.9. Pri kliknutí na tlačidlo **Change color** sa zobrazí vzorkovník s farbami a užívateľ si môže vybrať želanú farbu. Vzorkovník poskytuje Java v triede `javax.swing.JColorChooser`. Pomocou jej metódy `showDialog()` si užívateľ vyberie farbu s ktorou ďalej pracuje tento program. Uloženie farby je automatické po potvrdení jej výberu v dialógu.

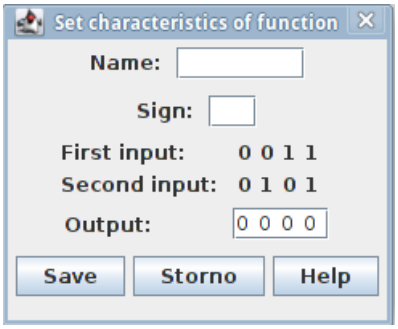
Je možné nastaviť aj šírku a výšku rozostupov medzi elementami v chromozómoch. Na prejavenie tejto zmeny je však potrebné spustiť prepočítanie súradníc kliknutím na vyresetovanie zobrazenia, pretože treba uvažovať prípad, keď má užívateľ rozpracovaný nejaký chromozóm a popresúval nejaké hradlá. V tomto prípade je lepšie, keď má užívateľ túto zmenu vo svojich rukách.

Obrázek 6.9: Okno nastaviteľných vlastností.

6.3.6 Nastavenie vlastností simulátoru obvodu

Program umožňuje simulovať kombinačné obvody. K tomu je potrebné nastaviť množinu funkcií, ktoré sú podporované v CGP. Tieto funkcie je možné nastaviť pomocou tlačidla **Set function**, viď obrázok 6.10. V tomto novom okne je možné zmeniť názov funkcie, jej symbolický znak, ktorý sa zobrazuje pri vykreslení chromozómu a pravdivostnú tabuľku, ktorú používa simulácia. Detailný popis simulácie bude uvedený neskôr.

Vlastnosti funkcií je možné nastaviť aj pomocou načítania vhodného súboru. Tento súbor má mať riadky v tvare `<poradové číslo> <názov> <znak> <výsledok> <farba>`. Jednotlivé položky musia byť oddelené medzerou a nesmú tam byť žiadne znaky navyše. Poradové číslo musí byť z intervalu 0-15, spojí totiž hodnoty zo vstupného súboru evolúcie s ďalšími hodnotami na riadku. V CGP má každá funkcia svoje číselné označenie a toto musí byť použité v súbore. Názov funkcie je použitý iba pre prehľadnosť pri zobrazovaní vlastností. Znak by mal byť jeden a použije sa pri vykreslení chromozómu. Výsledok je postupnosť štyroch 1 a 0, ktoré sa inak zapisujú do posledného riadku na obrázku 6.10. Posledným prvkom je farba v šestnástkovej (hexadecimálnej) sústave. Týmto súborom môže užívateľ hromadne nastaviť zobrazovacie vlastnosti funkcií a taktiež výsledky simulácie pre



Obrázek 6.10: Okno nastaviteľných vlastností funkcie jedného dvojvstupového hradla.

všetky typy hradiel. Súbor môže byť ľubovoľne dlhý, ale použije sa vždy posledný výskyt riadku s konkrétnym poradovým číslom. Taktiež v súbore nemusia byť uvedené všetky typy hradiel.

6.4 Export do SVG formátu

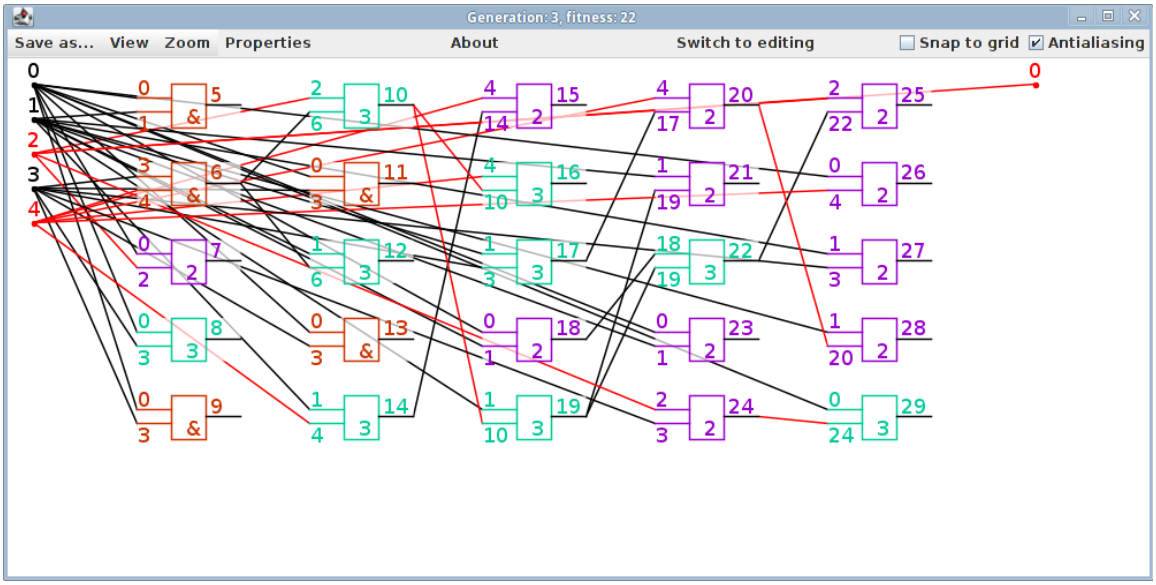
Na export do SVG som využila externú knižnicu Batik SVG Toolkit [1], ktorá slúži na manipuláciu s obrázkami SVG. V tejto práci postačuje možnosť kresliť do dokumentu so štruktúrou DOM (Document Object Model) rovnakým spôsobom, akým prebieha kreslenie na plátno pomocou objektu `Graphics`. Túto možnosť som využila v programe, avšak bolo by možné už pri spustení vytvoriť obrázok vo formáte SVG a ten zobraziť na plátne a ďalej s ním pracovať. Tento prístup je však zbytočný, pretože nie je predpoklad, že by si užívateľ ukladal chromozómy do SVG tak často, ako ich bude zobrazovať.

6.5 Simulácia

Pri zapnutej simulácii kandidátneho obvodu (tlačidlo **Switch to simulation**) nie je možné posúvať hradlami ani celou plochou, ostatné možnosti však stále fungujú (zoom, zošednutie, vymazanie zbytočných hradiel a podobne). Nastavenie farieb funkcií sa použije, avšak farba spojov (v nastaveniach položka **Lines**) sa nastaví na čiernu, ak sa tam nachádza logická nula a na červenú, ak sa tam nachádza logická jedna. Ukážka je na obrázku 6.11. Rovnaké označenie farbami sa použije tiež na vstupy a výstupy.

Po spustení simulácie sa prechádza zoznam všetkých elementov, u ktorých sa zavolá metóda pre simuláciu a každý element túto simuláciu uskutoční, tj vypočíta hodnotu, ktorá bude na výstupe. Túto hodnotu potom získa hradlo, ktoré je pripojené na tento výstup. Takýmto spôsobom sa prejde celý obvod.

Výpočet simulácie jednotlivých hradiel je uskutočnený pomocou bitových operácií. Pre výpočet je potrebné vedieť, aká logická hodnota je na ktorej pozícii vstupu. Tá sa zisťuje pomocou jednotky posunutej o n bitov. Zisťuje sa to pomocou algoritmu 6.5.1. Premenná $in1val$ je hodnota na prvom vstupe, analogicky k nej je $in2val$ hodnota na druhom vstupe. Posúvať bude potrebné o 0, 1, 2 a 3 bity, pretože máme 4 možné kombinácie hodnôt pri dvojvstupových hradlách. K tomu nám pomôže negácia ($\sim$). Tabuľka vstupov 6.1 totiž po negácii dáva správne výsledky, ak negované hodnoty sčítame, viď tabuľka 6.2. Ak by sme však na negované hodnoty nepoužili masku pomocou AND (&), z čísla 1 by vzniklo číslo, ktoré by v binárnej sústave malo na začiatku samé jednotky a posledná by bola nula, čo



Obrázek 6.11: Simulácia. Červenou sú spoje, na ktorých je logická hodnota jedna. Čierne obsahujú nulu.

by k výsledku nijak nepomohlo. Po použití masky ostane iba posledná číslica, tj. tá ktorá je potrebná pre výsledok.

Algoritmus 6.5.1.

```
n = (((~in1val) & 1) << 1) + ((~in2val) & 1);
```

prvý vstup	0	0	1	1
druhý vstup	0	1	0	1

Tabulka 6.1: Tabuľka vstupov.

`((~in1val) & 1) << 1`	10	10	00	00
`(~in2val) & 1`	01	00	01	00
súčet predchádzajúcich dvoch riadkov	11	10	01	00

Tabulka 6.2: Tabuľka vstupov po negácii a bitovom posune a tretí riadok je výsledok po sčítaní (pre krajší vzhľad sú nuly doplnené v druhom riadku).

V tejto chvíli už existuje hodnota, ktorá je potrebná pre simuláciu. Simulácia hradla je popísaná algoritmom 6.5.2. Premenná *value* je výsledná hodnota hradla, *func* je binárna reprezentácia funkcie hradla. Takúto binárnu reprezentáciu vidíme na treťom riadku v tabuľke 6.3, tj. je to binárne číslo 0001, ktoré sa v premennej uchováva v podobe integeru (celého čísla), takže sa na ňom bez veľkých problémov dajú použiť bitové operácie. Najprv sa teda číslo jedna posunie o predtým vypočítaný počet bitov n, čím sa určí poloha hľadaného výsledku v binárnej reprezentácii funkcie hradla. Potom sa táto maska použije pomocou operácie AND na binárnu reprezentáciu funkcie hradla a v prípade, že výsledkom

tejto operácie bude nula, je aj celkovým výsledkom nula. Ak je to akékoľvek iné číslo, výslednou hodnotou bude jedna.

Algoritmus 6.5.2.
`value = (func & (1 << n)) == 0 ? 0 : 1;`

Na tomto mieste by som chcela upozorniť na operáciu NOT. Tá totiž využíva iba jeden vstup. V stave v akom je definovaná v programe, bude fungovať pre hodnoty prvého vstupu, tj. pre poradie hodnôt prvého vstupu, ktoré je uvedené v tabuľkách vyššie, budú jeho výsledky 1100. Na druhom vstupe teda nezáleží.

prvý vstup	0	0	1	1
druhý vstup	0	1	0	1
výstup	0	0	0	1

Tabulka 6.3: Prvé dva riadky sú vstupy, posledný riadok sú výstupy po operácii AND.

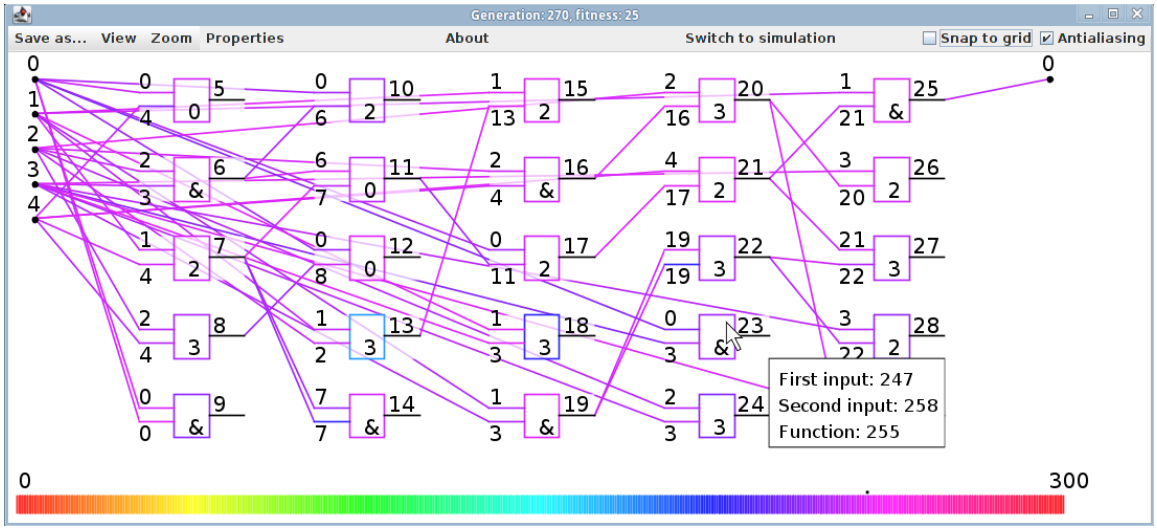
Popis tohto algoritmu bol prevzatý z [10]. Tento algoritmus nevyžadoval takmer žiadne zmeny. Jediná zmena je v zafarbovaní samotných hradiel, už sa nezafarbujú podľa hodnôt funkcie, ale využívajú nastavené farby funkcií od užívateľa. Tento algoritmus funguje ako pre zobrazenie detailu jedného chromozómu, tak pre zobrazenie dvoch chromozómov.

6.6 Zobrazenie histórie

Pri zobrazovaní histórie evolúcie chromozómu som opäť použila radšej farebné označenie než textový výpis. Na vyfarbenie používam tisíc farieb s plynulým prechodom. Najprv sa vytvorí pole farieb a pri vykresľovaní každé hradlo zisťuje, ktorou farbou má ktorú časť vykresliť (vstup, výstup, funkciu). Keďže generácií môže byť viac ako tisíc, je potrebné natiahnuť paletu farieb, respektíve upraviť číslo generácie tak, aby sa nachádzalo v rozsahu poľa. Opačné prispôsobenie je potrebné ak je generácií menej než farieb. Na obe možnosti však postačuje jediný vzorec. Počet farieb sa vydelí celkovým počtom generácií a výsledok sa vynásobí generáciou, v ktorej nastala posledná zmena. Táto generácia sa zisťuje pomocou cyklu, v ktorom sa prejde celý záznam evolúcie až po generáciu vybraného chromozómu, a každý chromozóm sa porovná s aktuálnym. Ak nastala zmena, tak sa číslo generácie uloží do premennej v hradle, ktorého sa táto zmena týkala. Zmeny môžu nastať iba vo vstupoch (výstup ktorého hradla je pripojený na vstup) a vo funkcii hradla. Číslo výstupu sa nemení, je to jeho poradové číslo v obvode, aby bolo možné sa naňho presne odkazovať. Po tomto výpočte je už možné zobraziť chromozóm s označenou históriou. Toto zobrazenie funguje pre zobrazenie detailu jedného aj dvoch chromozómov. V oboch prípadoch sa v spodnej časti vykreslí paleta farieb, aby užívateľ vedel približne odhadnúť, kde zmena nastala, ukážka na obrázku 6.12. Z obrázku je poznať, že takmer všetky zmeny nastali nedávno. Je to spôsobené tým, že aktuálny chromozóm sa porovnáva so všetkými chromozómami v každej generácii, nie iba s tými, z ktorých bol vytvorený mutáciou. Bohužiaľ nie je možné sledovať presné zmeny jedného konkrétneho chromozómu, od prvého jedinca až po aktuálneho, pretože takúto informáciu záznam evolúcie neobsahuje.

Pre lepšiu orientáciu je možné posunúť kurzor myši nad hradlo, ktoré je zaujímavé a od jeho pravého dolného rohu sa vykreslí obdĺžnik s podrobnými informáciami o histórii hradla, opäť ukážka na obrázku 6.12. Nie je potrebné na toto hradlo klikať, pretože kliknutie

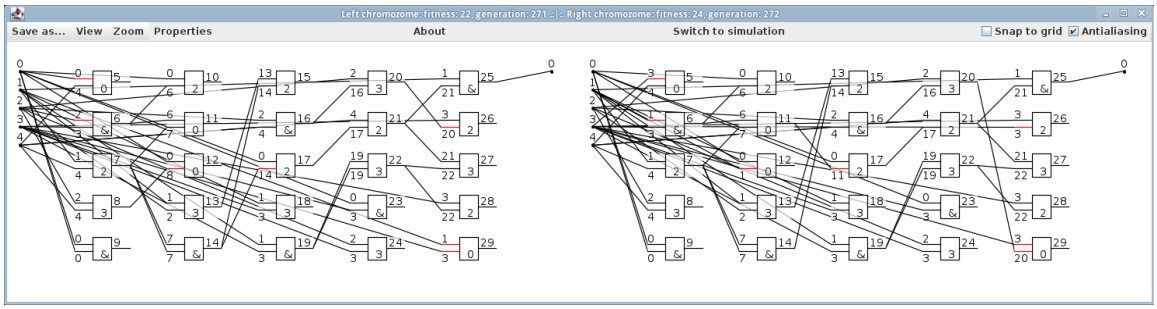
je odchytené ako pokus o posunutie hradla. V tomto zobrazení je teda možné aj posúvať hradlá a podobne.



Obrázek 6.12: Zobrazenie histórie chromozómu. V dolnej časti sa nachádza paleta použitých farieb. Ako je možné vidieť, tak v tomto prípade nastali zmeny v nedávnych generáciách, farby sú veľmi blízko seba v palete. Malá čierna bodka ukazuje aktuálnu generáciu v palete farieb, presné poradové číslo sa nachádza v titulku okna. Podrobnosti sú zobrazené v obdĺžniku vedľa kurzoru myši.

6.7 Porovnanie rozdielov dvoch chromozómov

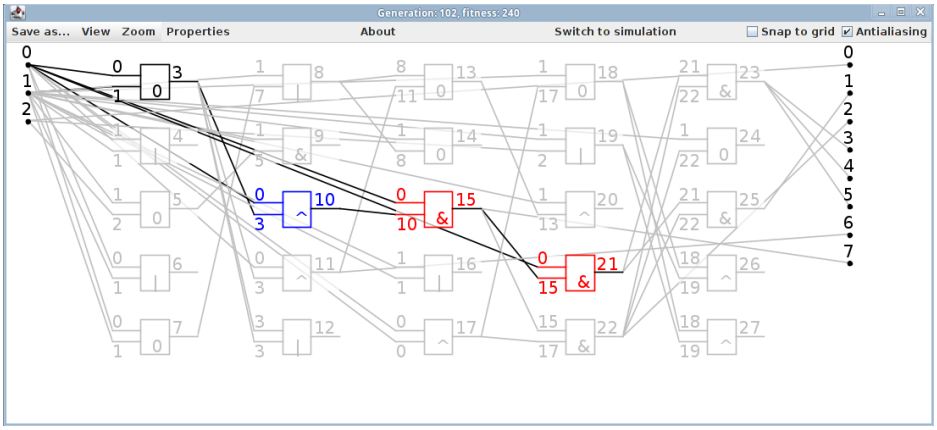
V zobrazení dvoch chromozómov je možné si nechať označiť rozdiely medzi nimi. Na toto označenie som vybrala červenú farbu, pretože je dobre viditeľná a dostatočne kontrastná s čiernou. Zisťovanie rozdielov prebieha po kliknutí na voľbu v menu. Pri vykreslení každé hradlo porovná svoje hodnoty s hodnotami odpovedajúceho hradla v druhom chromozóme a pokiaľ sa líšia, označí ich červenou farbou, viď obrázok 6.13. V titulku okna sa nachádzajú údaje o zobrazovaných chromozómoch.



Obrázek 6.13: Rozdiely zvýraznené červenou farbou.

6.8 Zobrazenie podgrafu v chromozóme

V zobrazení jedného aj dvoch chromozómov je možné nechať si zobraziť podgraf chromozómu. Slúži na to možnosť **Allow showing subgraph** v menu. Po kliknutí na túto možnosť sa viditeľne nestane nič, iba sa nastaví prepínač, avšak týmto sa povolí možnosť kliknúť na hradlo a zobraziť si jeho podgraf. Znamená to, že najprv sa všetkým hradlám nastaví viditeľnosť na `GREY`, ako pri skrývaní a zošednutí hradiel, a potom sa prechádza obvod od kliknutého hradla smerom k vstupom obvodu a jednotlivé hradlá sa nastavia ako viditeľné, viď obrázok 6.14. Týmto spôsobom je možné presne skúmať podgrafy v chromozóme. Bolo by vhodné zobraziť aj pravdivostnú tabuľku, v ktorej by ako vstupy boli vstupy celého obvodu a ako výstup by bol výstup vybraného hradla, avšak pri väčšom počte vstupov by tabuľka bola príliš veľká, pričom užívateľa by určite nezaujímali úplne všetky možnosti vstupov, pretože často sa stáva, že nejaký obvod má definované iba určité kombinácie vstupov a tie ostatné sú nevalidné, napríklad kóder. Pre konkrétne kombinácie vstupov je možné použiť simuláciu.



Obrázek 6.14: Zvýraznený podgraf v chromozóme.

Kapitola 7

Príklady použitia

Výsledný program je vhodný hlavne pre experimentálne účely, pre sledovanie zmien v menších obvodoch a s obmedzeným množstvom generácií. Je však samozrejme možné nechať si do záznamu behu CGP vypisovať napríklad iba každú desiatu alebo stú generáciu a podobne. Dostupné sú rôzne možnosti zvýrazňovania, z ktorých je možné vysledovať rôzne zmeny, rozdiely a správanie evolúcie v konkrétnych prípadoch.

7.1 Vytvorenie záznamu behu CGP

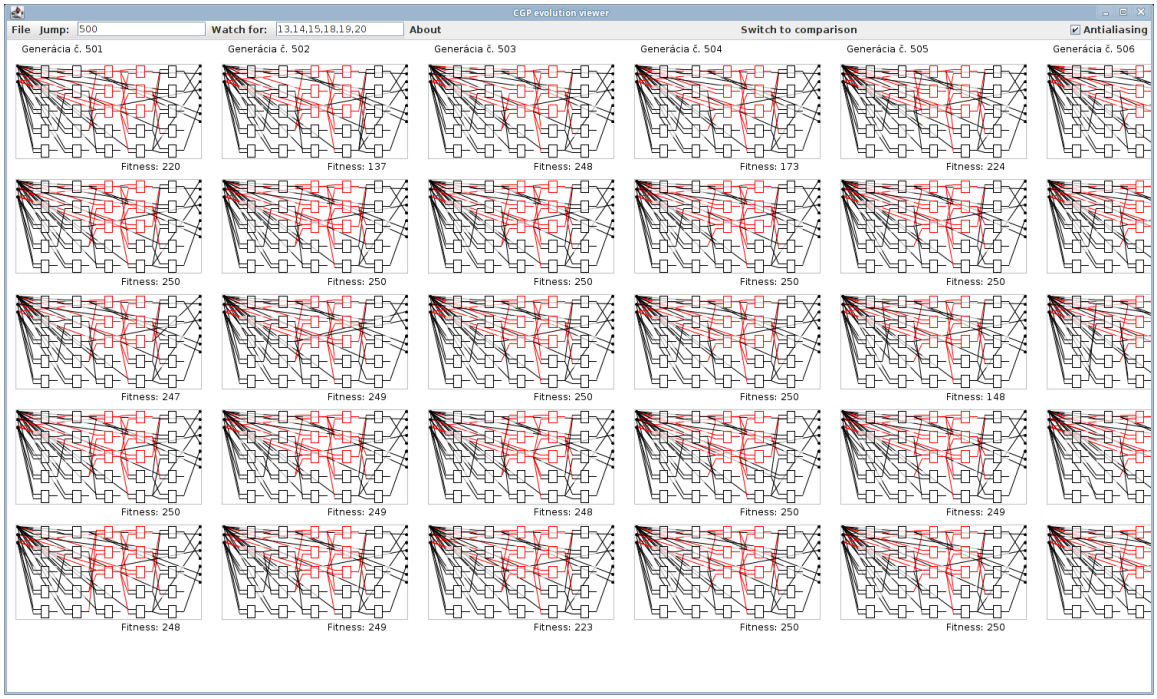
Najprv je potrebné vytvoriť pravdivostnú tabuľku žiadaného obvodu v správnom formáte, ktorý je uvedený v [11] (kde je aj odkaz na stiahnutie programu `tab2h`), z ktorého program `tab2h` vytvorí hlavičkový súbor v programovacom jazyku C. Názov tohto súboru sa vloží do hlavičkového súboru programu `cgp` [12]. Ten sa potom spustí s vhodnými parametrami a presmerovaním výstupu do súboru s príponou `.ev`. Tento postup som zopakovala s niekoľkými typmi obvodov. Výsledné súbory, ktoré som použila na otestovanie, je možné nájsť v prílohe na CD.

7.2 Zobrazenie

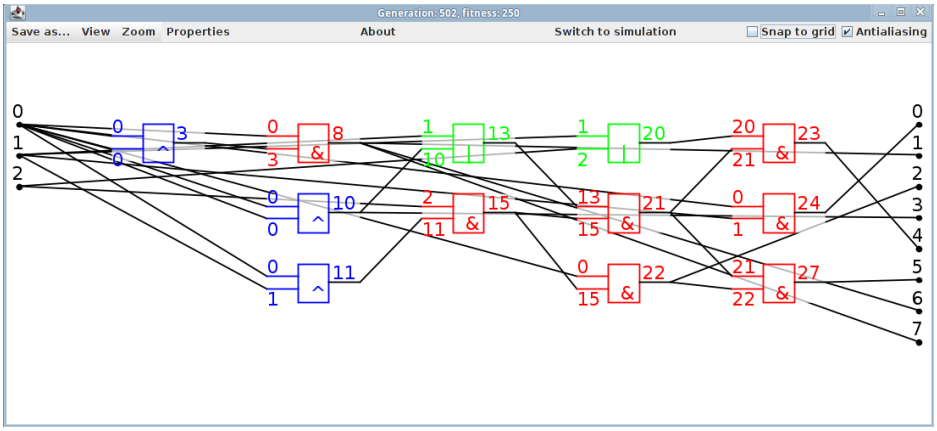
Na obrázku 7.1 je možné vidieť niekoľko generácií z evolúcie dekodéru. Nechala som program zvýrazniť podgraf, čo je v praxi užitočné pre sledovanie zmien. Spolu so zobrazením fitness to pomáha zisťovať, kde a kedy sa stala dôležitá zmena, či sa dostala do ďalších generácií, alebo zanikla a podobne. Z obrázku 7.2 je vidieť, ktoré hradlá sú použité pre získanie výsledku. Taktiež je vidieť, koľko hradiel ktorého typu sa podieľa na výsledku. Na obrázku 7.3 sú vidieť rozdiely medzi dvoma chromozómami vzdialenými od seba o sto generácií.

Ďalším príkladom je evolúcia kóderu, ktorá skončila úspešne, takže v prípade zobrazenia chromozómu z poslednej generácie je možné si odsimulovať správne správanie obvodu, viď obrázok 7.4. V tabuľke 7.1 je vidieť očakávané výsledky obvodu (nezadané kombinácie vstupov sú nevalidné). Na vstupe kóderu na obrázku 7.4 je vytvorená kombinácia z predposledného riadku tabuľky a je vidieť, že výstup je správny. Červenou je znázornená logická jedna, čiernou logická nula.

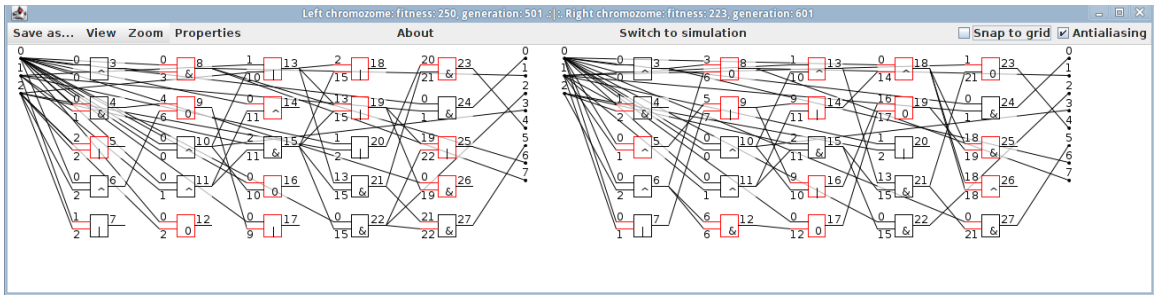
Na obrázku 7.5 je vidieť históriu kóderu v poslednej generácii úspešného behu CGP. Keďže zobrazenie histórie berie do úvahy všetky hradlá v každej generácii, vo väčšine prípadov sa stane niečo podobné ako v tomto prípade. Je vidieť, že farby sú takmer rovnaké a podľa palety farieb je vidieť, že znázorňujú nedávne generácie. Znamená to, že



Obrázek 7.1: Evolúcia dekodéru so zvýrazneným podgrafom.



Obrázek 7.2: Detail dekodéru so zafarbenými hradlami a skrytými nepotrebnými hradlami.

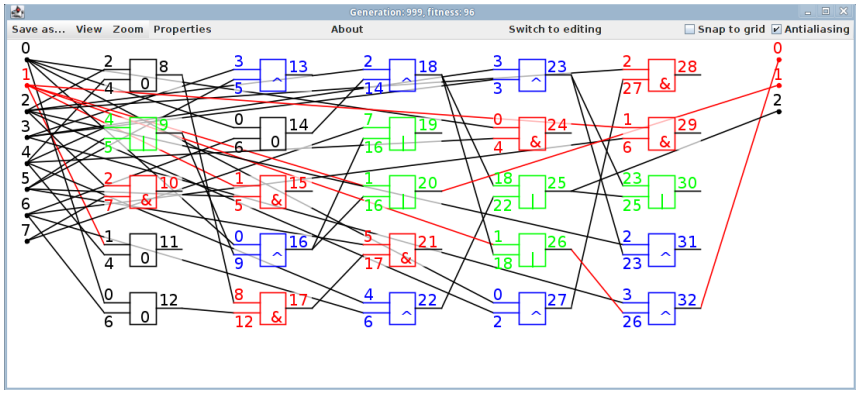


Obrázek 7.3: Rozdiely medzi dvoma chromozómami.

každá zmena v tomto obvode sa udiala len v nedávnych generáciách, pričom tieto zmeny sa nemuseli udiať v jednom chromozóme, ale vo viacerých, čo však toto zobrazenie nemá ako zistiť. Z tohto zobrazenia nie je možné zistiť presne z ktorého chromozómu sa vyvinul tento konkrétny. Na zobrazenie takejto informácie by bolo potrebné najprv túto informáciu mať obsiahnutú v zázname behu CGP. Znamenalo by to vytvoriť línie rodičov a potomkov, aby bolo zjavné, ktorý chromozóm bol vytvorený mutáciou z ktorého z predchádzajúcej generácie.

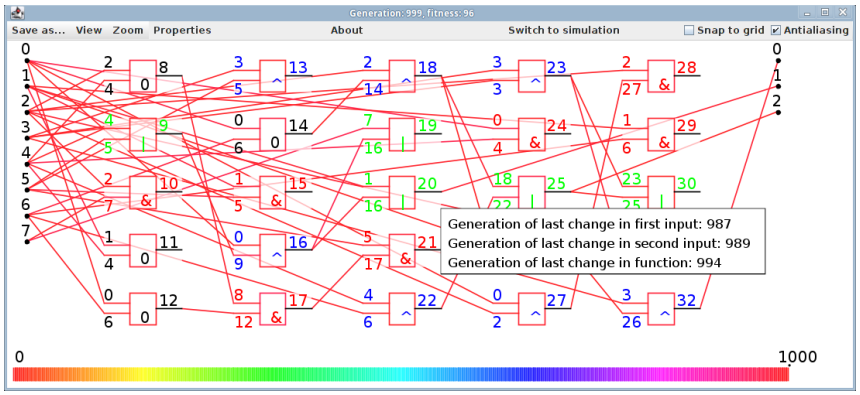
vstupy	výstupy
00000001	000
00000010	001
00000100	010
00001000	011
00010000	100
00100000	101
01000000	110
10000000	111

Tabulka 7.1: Pravdivostná tabuľka kóderu.

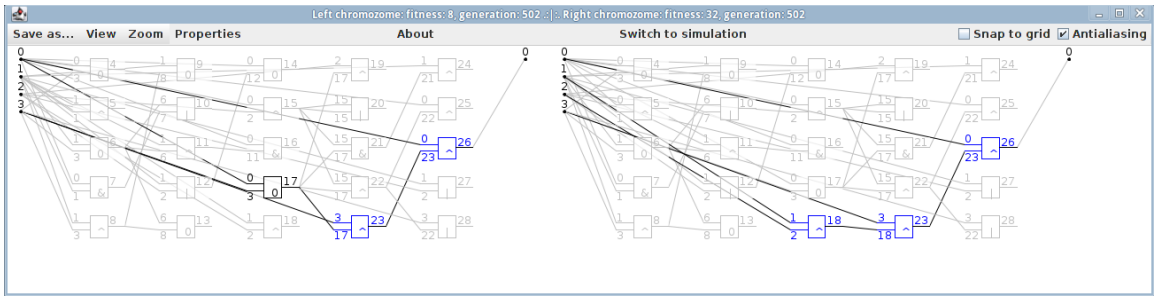


Obrázek 7.4: Simulácia kóderu v poslednej generácii jedného behu CGP.

Pri zobrazení výpočtu parity pre štyri bity som si nechala zvýrazniť podgraf, viď obrázok 7.6, v zobrazení dvoch chromozómov. Z tohto zobrazenia je možné lepšie pochopiť ako vplýva malá zmena na konkrétny podgraf a na celkový výsledok. Ako je vidieť v titulku okna na obrázku 7.6, ľavý chromozóm má fitness 8 a pravý 32, pričom v obvode nastala len malá, ale dôležitá zmena.



Obrázek 7.5: História kóderu v poslednej generácii jedného behu CGP.



Obrázek 7.6: Zvýraznenie podgrafu v chromozóme, ktorý reprezentuje obvod pre výpočet parity pre štyri bity.

Kapitola 8

Záver

Cieľom tejto práce bolo zoznámiť sa s teoretickým základom CGP a jeho použitím z pohľadu evolučného návrhu obvodov. Taktiež bolo dôležité zoznámiť sa so situáciou na poli programov, ktoré nejakým spôsobom umožňujú vizualizáciu a prácu s CGP a jeho chromozómami. Na tomto základe vznikol návrh konceptu vizualizácie CGP spolu s rozhodnutím o potrebnom software pre implementáciu. S využitím týchto znalostí a existujúcej bakalárskej práce bol návrh konceptu úspešne implementovaný a otestovaný na ukážkových problémoch.

Vo vizualizácii je dôležité mať možnosť manipulovať s rôznymi prvkami. V zobrazení celej evolúcie je to možnosť presúvať sa medzi generáciami a možnosť výberu jedného alebo dvoch chromozómov a ďalšia práca s nimi. Tá spočíva v presúvaní jednotlivých hradiel, ich zafarbovaní, zobrazovaní histórie, jednoduchej simulácie, odstraňovaní hradiel, ktoré sa nepodieľajú na výsledku obvodu, približovaní a oddaľovaní obrazu a podobne. Všetky tieto vlastnosti sú dôležité pri práci programátorov CGP.

Prínosom tejto práce je vytvorenie prehľadu v problematike vizualizácie CGP a vytvorenie komplexného programu, ktorý zatiaľ nemá inú alternatívu a je použiteľný v praxi. Výhodou je možnosť simulácie, vďaka ktorej si programátori môžu okamžite vyskúšať funkčnosť vybraného chromozómu. Taktiež je praktické zobrazenie histórie, kde môže programátor sledovať postupnosť zmien na rôznych miestach v chromozóme. Určite sú však možné aj rozšírenia tohto programu, napríklad integrácia samotnej evolúcie CGP a po jej dokončení okamžité zobrazenie výsledkov a možná manipulácia s nimi.

Literatura

[1] Batik SVG Toolkit. online, 12. mája 2012.
URL http://xmlgraphics.apache.org/batik/

[2] Clegg, J.; Walker, J.; Miller, J.: A New Crossover Technique for Cartesian Genetic Programming. In *Proc. Genetic and Evolutionary Computation Conference*, ACM Press, 2007, s. 1580–1587.

[3] Harding, S. L.; Miller, J. F.; Banzhaf, W.: Self-Modifying Cartesian Genetic Programming. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 101–124.

[4] Miller, J.; Thomson, P.: Cartesian Genetic Programming. In *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, *LNCS*, ročník 1802, Springer Verlag, 2000, s. 121–132.

[5] Miller, J. F.: Cartesian Genetic Programming. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 17–34.

[6] Miller, J. F.: Software. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 338–339.

[7] Schwarz, J.; Sekanina, L.: Aplikované evoluční algoritmy. 2006.

[8] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Praha: Academia, 2009, ISBN 9788020017291.

[9] Sekanina, L.; Walker, J. A.; Kaufmann, P.; aj.: Self-Modifying Cartesian Genetic Programming. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 125–176.

[10] Staurovská, J.: *Grafické rozhraní pro manipulaci s chromozomy genetického programování v Javě, bakalářská práce*. Brno: FIT VUT v Brně, 2010.

[11] Vašíček, Z.: Návod pre program tab2h.
URL http://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/

[12] Vašíček, Z.: Program cgp. 2006.
URL http://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/bin/cgp.zip

[13] Vašíček, Z.: Program cgpviewer. 2006.
URL http://www.fit.vutbr.cz/~vasicek/cgp/bin/viewer.zip

[14] Vašíček, Z.; Sekanina, L.: Evoluční návrh kombinačních obvodů. *Elektrorevue - www.elektrorevue.cz*, ročník 2004, č. 43, 2004: s. 1–6.

Příloha A

Obsah CD

Zložky:

- technicka sprava – zdrojovy kod pre LaTeX — Elektronická verzia technickej správy vo forme PDF a jej zdrojový text pre LaTeX
- CGPevolutionViewer — Zdrojový kód programu v Jave, ant súbor pre vytvorenie .jar archívu, spustiteľný .jar archív
- ukazkove subory — Ukážkové súbory so záznamom evolúcie, ukážkový súbor s nastavením
- uzivatelska prirucka — Užívateľská príručka v HTML

Příloha B

Návod na použitie

Prerekvizity pre preklad: Java SDK (testované na verzii 1.6.0_24, OpenJDK Runtime Environment (IcedTea6 1.11.1), OpenJDK 64-Bit Server VM (build 20.0-b12, mixed mode)).

Príkaz na spustenie prekladu `ant` musí byť spustený v zložke projektu, ktorá obsahuje zdrojový kód a súbor `build.xml`, ktorý načíta a vytvorí spustiteľný `.jar` súbor. Je pravdepodobné, že v operačnom systéme Linux bude potrebné označiť tento súbor ako spustiteľný. Spustiť program je možné dvoma spôsobmi. Buď obvyklým poklikaním na súbor v prehliadači súborov, alebo z príkazovej riadky pomocou príkazu `java -jar cgpviewer.jar`. Užívateľská príručka sa nachádza na CD vo formáte HTML, takže na jej prečítanie je potrebné mať nainštalovaný nejaký prehliadač webových stránok.