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 ANALÝZU ZÁZNAMŮ O PRŮBĚHU EVOLUCE ČÍSLICOVÉHO OBVODU

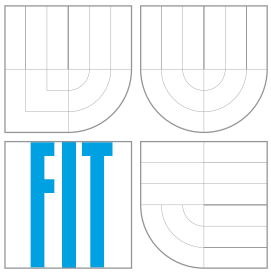
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLASTIMIL KAPUSTA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁSTROJ PRO ANALÝZU ZÁZNAMŮ O PRŮBĚHU EVOLUCE ČÍSLICOVÉHO OBVODU

A TOOL FOR ANALYSIS OF DIGITAL CIRCUIT EVOLUTION RECORDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLASTIMIL KAPUSTA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce popisuje stochastické optimalizační algoritmy inspirované přírodou, které využívají populaci jedinců – konkrétně evoluční algoritmy. Blíže je popsáno genetické programování a jeho varianta – kartézské genetické programování. Dále se práce zaměřuje na analýzu a vizualizaci záznamů o průběhu evoluce číslicového obvodu. Byly zmapovány existující nástroje pro vizualizaci průběhu evoluce obvodů. Protože nebyl nalezen vyhovující nástroj, který by umožnil komplexní analýzu průběhu evoluce obvodů, byla pro tento účel navržena sada analytických funkcí. Navržené funkce byly implementovány ve formě interaktivního nástroje s grafickým uživatelským rozhraním v jazyce Java. Vytvořená aplikace byla detailně popsána a poté použita k analýze zvolených evolučních záznamů.

Abstract

This master thesis describes stochastic optimization algorithms inspired in nature that use population on individuals – evolutionary algorithms. Genetic programming and its variant – cartesian genetic programming is described in a greater detail. This thesis is further focused on the analysis and visualization of digital circuit evolution records. Existing tools for visualization of the circuit evolution were analysed, but because no suitable tool allowing complex analysis of the circuit evolution was found, a new set of functions was proposed and the principles of a new tool were formulated. These functions were implemented in form of an interactive GUI application in Java programming language. The application was described in detail and then used for analysis of digital circuit evolution records.

Klíčová slova

Evoluční algoritmy, kartézské genetické programování, vizualizace, evoluce číslicového obvodu, GUI, Java.

Keywords

Evolutionary algorithms, cartesian genetic programming, visualization, digital circuit evolution, GUI, Java.

Citace

Vlastimil Kapusta: Nástroj pro analýzu záznamů o průběhu evoluce číslicového obvodu, diplomová práce, Brno, FIT VUT v Brně, 2015

Nástroj pro analýzu záznamů o průběhu evoluce číslicového obvodu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Prof. Ing. Lukáše Sekaniny, Ph.D. a uvedl jsem všechny zdroje, ze kterých jsem čerpal.

.....
Vlastimil Kapusta
24. května 2015

Poděkování

Chtěl bych poděkovat panu Prof. Ing. Lukáši Sekaninovi, Ph.D. za cenné rady a pomoc při zpracování této práce.

Obsah

Kapitola 1

Úvod

Optimalizační algoritmy slouží k nalezení optimálního, popřípadě co nejlepšího řešení problému. Existuje několik skupin optimalizačních algoritmů: deterministické, pravděpodobnostní nebo kombinované. Tato práce se zaměřuje na stochastické optimalizační metody inspirované přírodou, které využívají populaci jedinců – konkrétně na evoluční algoritmy. Při snaze porozumět činnosti evolučního algoritmu vzniká potřeba vizualizace průběhu výpočtu z důvodu možnosti provedení následné analýzy, která by mohla vést k vylepšení nastavení parametrů optimalizačního algoritmu, přizpůsobení genetických operátorů řešenému problému nebo akceleraci algoritmu.

Cílem této práce je zmapovat a navrhnout možnosti vizualizace průběhu evoluce, především kartézského genetického programování a poté implementovat vybrané funkce v podobě aplikace s grafickým uživatelským rozhraním, která umožní analyzovat průběh evoluce.

Kapitola 2 se zabývá evolučními algoritmy. Jsou zde popsány společné rysy a vlastnosti evolučních algoritmů a jednotlivé skupiny těchto algoritmů: evoluční strategie, evoluční programování, genetické algoritmy a genetické programování. V sekci 2.6 je dále podrobně rozebráno kartézské genetické programování – jeho vlastnosti, vstupy a výstupy, kódování a dekódování genotypu, vlastní algoritmus, genetické operátory apod. Dále v této kapitole nalezneme popis vytvořeného záznamu evoluce obvodu pomocí CGP.

Kapitola 3 zkoumá možnosti vizualizace průběhu evoluce. Jsou zde představeny existující nástroje pro vizualizaci a jejich možnosti.

V kapitole 4 nalezneme nastínění vlastností, které by měl mít zamýšlený nástroj a seznam navržených funkcí pro vizualizaci záznamu evoluce. Dále jsou blíže popsány zvolené funkce pro vyvíjený nástroj.

Kapitola 5 popisuje prvotní návrh implementace vyvíjené aplikace a pomocí diagramů tříd je vysvětleno uspořádání dat v aplikaci.

V kapitole 6 je představena vytvořená aplikace. Pomocí obrázků jsou popsány části aplikace a všechny její funkce.

Kapitola 7 popisuje důležité implementační detaily, sloužící pro dodatečné vysvětlení funkcí aplikace.

V kapitole 8 je předvedeno využití aplikace na vygenerovaných záznamech o průběhu evoluce. Tyto záznamy jsou detailně prozkoumány a popsány.

Kapitola 9 obsahuje závěr.

Kapitola 2

Evoluční algoritmy

Pro všechny algoritmy, které řadíme do skupiny evolučních algoritmů, platí společná idea, že existuje populace jedinců (individuí), na které působí tlak prostředí způsobující přirozený výběr (přežití nejschopnějších). V průběhu evoluce dochází k růstu kvality (fitness) populace. Mějme funkci kvality, kterou maximalizujeme. Můžeme náhodně vytvořit populaci kandidátních řešení a aplikovat na ně funkci kvality jako abstraktní měření hodnoty fitness (kvality jedince), kde větší dosažená hodnota je lepší. Podle této fitness jsou vybráni někteří lepší jedinci k vytvoření nové generace s použitím rekombinace (křížení) anebo mutace.

Rekombinace je operátor aplikovaný na dva či více vybraných jedinců (nazývaných rodiče). Výsledkem je jeden nebo více jedinců (potomků).

Mutace je aplikována na jednoho jedince a vznikne zase jeden zmutovaný jedinec.

Použití rekombinace anebo mutace vede ke skupině nových jedinců (potomků), kteří soupeří na základě hodnoty své fitness s rodiči o místo v další generaci. Tento proces může být opakován, dokud není nalezen jedinec s dostatečnou kvalitou (řešení) nebo je dosažen maximální počet iterací.

V tomto procesu jsou dvě základní „síly“, které formují základ evolučních systémů:

- Variační operátory (křížení a mutace) vytváří nezbytnou diverzitu, čímž přináší novinky.
- Proces selekce jako síla protlačující kvalitu.

Kombinace variace a selekce obecně vede ke zlepšující se hodnotě fitness v dalších populacích. Můžeme tak evoluci vidět jako optimalizační proces, který se postupně přibližuje k nejvýhodnějšímu řešení. Alternativně může být evoluce vnímána jako proces adaptační. Z tohoto pohledu je fitness vyjádřením požadavků prostředí. Jedinec lépe splňující tyto požadavky je více životaschopný, což je reflektováno větším množstvím potomků. Evoluční proces dělá populaci čím dál lepší, adaptováním se na prostředí.

Mnoho částí takového evolučního procesu je stochastických. Při výběru mají lepší jedinci větší šanci být vybráni než horší jedinci, ale i ti mají šanci být vybráni a stát se rodiči nebo přežít. Při křížení je výběr částí jedince, které budou kříženy, náhodný. Podobně je tomu u mutace: části, které budou mutovány v rámci jedince, a části, kterými budou ty původní nahrazeny, budou vybrány náhodně. Základní schéma evolučního algoritmu můžeme vidět v algoritmu 1 [9].

Algoritmus 1: Evoluční algoritmus

```
1: Inicializace populace náhodnými kandidátními řešeními
2: Ohodnocení všech jedinců
3: while Ukončovací podmínka není splněna do
4:     Výběr rodičů
5:     Křížení párů rodičů
6:     Mutace vzniklých potomků
7:     Ohodnocení nových jedinců
8:     Selekce individuí pro novou generaci
9: end
```

2.1 Vlastnosti evolučních algoritmů

Prvním krokem k definování EA je spojení s reálným světem, neboli vztah mezi reálným problémem a prohledávaným prostorem, kde probíhá evoluce. Objekty, reprezentující možná řešení v kontextu původního problému, se nazývají fenotypy, zatímco jejich zakódování, což jsou jedinci v EA, se nazývají genotypy. Mapování fenotypu na genotyp se běžně říká reprezentace. Základní část jedince se nazývá gen a jeho hodnota je alela.

Ohodnocující (evaluační/fitness) funkce reprezentuje požadavky prostředí. Definuje, co znamená zlepšení a umožňuje následně selekci. Je to funkce, která genotypu přiřazuje skóre.

Populace obsahuje kandidátní řešení. Je to multimnožina genotypů. Narozdíl od variačních operátorů, které pracují s několika jednotlivci, selekční operátory pracují na úrovni populace. Ve většině EA je velikost populace konstantní a v průběhu evoluce neměnná. Měřítko počtu různých řešení v populaci se nazývá diverzita populace.

Mechanismus výběru rodičů má za úkol umožnit lepším jedincům stát se rodiči další generace. Společně s mechanismem pro výběr přeživších jedinců je zodpovědný za tlak na zlepšování kvality. V EA je tento mechanismus většinou pravděpodobnostní, což dává méně kvalitním jedincům, bez kterých by algoritmus v budoucnu mohl skončit v lokálním optimu, šanci stát se také rodiči.

Variační operátory vytváří nové jedince ze starých. Mutace je unární variační operátor, aplikovaný na jednoho jedince, jehož výstupem je zmutovaný jedinec. Jedná se o stochastický operátor, protože výstup záleží na sérii náhodných voleb. Úloha mutace se mění podle varianty EA. Křížení je binární operátor [1], spojující informace ze dvou rodičovských genotypů. Podobně jako mutace je rekombinace stochastický operátor: volba rodičů, míst a způsobu kombinace rodičů závisí na náhodě. Úloha křížení opět závisí na variantě EA. Základním principem křížení je předpoklad, že spojením dvou jedinců s odlišnými, ale žádoucími vlastnostmi, může vzniknout potomek kombinující tyto vlastnosti. Je důležité zmínit, že variační operátory jsou závislé na reprezentaci. Pro různé reprezentace musí být definovány různé variační operátory.

Mechanismus výběru přeživších jedinců (tzv. nahrazení) slouží k rozlišení jedinců na základě kvality. Jedná se o podobný mechanismus jako je výběr rodičů, ale je použit v jiné části EA. Protože populace v EA bývá konstantní, je třeba učinit rozhodnutí, kteří jedinci budou vybráni do další generace, většinou podle jejich fitness. Oproti stochastickému výběru

[1] Rekombinační operátory s vyšší aritou jsou matematicky možné a jednoduše implementovatelné, ale nemají biologický ekvivalent a proto se nejspíše běžně nepoužívají, ačkoliv některé studie [10] naznačují, že mohou mít pozitivní efekt na evoluci.

rodičů je tento mechanismus většinou deterministický.

Inicializační krok bývá většinou velmi jednoduchý. Počáteční populace se vytvoří generováním náhodných jedinců. Mohou být použity problémově-specifické heuristiky pro zvýšení fitness počáteční populace.

Ukončovací podmínka může být dvou typů. Pokud má problém známou úroveň optimální hodnoty fitness, pak by dosažení tohoto optima mělo být ukončovací podmínkou. Protože EA jsou stochastické a většinou negarantují nalezení optima, tato podmínka nemusí být nikdy splněna. Je tedy třeba rozšířit ukončovací podmínku tak, aby se algoritmus někdy zcela jistě zastavil. Běžně se používá některá z následujících variant:

1. Uplynutí maximálního povoleného času CPU.
2. Dosažení limitu počtu evaluací fitness.
3. Nedostatečné zlepšení fitness za určitý čas.
4. Pokles diverzity populace pod určitou hranici.

Ukončovací podmínka má většinou tvar disjunkce: buď je dosaženo optima *nebo* je splněna podmínka x. Pokud problém nemá známé optimum, není potřeba disjunkce, ale prostě stačí jedna podmínka ze seznamu výše [9].

Dle [14] jsou nejznámější varianty evolučních algoritmů:

- evoluční strategie (ES),
- evoluční programování (EP),
- genetické algoritmy (GA) a
- genetické programování (GP).

2.2 Evoluční strategie

Tato metoda, původem německá, byla zpočátku využívána pro optimalizaci složitých inženýrských úloh v oblasti aerodynamiky. ES typicky slouží k optimalizaci vektoru reálných parametrů. Základní varianta používá pouze operátor mutace, využívající Gaussovo rozložení s nulovou střední hodnotou a rozptylem σ. Parametr σ je modifikován tak, aby pětina potomků byla lepší, než jsou rodiče. Pokud je úspěšných potomků méně, vliv mutací se sníží, pokud je úspěšných potomků více, vliv mutací se zvýší. Tento postup je znám jako „pravidlo jedné pětiny“.

Základní varianta ES je označována jako ES(1+1), což znamená, že z jednoho rodiče je mutací vytvářen jeden potomek. Pokud má potomek větší hodnotu fitness než jeho rodič, stává se rodičem pro novou generaci, jinak se v nové generaci použije rodič z minulé generace. V současnosti se nejvíce používají dvě varianty s populacemi, které obsahují větší množství jedinců a to ES($\mu + \lambda$) a ES(μ, λ). Parametr μ označuje počet rodičů a parametr λ počet potomků. Varianta ES($\mu + \lambda$) vybere μ nejlepších jedinců ze všech dostupných možností. Varianta ES(μ, λ) vybírá μ nejlepších jedinců z množiny potomků. Rodiče nové generace jsou vybíráni deterministicky.

ES využívají prvek, netypický pro jiné evoluční techniky. Parametry ES jsou zakódovány

do chromozomu a evolučně se mění spolu s parametry hledané funkce. Jedná se tedy o *samo-adaptaci*. V chromozomu pak najdeme např. rozptyl pro každý z parametrů optimalizované funkce [22].

Pokud je řešení problému reprezentováno l atributy, potom se jedinec v pokročilé variantě ES může skládat z $l(l+3)/2$ reálných hodnot, ze kterých

- l hodnot reprezentuje přímo hodnoty atributů $h_1, \ldots, h_l$,
- l hodnot reprezentuje směrodatné odchylky $\sigma_1, \ldots, \sigma_l$,
- $l(l-1)/2$ hodnot reprezentuje rotační úhly $\alpha_{1,2}, \ldots, \alpha_{l-1,l}$.

Mutace je realizována jako trojice za sebou následujících mutačních kroků, přičemž každý z nich operuje nad jinou částí jedince. Nejprve se určí nová hodnota směrodatných odchylek. Poté se určuje nová hodnota rotačních úhlů. Na konci procesu dochází ke změně hodnot atributů. Vygenerují se hodnoty odchylek pro všechny atributy, přičemž pro každou z nich se použije normální rozdělení pravděpodobnosti s nulovou střední hodnotou a novou hodnotou směrodatné odchylky, která přísluší danému atributu. Tyto odchylky jsou upravené pomocí rotačních matic, využívajících změněné hodnoty rotačních úhlů. V každé hodnotě je zohledněna odchylka vyprodukovaná rotačním procesem [15].

2.3 Evoluční programování

EP vzniklo v USA k evoluci automatů pro účely predikce posloupností. Dále je popsán algoritmus založený na variantě známé jako metaevoluční programování [1].

Jedinec kóduje nejen řešení, ale i parametry algoritmu. Pokud je řešení problému reprezentované pomocí l atributů, potom se jedinec skládá z $2l$ reálných hodnot, z kterých

- l hodnot reprezentuje přímo hodnoty atributů $h_1, \ldots, h_l$,
- l hodnot reprezentuje rozptyl $\sigma_1^2, \ldots, \sigma_l^2$.

Každá z hodnot atributů je ve fázi inicializace obvykle omezena na určitý podinterval reálných čísel $< min(v_i); max(v_i) >$. Při běhu algoritmu na uvedené omezení není brán zřetel – prohledávaný prostor je neomezený, pokud to nevylučuje charakter úlohy.

Podobně i rozptyly jsou omezeny na interval $< 0; k >$, kde $k > 0$, pouze při inicializaci a v průběhu algoritmu mohou nabývat libovolných kladných reálných hodnot.

Selekce je realizována jako elitistická selekce bez vytváření selekčního tlaku. Jedinci jsou do úlohy rodičů vybíráni deterministicky – každý jedinec se stává rodičem.

Jediným genetickým operátorem použitým v tomto přístupu je mutace. Křížení není použito v žádné podobě. Operátor mutace je realizovaný jako dvojice za sebou následujících mutačních kroků, kde každý z nich operuje nad jinou částí jedince:

$$h_i' = h_i + \sqrt{\sigma_i^2} N_i(0,1) \tag{2.1}$$

$$(\sigma_i^2)' = \sigma_i^2 + \sqrt{\xi \sigma_i^2} N_i(0,1) \tag{2.2}$$

Nejprve dochází ke změně hodnot atributů, potom se určí nová hodnota směrodatných odchylek s použitím škálovacího parametru ξ. $N_i(0,1)$ reprezentuje hodnotu generovanou podle normálního rozložení pravděpodobnosti.

Pro výběr jedinců do nové populace se používá kombinace pravděpodobnostně deterministického výběru elitistického charakteru. Protože každý rodič vyprodukoval jednoho potomka, je k dispozici μ jedinců aktuální populace a μ nových potomků. Při výběru do nové populace jedinec nesoutěží se všemi ostatními $2\mu - 1$ jedinci, ale pro každého jedince se náhodně vybere q jedinců, se kterými bude soutěžit. Jedinci jsou poté seřazeni podle počtu vítězství a μ nejlepších vytvoří další generaci [15].

2.4 Genetické algoritmy

Genetický algoritmus původně sloužil ke studiu adaptivního chování, dnes je ale GA považován za optimalizační metodu. Oproti ES se nepoužívá pouze na numerickou optimalizaci, ale v řadě dalších oblastí, např. pro kombinatorickou optimalizaci. Tato kapitola je zpracována podle [9].

Velmi důležitým krokem v návrhu genetického algoritmu je volba vhodné reprezentace. První možností je binární reprezentace, kde chromozom sestává z řetězce binárních číslic. Problém bitového kódování spočívá v rozdílné důležitosti bitů, takže mutací jednoho bitu může vzniknout naprosto rozdílný fenotyp. Tomu lze předejít použitím Grayova kódování, které zaručuje, že po sobě jdoucí hodnoty, která jsou tímto způsobem zakódovaná, mají Hammingovu vzdálenost rovnu jedné. Další běžně používané reprezentace jsou celočíselná, reprezentace reálnými čísly a permutační reprezentace. V permutačním kódování je jedinec tvořen permutací dané množiny celých čísel. Je zřejmé, že pro toto kódování je zapotřebí upravených variačních operátorů.

Forma mutace závisí na zvoleném kódování, stejně jako příslušný parametr, často nazývaný četnost mutace. Nastavení četnosti mutace je velmi závislé na řešeném problému a požadovaném řešení.

Rekombinace neboli proces, kde je nový jedinec vytvořen z informace obsažené ve dvou (nebo více) rodičích, je považován za jeden z nejdůležitějších rysů genetických algoritmů. Mnohé výzkumy se na ni zaměřovaly jako na primární mechanismus vytváření diverzity. Je to rozhodně vlastnost, která nejvíce odlišuje GA a jiné EA od jiných optimalizačních metod. Operátory rekombinace bývají používány pravděpodobnostně. Typická pravděpodobnost křížení je v intervalu $< 0,5; 1,0 >$. Většinou jsou vybráni dva rodiče a na základě zvolené pravděpodobnosti křížení jsou noví potomci vytvořeni rekombinací nebo zkopírováním rodičů. V nové populace se tudíž nachází jak kopie rodičů, tak nová řešení.

V literatuře jsou rozeznávány dva modely GA:

- Generační model – každá generace začíná s populací velikosti μ, z které je vybrána podmnožina rodičů. Dále je aplikací variačních operátorů vytvořeno a ohodnoceno $\lambda\,(=\mu)$ potomků. Po každé generaci je celá populace nahrazena svými potomky.
- Steady-state model – celá populace se nemění najednou, ale po částech: $\lambda\,(<\mu)$ starých jedinců je nahrazeno λ novými potomky.

Výběr rodičů lze provést různými způsoby na základě fitness hodnoty jedince:

- Fitness proportional selection (FPS) – pravděpodobnost výběru jedince je absolutní hodnota jeho fitness v porovnání s fitness zbytku populace, neboli $f_i = f_i / \sum_{j=1}^{\mu} f_j$. Tento způsob výběru se ukázal jako problémový, protože vyčnívající jedinci - například takoví, jejichž fitness je o hodně lepší než fitness ostatních, velmi brzo vytlačí ostatní jedince z populace. Tento jev je znám jako předčasná konvergence. Další problém nastane, pokud jsou hodnoty fitness přibližně stejné, tak není vytvářen téměř žádný selekční tlak.

- Ranking selection – metoda navržená s přihlédnutím k problémům předchozího přístupu. Vytváří neustálý selekční tlak řazením populace podle fitness a přiřazováním pravděpodobnosti výběru na základě ohodnocení (rank) místo na základě vlastní hodnoty fitness. Mapování z ohodnocení na pravděpodobnost výběru je náhodné a může být provedeno různými způsoby.

- Turnajová selekce je vhodná pro výběr rodičů z velké populace jedinců, protože pro výběr rodiče potřebuje pouze $k \geq 2$ jedinců, ze kterých vybere jako rodiče toho nejlépe ohodnoceného.

Metody výběru jedinců do nové populace (výběr přeživších, nahrazení) mohou být rozděleny podle toho, jestli rozhodují na základě fitness nebo stáří jedince.

- Nahrazení na základě stáří nebere v úvahu fitness jedince, ale pracuje s myšlenkou, že každý jedinec existuje v populaci pouze po omezený počet iterací GA. Tato strategie se používá například v základní variance GA, kde je počet potomků stejný jako počet rodičů ($\mu = \lambda$), takže délka života každého jedince je jedna iterace GA. Poté jsou rodiče jednoduše nahrazeni potomky.

- Nahrazení na základě fitness určuje, kteří rodiče budou nahrazeni potomky podle hodnoty fitness.

- Nahrazení nejhorších jedinců funguje tak, že je vybráno λ nejhorších jedinců populace, kteří budou nahrazeni. Toto může vést k rychlému růstu průměrné hodnoty fitness populace, ale také k předčasné konvergenci, protože se populace omezuje jen na jedince s nejlepší hodnotou fitness.

- Elitismus je metoda bránící ztrátě jedince s nejlepší fitness. Je udržován záznam o nejlepším jedinci a ten je vždy vybrán do další generace.

2.5 Genetické programování

Podle [22] je cílem této metody je kromě evolučního hledání optimálních hodnot parametrů, zejména automatické generování celých programů.

Od genetického algoritmu se GP liší v následujících bodech. V reprezentaci, protože GP pracuje se spustitelnými strukturami – nejčastěji s programy reprezentovanými stromy proměnné délky. Dále se liší v genetických operátorech, protože v GP pracují nad spustitelnými strukturami a kromě křížení a mutace existují i pokročilé operátory, například pro evoluční vytváření podprogramů v rámci vyvíjených programů. Pro zjištění fitness je proveden kód kandidátního programu pro definovanou množinu vstupů a jsou vyhodnoceny

získané výsledky.

Před řešením úlohy pomocí GP je třeba provést tyto přípravné kroky:

1. definovat množinu terminálů
2. definovat množinu funkcí
3. definovat způsob výpočtu hodnoty fitness
4. definovat parametry GP (velikost populace, počet generací atd.)
5. definovat způsob určení výsledku a způsob ukončení evoluce.

Množina terminálů reprezentuje vstup do programu vyvíjeného pomocí genetického programování, konstanty a funkce bez argumentů s vedlejším účinkem. V klasické variantě GP je pro populaci zvolena množina základních konstant, které se v průběhu evoluce nemění. Nová konstanta může být vytvořena kombinací základních konstant a aritmetických funkcí.

Množina funkcí může být specifická pro danou oblast nebo obecná. Funkce by neměly být zbytečně složité, aby jejich vyčíslení nebylo příliš časově náročné. Výstup funkce nebo hodnota terminálu musí být použitelná jako argument ostatních funkcí. Pro funkce, které nejsou definovány pro určité hodnoty, se v genetickém programování používají tzv. *chráněné varianty*. Kdyby mělo při evaluaci dojít například k dělení nulou, vrátí chráněné dělení nějakou bezpečnou, předem zvolenou hodnotu.

Programy jsou v genetickém programování chápány jako struktury sestávající z funkcí a terminálů. Dále jsou definována pravidla použití jednotlivých funkcí a terminálů. Základními strukturami, používanými v genetickém programování, jsou stromy, lineární struktury a obecné grafové struktury. Průchodem stromu dojde k vykonání programu, který reprezentuje. Příkladem systému s grafovou reprezentací je kartézské genetické programování, popsané v sekci 2.6.

Při inicializaci se do programů náhodně vybírají terminály a funkce z množiny symbolů.

Operátory selekce pracují u genetického programování stejně jako u genetického algoritmu. GP využívá obvykle jak křížení, tak mutaci. Základní operátor křížení kombinuje genetický materiál dvou rodičů vzájemnou výměnou části jednoho rodiče a části druhého. Po selekci rodičů je u obou náhodně označen jeden uzel a jemu příslušný podstrom. Tyto podstromy se následně mezi rodiči prohodí. Operátor mutace pracuje tak, že náhodně označí jeden uzel ve stromu a spolu s odpovídajícím podstromem tento uzel nahradí nově vygenerovaným stromem, který se generuje stejně jako stromy počáteční populace.

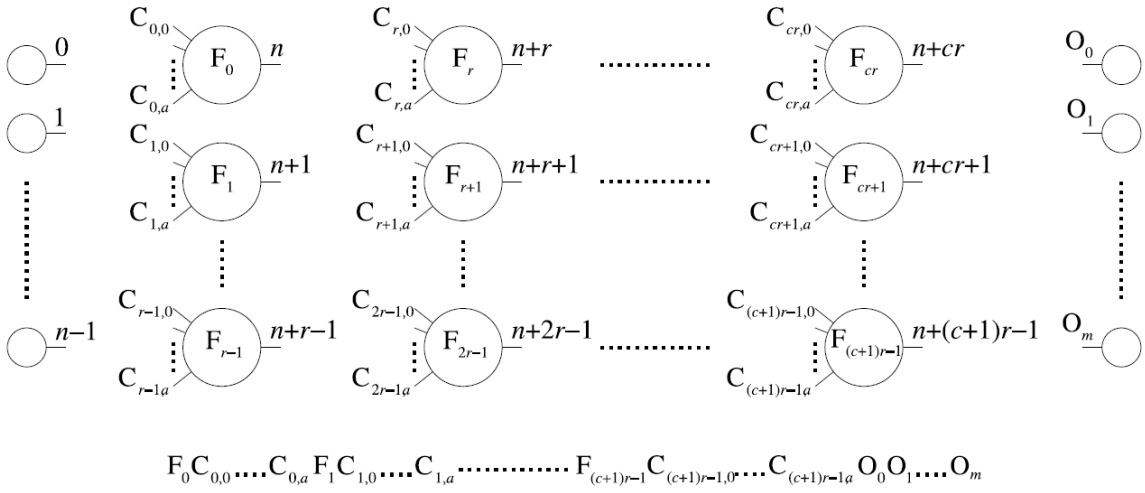
Fitness funkce závisí na dané aplikaci. V mnoha případech je použit přístup testování kandidátního programu pomocí trénovací množiny, která sestává z N vstupních hodnot. Výstupy y_j^{GP} získané kandidátním programem jsou porovnány s požadovanými hodnotami y_j a je vypočtena například průměrná odchylka mezi získanými a požadovanými hodnotami, která je chápána jako hodnota fitness:

$$\sum_{j=1}^{N}(y_j^{GP} - y_j)^2. \tag{2.3}$$

Cílem evoluce je minimalizovat tuto odchylku. Uvedený způsob výpočtu fitness je charakteristický pro tzv. symbolickou regresi, což je hlavní aplikace genetického programování. Cílem je najít program (matematický výraz), který co nejlépe aproximuje zadaná data v určitém intervalu [22].

2.6 Kartézské genetické programování

Podle [22] je kartézské genetické programování varianta genetického programování, u které jsou kandidátní řešení reprezentována pomocí obecných orientovaných grafů viz obrázek 2.1. V této práci se zaměříme na CGP pro evoluční návrh číslicových kombinačních obvodů.



Obrázek 2.1: Základní forma CGP z [17].

Kandidátní obvod je modelován jako pole programovatelných elementů o velikosti n_c (počet sloupců) $\times$ n_r (počet řádků). Počet primárních vstupů obvodu n_i i počet primárních výstupů obvodu n_o je pevně určen na začátku evoluce. Každý z uzlů realizuje právě jednu funkci s až n_n argumenty, která je vybrána z množiny dostupných funkcí Γ. Označme počet těchto funkcí $n_f = |\Gamma|$. Vstupy uzlu, který se nachází v i-tém sloupci, mohou být připojeny buď na primární vstupy obvodu nebo na výstupy uzlů umístěných až v L předchozích sloupcích. Uzly stejného sloupce se nesmějí propojovat. Parametr L (L-back) určuje míru propojitelnosti uzlů obvodu. Pro $L = 1$ je propojitelnost minimální, protože je možné propojovat pouze uzly sousedících sloupců. Pokud naopak zvolíme $L = n_c$ a $n_r = 1$, bude možné propojovat uzly libovolně (avšak bez zpětných propojení v případě kombinačních obvodů). Primární vstupy mohou být připojeny k libovolnému uzlu obvodu.

Chromozom popisující zapojení obvodu v CGP sestává z Λ_{CGP} celočíselných hodnot, kde

$$\Lambda_{CGP} = n_r n_c (n_n + 1) + n_o. \tag{2.4}$$

Způsob kódování je takový, že každému primárnímu vstupu obvodu je přiřazen index z intervalu $0, \ldots, n_i - 1$. K výstupům uzlů jsou postupně rovněž přiřazeny indexy, a to po sloupcích, s počáteční hodnotou n_i pro nejlevější horní uzel. Každý uzel je kódován pomocí $n_n + 1$ celočíselných hodnot. Prvních n_n hodnot určuje indexy uzlů, ke kterým budou připojeny vstupy uvažovaného uzlu. Poslední hodnota určuje funkci uzlu[2]. Na konci chromozomu najdeme n_o hodnot definujících indexy uzlů, ke kterým jsou připojeny primární výstupy obvodu. Základní vlastností uvedeného kódování je, že zatímco délka chromozomu je vždy konstantní, velikost zakódovaného obvodu (fenotypu) je variabilní. Vzniká tak redundance na několika úrovních:

[2]Případně naopak jako v [17], kde je na prvním místě funkce uzlu následovaná zapojením vstupů.

- Některé uzly nemusí být vůbec ve fenotypu použity, jiné mohou být použity vícenásobně.
- Některé vstupy uzlů nemusí být použity (například pokud dvouvstupový uzel pracuje jako invertor).
- Některé primární vstupy nemusí být ve fenotypu využity.

2.6.1 Dekódování genotypu a konstrukce fenotypu

Prvním krokem je určení, které uzly jsou aktivovány přímým propojením s výstupem. Poté jsou tyto uzly prozkoumány, aby se zjistilo, které další uzly jsou připojeny. Při dekódovacím procesu nejsou neaktivní uzly zpracovávány, takže jejich přítomnost v genotypu téměř nezvyšuje výpočetní náročnost. Jsou různé způsoby implementace tohoto procesu. Detailní příklad dekódovacího procesu je na obrázku 2.2 a příslušný fenotyp nalezneme na obrázku 2.3. [17]

2.6.2 Algoritmus prohledávání

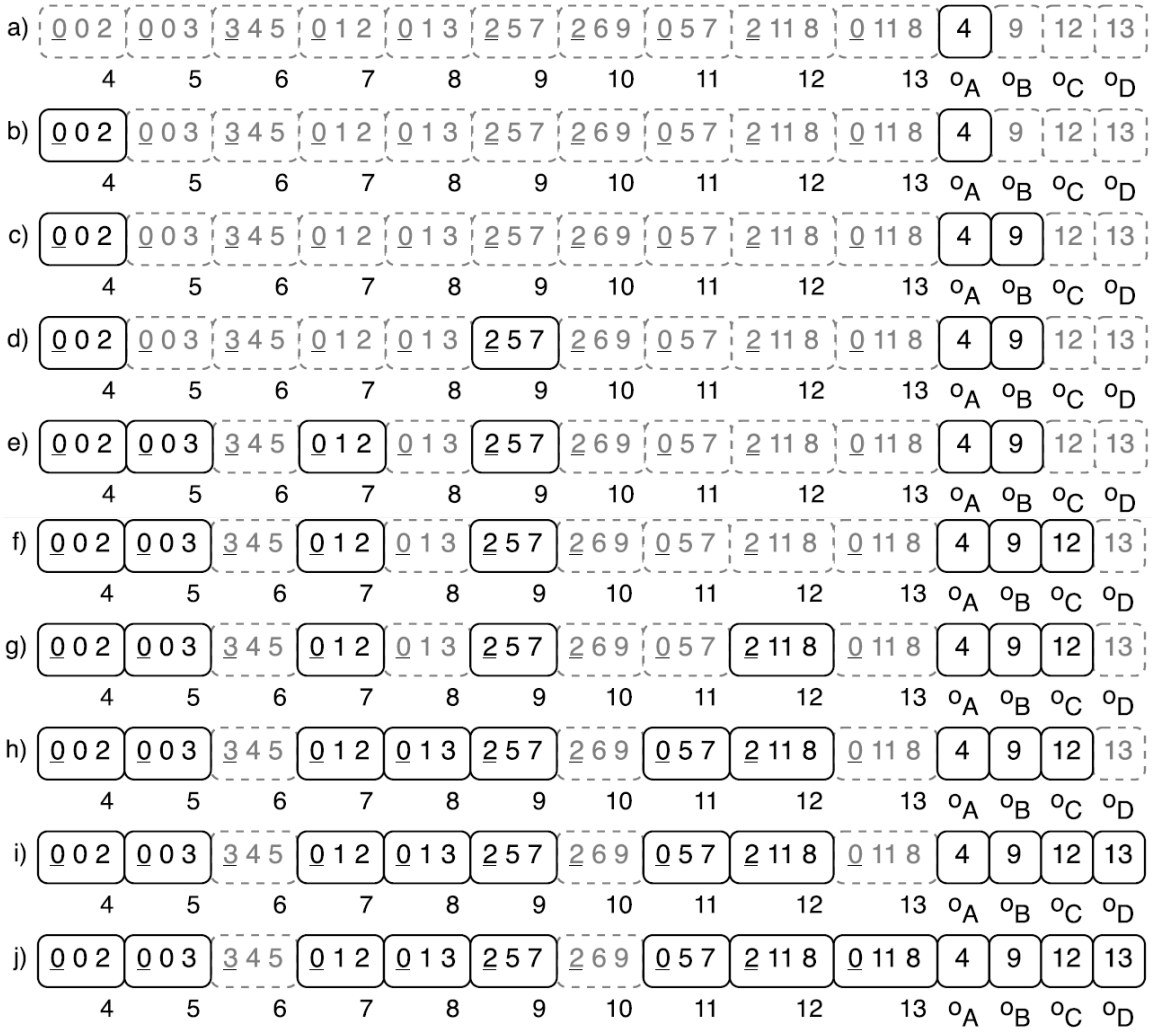
Pro CGP se často používá varianta evolučního algoritmu známá jako $1 + \lambda$. Většinou se hodnota λ volí 4. Taková varianta je znázorněna v algoritmu 2, který je převzat z [21].

Algoritmus 2: CGP

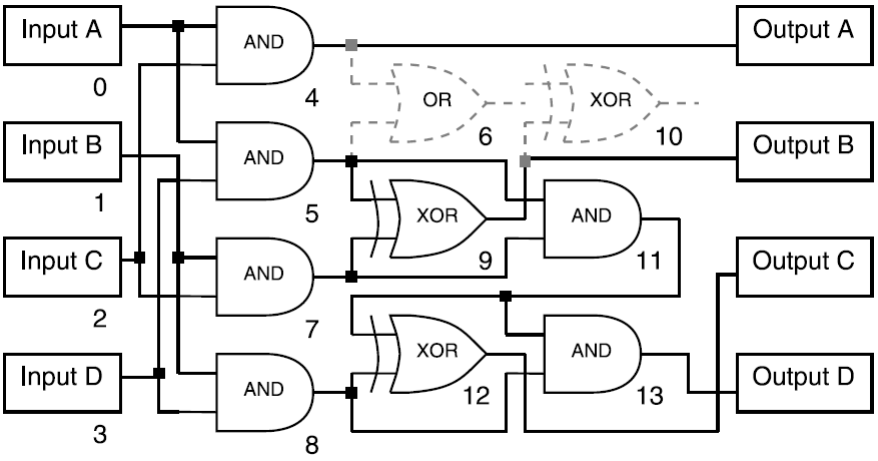
Input: Parametry CGP, fitness funkce
Output: Nejlépe ohodnocený jedinec p a jeho fitness

```
1:  P ← náhodně generuj rodiče p a λ jeho potomků;
2:  OhodnoťPopulaci(P);
3:  while není splněna ukončující podmínka do
4:      α ← nejlépe-ohodnocený-jedinec(P);
5:      if fitness(α) ≥ fitness(p) then
6:          p ← α;
7:      end
8:      P ← mutací vytvoř λ potomků z rodiče p;
9:      OhodnoťPopulaci(P);
10: end
11: return p, fitness(p);
```

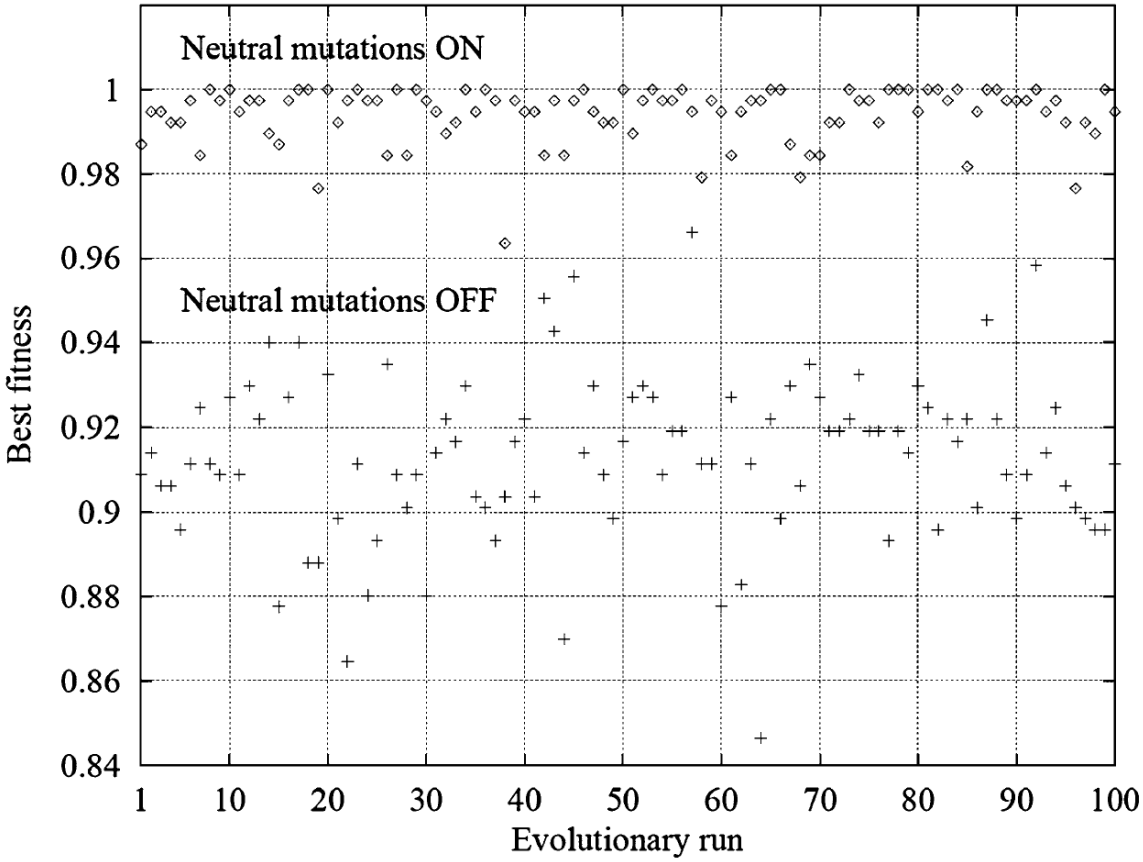
Na řádku 5 je podmínka, že v případě shody fitness potomka a rodiče a neexistence dalšího potomka s lepší hodnotou fitness než rodič bude jako nový rodič vybrán potomek. Uvedená podmínka je velmi důležitá vlastnost algoritmu, která zajišťuje větší diverzitu a potenciálně lepší maximální fitness populace, což je znázorněno na obrázku 2.4 [17].



Obrázek 2.2: Proces dekódování genotypu CGP pro dvou-bitovou násobičku. (a) Výstup A je připojen k uzlu 4; přesun na uzel 4. (b) Uzel 4 je připojen k primárním vstupům 0 a 2; výstup A je tudíž dekódován. Přesun na výstup B. (c) Výstup B je připojen k výstupu uzlu 9; přesun na uzel 9. (d) Uzel 9 je připojen k výstupům uzlů 5 a 7; přesun na uzly 5 a 7. (e) uzly 5 a 7 jsou připojeny na primární vstupy 0, 3, 1 a 2; výstup B je dekódován. Přesun na výstup C. Proces pokračuje dokud nejsou dekódovány výstupy C a D (postupně v krocích (f) až (h) a (i) až (j)). Když jsou dekódovány všechny výstupy, tak je dekódován celý genotyp. Převzato z [17].



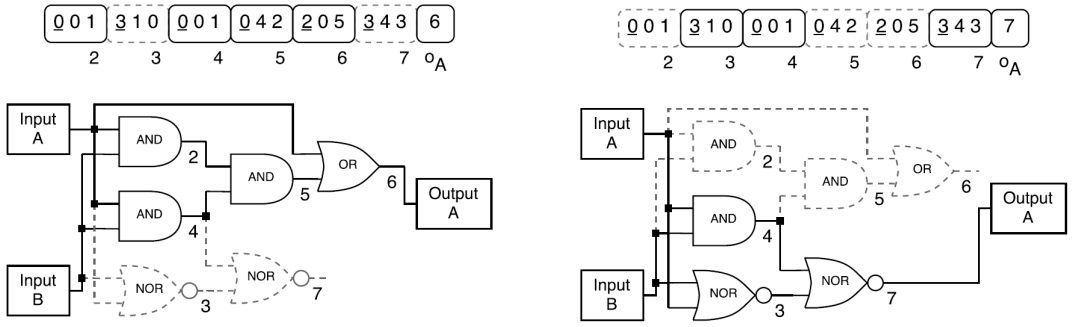
Obrázek 2.3: Fenotyp dvou-bitové násobičky. Převzato z [17].



Obrázek 2.4: Vliv neutrálních mutací [17].

2.6.3 Mutace v CGP

V CGP se používá bodový operátor mutace, kde je změněn náhodně vybraný gen na jinou, náhodnou, ale validní hodnotu. Pokud je k mutaci vybrán gen určující funkci, pak validní hodnotou je index v množině funkcí, čímž dojde ke změně funkce uzlu. Pokud je k mutaci vybrán vstupní gen, pak bude k uzlu připojen výstup jiného předcházejícího uzlu (omezeno parametrem L-back) nebo k primárnímu vstupu. K výstupu obvodu může být připojen výstup libovolného uzlu nebo primární vstup obvodu. Počet mutovaných genů je nastaven uživatelem a většinou se jedná o určité procento celkového počtu genů genotypu. Příklad bodové mutace je na obrázku 2.5, který ukazuje, jak může malá změna v genotypu způsobit velkou změnu fenotypu.



(a) Před mutací.

(b) Po mutaci.

Obrázek 2.5: Příklad bodové mutace v CGP aplikované na výstupní gen, který je změněn z 6 na 7. Toto způsobí aktivaci uzlů 3 a 7 a deaktivaci uzlů 2, 5 a 6. Neaktivní oblasti jsou znázorněny čárkovaně [17].

Křížení se v CGP většinou nepoužívá. Původně se používalo jednobodové křížení, ale zjistilo se, že přerušuje podgrafy v chromozomu a má škodlivý účinek na výkonnost CGP [17].

2.6.4 Redundance v genotypech CGP

Už jsme viděli, že v genotypu můžou být geny, které jsou úplně neaktivní, tudíž nemají vůbec vliv na fenotyp a fitness. Takové neaktivní geny jsou tedy neutrální k fitness genotypu. Tento jev je známý jako neutralita. Redundantní geny jsou v CGP dominantní. Podle článku [18] pracuje CGP nejlépe, když je většinou skoro celý genotyp neaktivní. Ukázalo se, že vliv neutrality v CGP je extrémně přínosný pro efektivitu evoluce u široké škály problémů [17].

2.6.5 Evoluce obvodů pomocí CGP

CGP může být použito pro evoluci obvodů, jejichž efektivní návrh pomocí konvenčních metod by byl velmi těžký nebo nemožný. Cílem evolučního návrhu obvodů je získat plně funkční obvod sestavený z množiny zadaných komponent. Každé kandidátní řešení je otestováno pro všechny možné vstupy. Jakmile je získán plně funkční obvod, tak by měl být optimalizován, to např. znamená minimalizaci počtu použitých hradel. V případě evoluce kombinačních obvodů je u CGP funkce fitness typicky definována takto:

$$fitness = \begin{cases} b & pokud \quad b < n_o \cdot 2^{n_i}, \\ b + (n_c \cdot n_r - z) & jinak, \end{cases} \tag{2.5}$$

kde b je počet správných výstupních bitů, získaných jako odezva pro všechny možné vstupy, z označuje počet použitých hradel v kandidátním obvodu, $n_c \cdot n_r$ je celkový počet dostupných elementů. Ze vzorce 2.5 je patrné, že výraz $n_c \cdot n_r - z$ je uvažován jen v případě, že chování kandidátního obvodu je bezchybné, což je když $b = n_o \cdot 2^{n_i}$. Můžeme pozorovat, že evoluce nejprve vytváří tlak na nalezení plně funkčního řešení bez ohledu na jeho velikost. Po objevení korektního řešení vyvíjí evoluce tlak na minimalizaci počtu hradel tohoto korektního obvodu. [12]

2.7 Záznam evoluce CGP

Pro vygenerování záznamu o průběhu evoluce jsem použil implementaci CGP ze sady Tools4CGP [24] (viz kapitola 3.2), která umožňuje vypisovat průběh evoluce. Bude popsán formát záznamu pro jeden kandidátní obvod z evoluce CGP. Data pro vstup vyvíjeného programu, který bude sloužit pro vizualizaci průběhu evoluce, budou právě tohoto formátu. Takových záznamů je však v průběhu evoluce vygenerováno velké množství – řádově minimálně miliony.

2.7.1 Formát záznamu průběhu evoluce

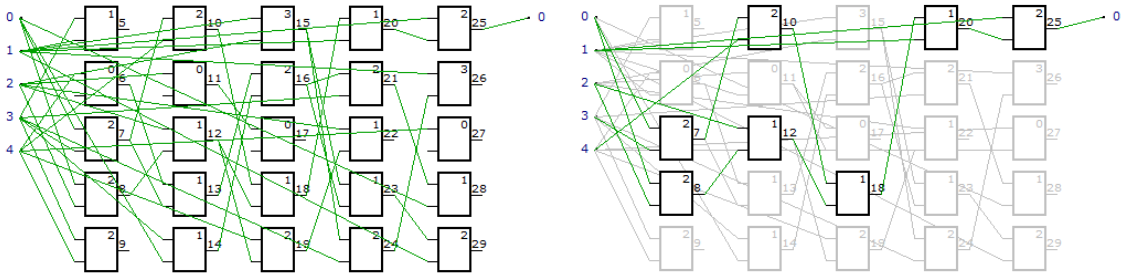
Následuje popis formátu záznamu evoluce obvodu pomocí CGP z příkladu 2.7.1.

Ve složených závorkách je uvedena postupně hodnota fitness a počet použitých bloků. Následují další složené závorky s těmito údaji: počet vstupů obvodu, počet výstupů obvodu, počet sloupců, počet řádků, počet vstupů hradla, L-back parametr, počet použitých bloků. Následuje popis jednotlivých hradel, kde popis každého hradla je uzavřen v závorkách – na prvním místě je v hranatých závorkách číslo výstupu bloku, za ním jsou čísla výstupů bloků, které jsou přivedeny na vstup tohoto bloku. Poslední číslo udává funkci bloku. Poslední část jednoho záznamu je definice výstupů, která je uzavřena v závorkách. Pro každý výstup je uvedeno číslo hradla nebo vstupu, ke kterému je výstup připojen. Jako oddělovač je použita čárka.

Jedinou změnou, která standardně není v tomto formátu přítomna, je uvedení maximální hodnoty fitness, nejlépe na začátku evoluce ve formátu „Maxfitness: hodnota“. Tento řádek není povinný, ale je nezbytný pro správné nalezení nejlepšího hradla po dosažení maximální hodnoty fitness.

Na obrázku 2.6a je zobrazen obvod z příkladu 2.7.1 a na obrázku 2.6b je tentýž obvod s nevyužitými hradly označenými šedou barvou.

Příklad 2.7.1. {22, 7}{5,1, 5,5, 2,1,7}([5]3,1,1)([6]4,1,0)([7]0,3,2)([8]0,2,2)([9]3,4,2)
([10]7,4,2)([11]2,1,0)([12]2,8,1)([13]6,7,1)([14]3,0,1)([15]1,2,3)([16]13,14,2)([17]11,11,0)
([18]10,12,1)([19]4,12,2)([20]18,1,1)([21]16,3,2)([22]2,19,1)([23]15,16,1)([24]15,3,2)
([25]1,20,2)([26]1,24,3)([27]4,4,0)([28]21,0,1)([29]23,1,2)(25)



(a) Obvod odpovídající celému genotypu. (b) Fenotyp.

Obrázek 2.6: Obvod z příkladu 2.7.1 zobrazený pomocí programu cgpviewer [24].

Kapitola 3

Vizualizace průběhu evoluce

Tato kapitola se zabývá technikami a nástroji pro vizualizaci a případně analýzu evoluce genetického programování a především kartézského genetického programování.

Evoluční algoritmy pracují na jednoduchém principu, ale produkují značné množství dat. Získání užitečných informací o vnitřním stavu a procesech uvnitř evolučního algoritmu není jednoduchý úkol.

3.1 Využití genealogické informace

Využití rodokmenu a genealogické informace je vhodné pro vývoj inovativních algoritmů genetického programování. Zde je přehled přístupů k tomuto tématu podle [6]:

- McPhee a Hopper [16] použili schéma číslování uzlů pro kvantifikaci relativního množství genetického materiálu z počáteční populace, která je přítomna na konci běhu algoritmu. Použili posledního společného předka všech jedinců v poslední generaci jako indikaci kvality běhu a také jako způsob měření diverzity podle množství sdíleného genetického materiálu mezi posledním společným předkem a zbytkem populace.

- Burke et al. [5] použili variantu turnajové selekce, která seskupuje jedince podle společné genetické linie pro zvýšení zachování diverzity během selekce. S využitím tzv. „rodokmenového výběru“ umožnili pouze jednomu jedinci u každé linie, aby se zúčastnil turnaje za účelem změnit výběr nejschopnějších na výběr nejschopnějších a nejrozmanitějších.

- Essam a Mckay [11] použili, pro větší diverzitu populace, systém značek k označení jedinců společného původu z důvodu zabránění jejich křížení. Jejich cílem bylo zabránit několika klíčovým jedincům a jejich potomkům v rapidním ovládnutí populace.

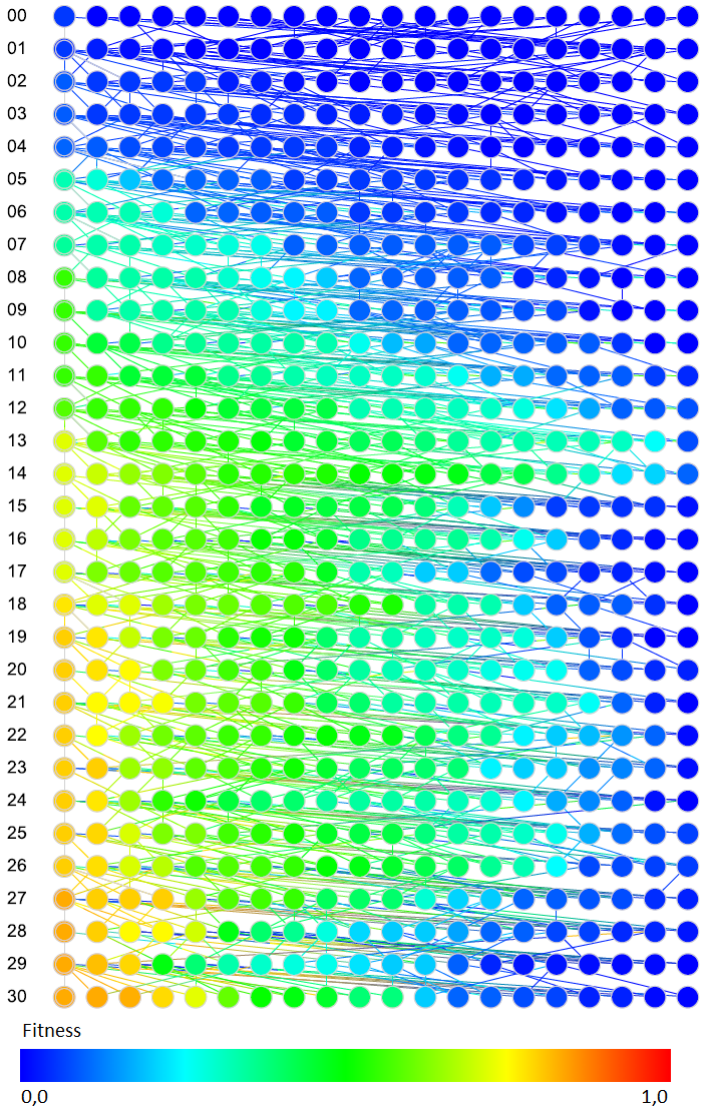
- Boetticher a Kaminksy [4] využili nedávnou historii (poslední generaci) k rozdělení jedinců do tříd podle fitness a směrovali selekci ke třídě s 20% nejlepších jedinců.

- Dong a Chen [7] použili větší historii (až čtyři předky) pro spočítání průměrné fitness dané linie před seskupením jedinců do tříd podle fitness pojmenovaných „nejlepší“, „střední“ a „nejhorší“.

- Podle Burlacu et al. [6] je využití pouze hodnoty fitness předků jedince nebo rodokmenu jedince nedostatečné a proto se rozhodli využít celou historii běhu gene-

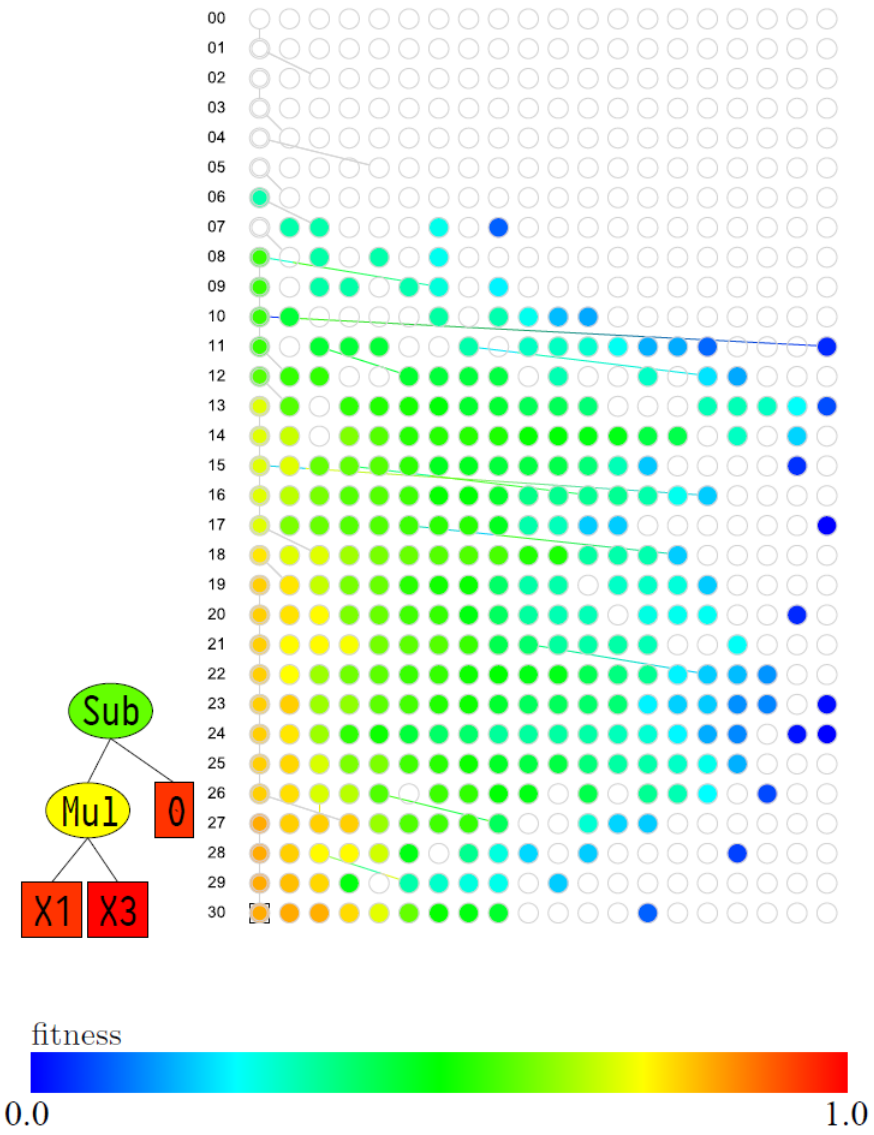
tického programování. Vytvořili zobrazovací nástroj umožňující zkoumat populace, genealogie a dědičnost genetického materiálu.

Obrázek 3.1 zobrazuje evoluci fitness všech jedinců populace z jednoho běhu genetického algoritmu. Vrcholy jsou zbarveny na základě fitness a každá hrana grafu je obarvena barevným přechodem, korespondujícím s barvou obou vrcholů, které spojuje. Vrstvy grafu reprezentují jednotlivé generace. Tyto vrstvy jsou označeny číslem generace, kde 0 reprezentuje počáteční populaci, 1 první generaci atd. Z grafu lze vyčíst např. distribuci hodnot fitness v každé generaci samostatně nebo v celém průběhu evoluce. U úspěšných běhů můžeme pozorovat zlepšování průměrné fitness ve většině dalších generací, dokud nedojde ke konvergenci. Dále můžeme identifikovat intervaly, ve kterých došlo ke stagnaci evoluce (generace 8-11, 13-17, 19-25 a 27-30) – v těchto generacích, kde nedošlo ke zlepšení, je zvýšená pravděpodobnost, že nejlepší jedinec začne převládat v populaci, protože bude častěji vybírán. Zvýší se sice průměrná fitness generace, ale sníží se tak diverzita populace.



Obrázek 3.1: Distribuce hodnot fitness.

Na obrázku 3.2 je zobrazeno jiné využití stejné zobrazovací metody. Tento graf slouží ke sledování šíření genetické informace. Jsou vyznačeni jedinci obsahující daný podgraf a barevně je označena jejich fitness. Takto je možné sledovat vznik zvolených komponent a jejich vliv na proces evoluce [6].

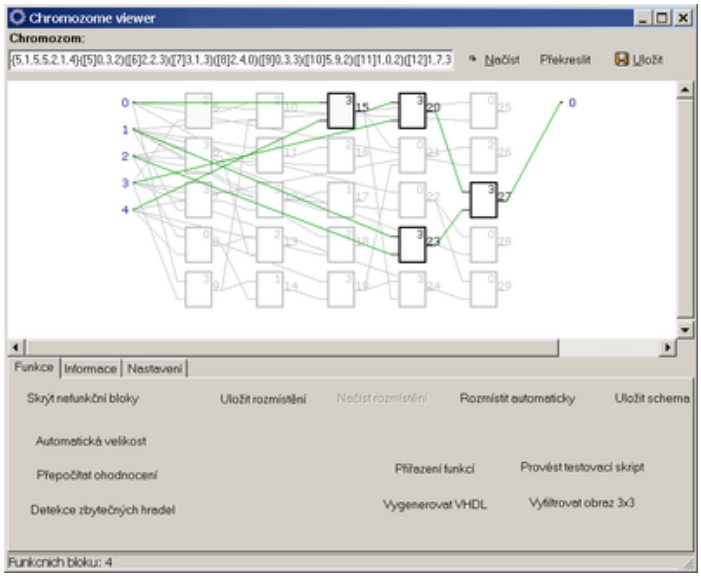


Obrázek 3.2: Jedinci obsahující daný podstrom.

3.2 Nástroje pro vizualizaci CGP

Při analýze informací o průběhu evoluce je nutné, aby si nástroje poradily se zpracováním velkého množství těchto dat v „rozumném“ čase. Také musí být zohledněno omezení zobrazovacích zařízení – především rozlišení monitorů a tedy množství informací, které je možno zobrazit. Proto je u praktických problémů nutná filtrace a zhuštění zobrazované informace, případně několik úrovní detailů zobrazení. Následuje přehled aplikací umožňujících práci případně základní analýzu CGP.

- Tools4CGP [24] je sada nástrojů, kterou na FIT VUT vytvořili Zdeněk Vašíček a Lukáš Sekanina pro práci s kartézským genetickým programováním. Tato sada nástrojů obsahuje implementaci CGP, modul pro načítání vstupních dat a prohlížeč chromozomů (cgpviewer). Tools4CGP jsou vytvořeny v jazyce C. Prohlížeč chromozomů cgpviewer (viz obr. 3.3) umožňuje zobrazit chromozom CGP, simulovat ho a exportovat do různých formátů (VHDL, bmp).



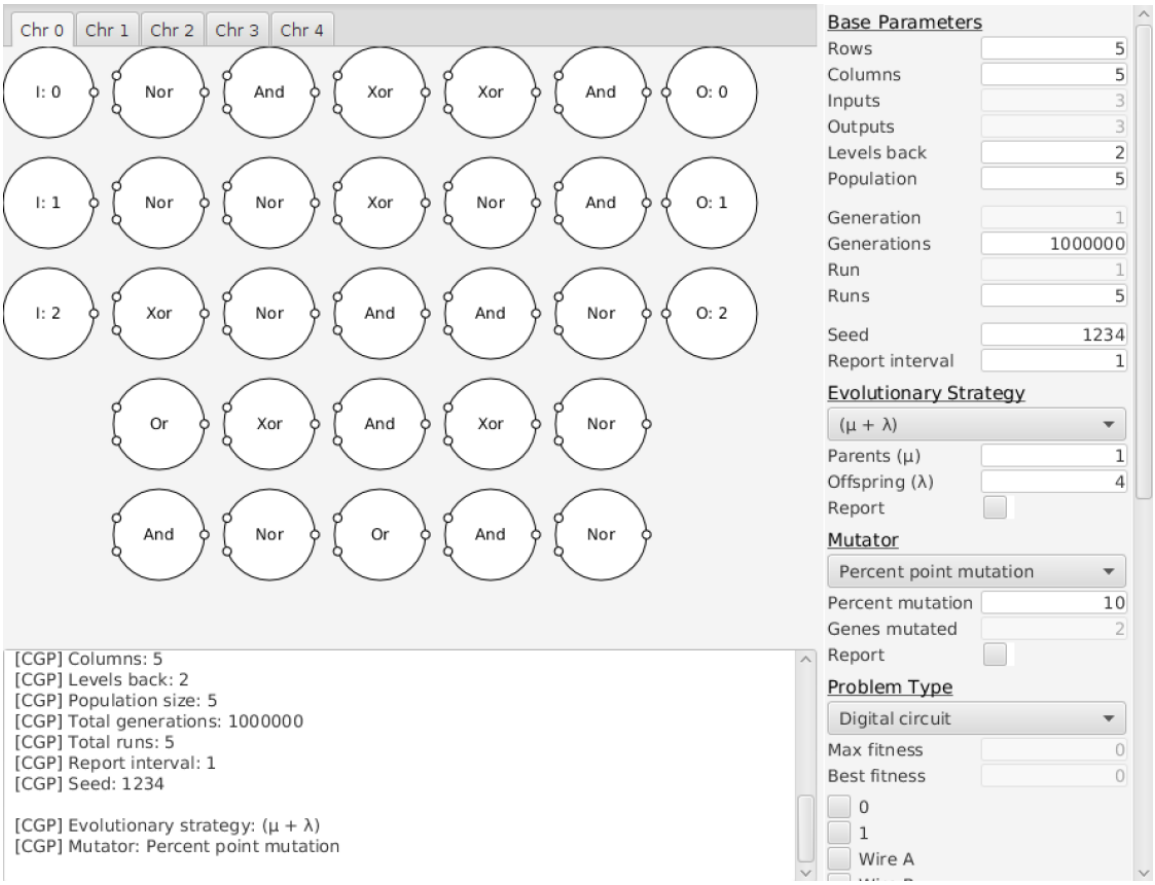
Obrázek 3.3: Zdeněk Vašíček – cgpviewer.

Mezi další funkce patří: zašednutí nefunkčních bloků, automatické rozmístění hradel podle zpoždění (zobrazí se jen funkční hradla), přiřazení funkcí jednotlivým blokům a zobrazení příslušné schématické značky (AND, OR, XOR...).

- V roce 2014 vytvořil Eduardo Pedroni, v jazyce Java v rámci své diplomové práce, nástroj pro vizualizaci CGP [20]. Jak můžeme vidět na obrázku 3.4, nástroj graficky zobrazuje jednotlivé chromozomy populace, které jsou vybírány ze záložek pojmenovaných „Chr x“, kde „x“ je číslo chromozomu. V rámci chromozomu nástroj rozlišuje aktivní a neaktivní uzly. Při kliknutí myší na výstup jsou zvýrazněny využité uzly a spoje.

  Grafické uživatelské rozhraní umožňuje nastavení široké škály základních parametrů CGP např. počet řádků, sloupců, vstupů a výstupů, L-back parametr, velikost populace, počet generací a běhů, seed pro pseudonáhodný generátor, četnost výpisů do konzole. Dále je možno nastavit evoluční strategii, počet rodičů a potomků, četnost mutace a počet mutovaných genů. Tlačítky Run, Step a Reset lze kontrolovat běh algoritmu.

- CGPevolutionViewer [23] je jediný nalezený nástroj, který je určený pro analýzu průběhu evoluce. Vznikl jako součást magisterské práce Jany Staurovské v roce 2012. Program umožňuje načtení záznamu evoluce, případně i souboru s definicemi funkcí bloků. Po načtení záznamu evoluce se zobrazí několik generací (omezením je velikost zobrazovací plochy), viz obrázek 3.5. Je možné se posouvat dopředu nebo dozadu o zvolený počet generací nebo metodou „drag and drop“.

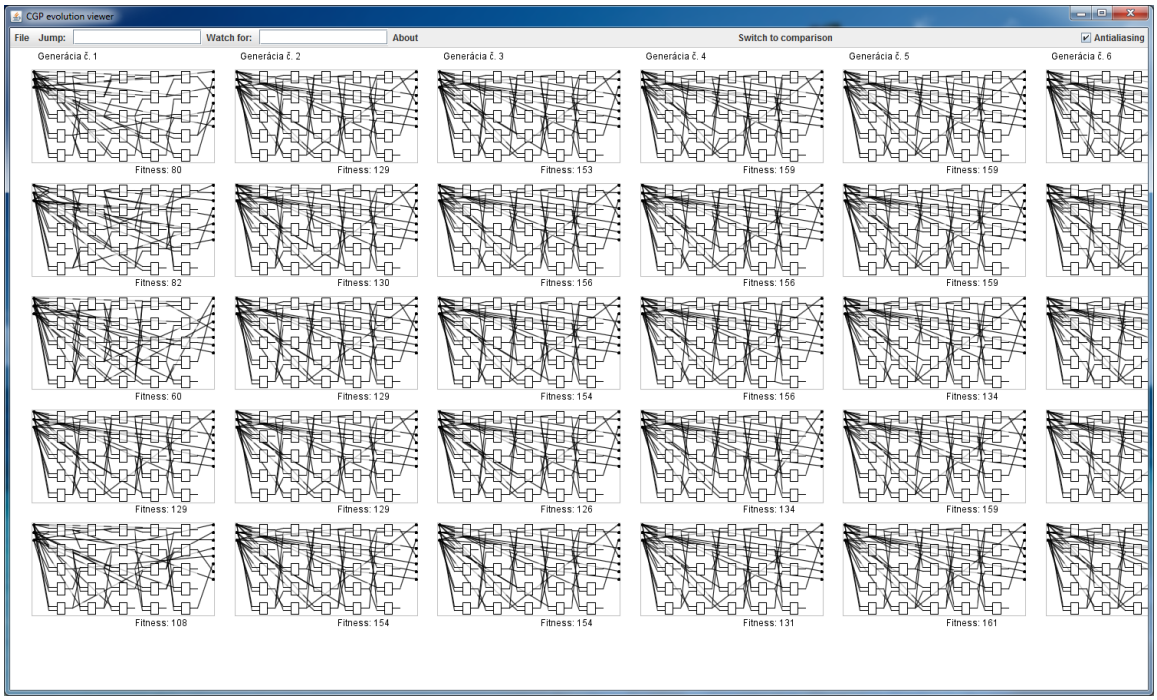


Obrázek 3.4: Eduardo Pedroni – CGP GUI.

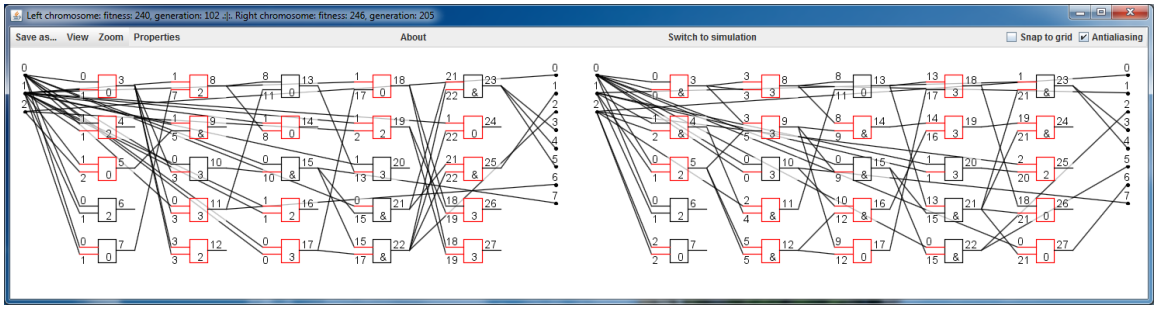
Pro analýzu jednotlivých obvodů jsou implementovány funkce: zašednutí a odstranění nepoužitých hradel, uspořádání hradel do mřížky, historie – barevné vyznačení stáří hradel a při najetí myší na hradlo se zobrazí číslo generace, ve které se udála poslední změna v tomto hradle, zobrazení podgrafu, který souvisí se zvoleným hradlem (ostatní hradla zašednou), vycentrování chromozomu, zobrazení všech hradel, přiblížení, oddálení, reset zobrazení. V módu simulace je možné klikáním na primární vstupy simulovat obvod.

Dále je implementována funkcionalita pro porovnávání dvou zvolených obvodů (viz obr. 3.6), kde jsou v novém okně zobrazeny oba vybrané obvody a barevně jsou zvýrazněny jejich rozdíly a také jsou zobrazeny informace o generacích obvodů a o jejich fitness.

Žádný z nalezených nástrojů neposkytuje dostatečnou podporu pro analýzu průběhu evoluce. Program CGPevolutionViewer [23], který je jako jediný určený přímo k tomuto účelu, nenabízí dostatečnou paletu nástrojů pro analýzu evoluce jako celku ani pro výběr a podrobnou analýzu částí evoluce.



Obrázek 3.5: Jana Staurovská – CGPevolutionViewer.



Obrázek 3.6: Jana Staurovská – CGPevolutionViewer – porovnání dvou obvodů.

Kapitola 4

Specifikace vyvíjeného nástroje

Cílem vyvíjeného nástroje je umožnit jak analýzu záznamu průběhu evoluce obvodů jako celku, tak zkoumání částí evoluce. Nástroj tedy musí poskytovat funkce pro zobrazení informací o celém průběhu evoluce. Zkoumáním částí evoluce se rozumí možnost sledovat, analyzovat a porovnávat jednotlivé kandidátní obvody a jejich vlastnosti jako např. fitness, pravdivostní tabulky, použité komponenty atd. Aplikace musí zohlednit a efektivně zpracovat velké množství dat, které záznamy o průběhu evoluce typicky obsahují. Je třeba také počítat s omezeným výpočetním výkonem a omezenou zobrazovací plochou monitoru. Získané informace budou sloužit k hlubšímu pochopení procesů uvnitř evoluce za účelem zrychlení a zefektivnění evoluce obvodů.

Možné (uvažované) funkce pro analýzu průběhu evoluce:

- simulace obvodu a podobvodu
- pravdivostní tabulka obvodu a podobvodu
- zašednutí nebo skrytí nevyužitých hradel
- manuální prohlídka průběhu evoluce – navigace např. grafem vývoje fitness
- porovnání dvou a více obvodů (rozdíly zapojení, fitness, shodné podobvody, srovnání pravdivostních tabulek)
- stáří komponent – barevné nebo jiné znázornění
- vývojový strom
- identifikace velkých změn fitness
- identifikace velkých změn počtu využitých bloků
- automatická identifikace zajímavých obvodů nebo podobvodů, které způsobily nalezené změny fitness atd.
- nalezení opakujících se konstrukcí v evoluci (i neužitečných, které by bylo vhodné negenerovat)
- nalezení částí, které se málo nebo vůbec nemění po dlouhou dobu
- analýza změn diverzity populace a diverzity fitness

- analýza úseků evoluce, kde se fitness mění velmi málo nebo vůbec – např. pro zjištění, jestli se vyvíjí neaktivní podobvod, který je později aktivován
- statistika využití hradel

Pro vyvíjený nástroj jsem se rozhodl implementovat následující funkce:

- Načtení záznamu evoluce. Zde je třeba buď podporovat různé typy záznamů nebo vytvořit detailní popis podporovaného formátu.
- Zobrazení jednoho i více obvodů najednou s případnými dalšími funkcemi jako navigace pomocí posuvníku nebo systémem „drag and drop“, zoom, zašednutí nebo skrytí nevyužitých hradel atd.
- Zobrazení měnící se fitness v průběhu evoluce pomocí grafu s možností navigace evolucí pomocí tohoto grafu. Bude nabídnuta možnost zobrazit a skrýt úseky, kde se nemění fitness z důvodu kompaktnosti grafu, čili budou zobrazeny jen změny fitness. Při navigaci evolucí se budou zobrazovat příslušné kandidátní obvody, které bude možno vybrat pro další analýzu.
- Další možností navigace evolucí bude barevný přehled vývoje fitness všech jedinců všech generací, tak že např. vertikálně budou znázorněni jedinci jedné generace a v horizontálním směru budou další generace. Z tohoto zobrazení bude také možno vybrat jedince pro další analýzu.
- Porovnání dvou vybraných obvodů – vyznačení rozdílů mezi obvody a možnost změny vybraných obvodů, například pomocí šipek směřujících na předchozí a následující obvody v evoluci.
- Zobrazení pravdivostní tabulky podobvodu po zvolení příslušného podobvodu např. kliknutím na spoj mezi hradly. Zde je nutno implementovat načítání nebo nastavení funkce (pravdivostních tabulek) jednotlivých hradel.
- Vyznačení stáří komponent.
- Statistika využití hradel (počet jednotlivých členů).

Kvůli povaze navržených funkcí je nástroj určen pro relativně malé kombinační obvody. Hradla obvodu mohou obsahovat pouze dva vstupy, výstup a provádí jednu z šestnácti logických funkcí.

Kapitola 5

Implementace

Pro implementaci byl zvolen jazyk Java z několika důvodů: platformová nezávislost a přenositelnost jak na úrovni zdrojového kódu, tak na úrovni přeloženého kódu (program vytvořený na jedné platformě může běžet prakticky na jakékoliv jiné). Dalším důležitým faktorem pro výběr jazyka Java byla možnost využití předešlé práce na toto téma a to programu CGPevolutionViewer Jany Staurovské [23] napsaného v jazyku Java, ze kterého by bylo možné využít především funkční jádro pro vykreslování obvodů (ve výsledné aplikaci je použita jen kostra pro načítání záznamu o průběhu evoluce a souboru s funkcemi hradel). Jazyk Java poskytuje široké spektrum knihoven a rozšíření pro práci s grafickým uživatelským rozhraním a se zobrazováním grafů, což bude pro tuto práci velmi užitečné. GUI bude implementována s využitím knihovny Swing.

Java je objektově orientovaný programovací jazyk (OOJ) pro všeobecné použití, založený na třídách. Umožňuje nám tedy využívat výhodné vlastnosti OOJ jako abstrakce, zapouzdření, polymorfizmus, dědičnost a kompozice.

5.1 Obecné principy

5.1.1 Abstrakce

Abstrakce je zjednodušení či zanedbání některých vlastností objektů z reálného světa, se kterými program pracuje. Velmi podstatná je zvolená míra abstrakce, protože čím více se přibližujeme k reálnému objektu, tím složitější je pak jeho model [8].

Objektově orientované programování ukázalo, že vhodným zvýšením abstrakce můžeme rychle navrhovat a vyvíjet větší a komplikovanější projekty [19].

5.1.2 Zapouzdření

Jednou ze základních a velice ceněných vlastností objektově orientovaných programů je schopnost tzv. zapouzdření. Zajišťuje, že s objekty můžeme komunikovat pouze skrze jejich rozhraní tzn. můžeme používat pouze explicitně povolené operace a nelze tak měnit stav objektů neočekávaným způsobem. Pro ostatní objekty zůstává vnitřní implementace daného objektu skryta. Čím jsou programy složitější, tím je tento koncept důležitější, abychom vyloučili možnost nechtěného ovlivnění chodu některé jiné části programu [19].

5.1.3 Polymorfizmus

Polymorfizmus neboli mnohotvárnost je další vlastností OOJ. Poskytuje další rozměr pro oddělení rozhraní objektu od implementace. Klíčová je možnost zacházet s objektem jednou jako s instancí jeho vlastního typu, jindy jako s typy bázové třídy. Tato vlastnost umožňuje programátorům zacházet s mnoha typy objektů, které jsou odvozeny od stejné bázové třídy tak, jako kdyby se jednalo o jeden typ. Polymorfizmus je mocný nástroj, který umožňuje efektivnější vytváření snadněji rozšiřitelných programů s lepší organizací a udržovatelností zdrojového kódu [8].

5.1.4 Dědičnost

Dědičnost je způsob jak zajistit opakovanou využitelnost kódu. Je-li dědičnost nevhodně používána, vzniká „křehký“ software. Je bezpečné používat dědičnost v rámci nějakého balíčku, kde je implementace podtříd a nadtříd pod kontrolou stejných programátorů. Je rovněž bezpečné používat dědičnost při rozšiřování tříd, které jsou pro takové rozšíření speciálně navrženy a zdokumentovány. Dědění z běžných konkrétních tříd mezi hranicemi balíčků je však nebezpečné.

Na rozdíl od volání metod narušuje dědičnost zapouzdření. Jinými slovy, podtřída závisí svým řádným fungováním na implementačních detailech své nadtřídy. Implementace této nadtřídy se může v další verzi změnit a dojde-li k tomu, může dojít ke zhroucení podtřídy, třebaže se její kód vůbec nezměnil.

Celkově lze říci, že dědičnost je účinná, ale také problematická, protože narušuje zapouzdření. Je vhodná pouze v situaci, kdy mezi třídou a příslušnou nadtřídou existuje opravdový vztah podtypů [3].

5.1.5 Kompozice

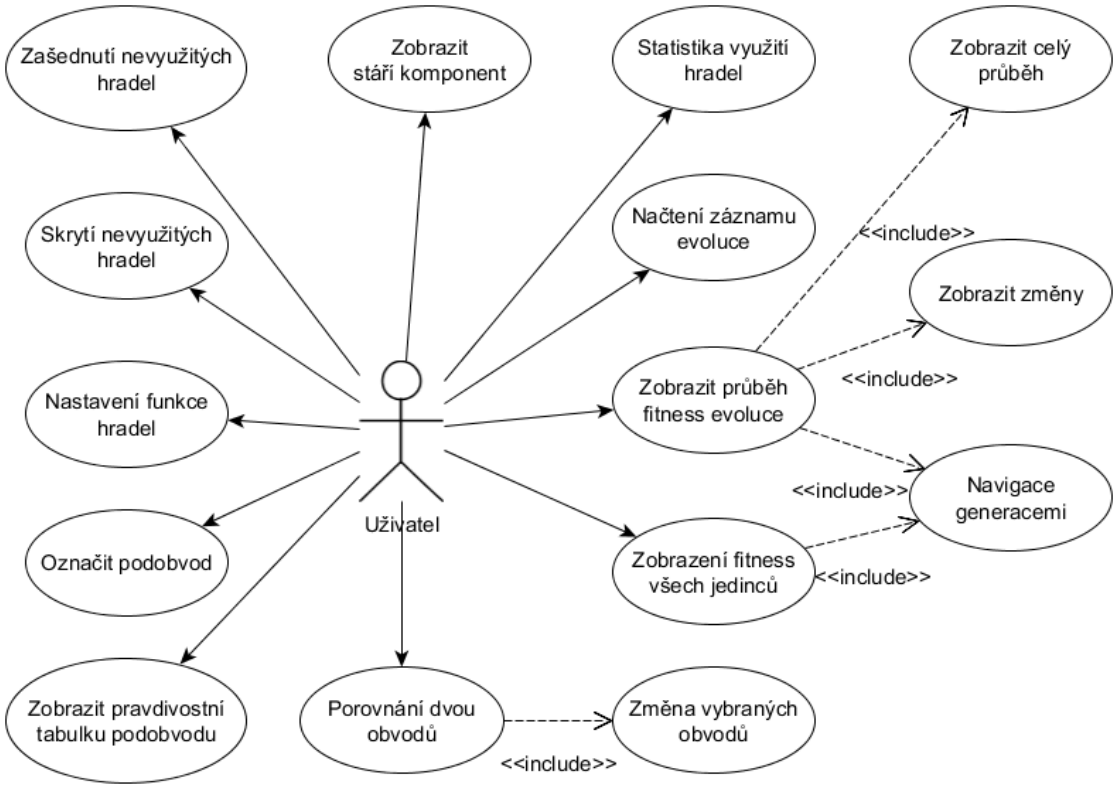
Způsob, jak se vyhnout problémům dědičnosti, je kompozice neboli skládání. Místo rozšiřování třídy je jí přidán atribut, který bude odkazovat na nějakou instanci existující třídy. Třída se tak stává komponentou jiné třídy. Výsledná třída bude stabilní a nezávislá na implementačních detailech existující třídy. Kompozice je tedy celkově „mocnější“ než dědičnost [3].

5.1.6 Grafické uživatelské rozhraní

Jak již bylo řečeno, GUI bude implementováno s využitím knihovny Swing. Jak je uvedeno v [25], knihovna Swing umožňuje multi-vláknovým aplikacím využívat tyto tři typy vláken:

- „Initial threads“ jsou vlákna spouštějící inicializační kód aplikace.
- „Event dispatch thread“ je speciální vlákno, kde běží kód řízený událostmi.
- „Worker threads“ jsou vlákna sloužící pro provádění časově náročných operací.

Při implementaci GUI bude brán zřetel na doporučení pro vytváření grafických uživatelských rozhraní z [2].



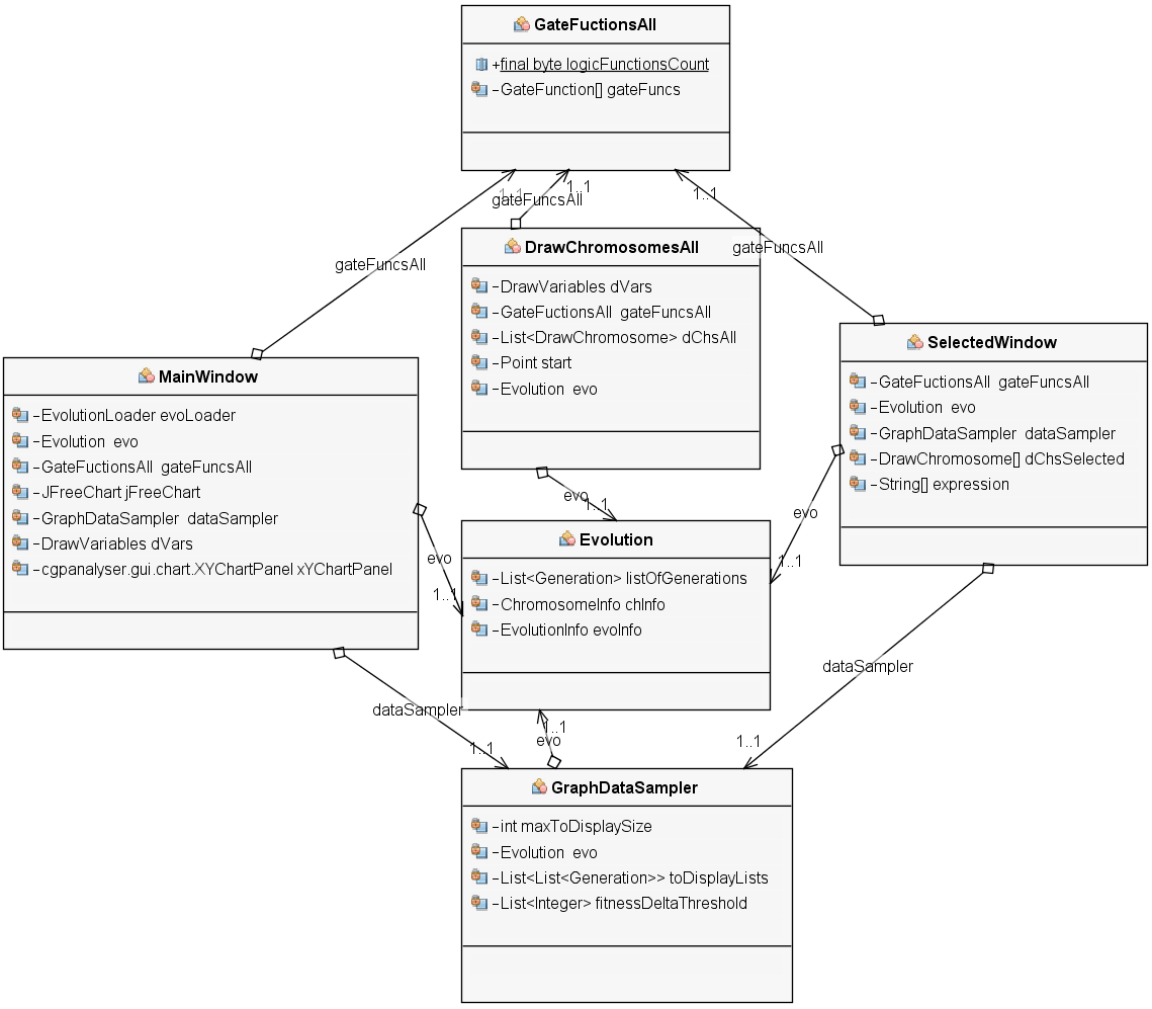
Obrázek 5.1: Diagram případů užití.

5.2 Diagram případů užití

Na obrázku 5.1 je diagram případů užití vyvíjeného nástroje, který vychází ze specifikace funkcí aplikace v kapitole 4.

5.3 Diagram tříd

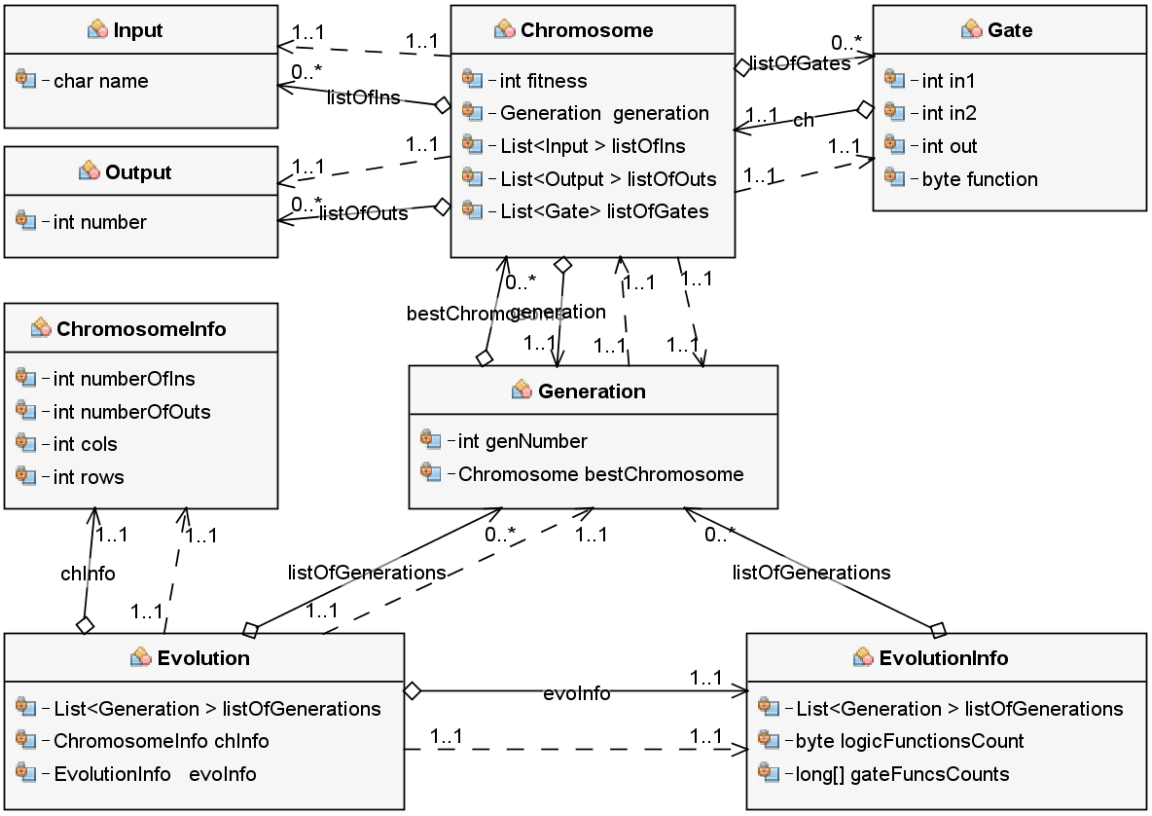
Na obrázku 5.2 je zjednodušený, obecný diagram tříd, který zachycuje základní komponenty aplikace. Základní třídou je třída `Evolution`, která uchovává informace získané ze záznamu o průběhu evoluce – především seznam generací, jedince a jejich komponenty. `Evolution` je navržena s ohledem na možnou vysokou paměťovou náročnost záznamu o průběhu evoluce a neobsahuje proto žádné dodatečné informace, které nejsou pořízeny ze samotného záznamu. Pro uložení dat, sloužících k vykreslování evoluce, byla vytvořena třída `DrawChromosomesAll` a příslušné podtřídy, které jsou grafickými ekvivalenty jednotlivým podtřídám třídy `Evolution`. Po načtení záznamu jsou navzorkovány nejzajímavější úseky evoluce pomocí třídy `GraphDataSampler` a zobrazeny ve formě grafu v hlavním okně – `MainWindow`. Kromě záznamu o evoluci umožňuje aplikace načíst i tzv. „function file“, což je soubor s informacemi o funkci jednotlivých hradel. Tyto informace spravuje třída `GateFuctionsAll`. Pro bližší analýzu chromozomů slouží okno reprezentované třídou `SelectedWindow`.



Obrázek 5.2: Obecný diagram tříd.

Diagram tříd 5.3 zobrazuje strukturu záznamu evoluce. Hierarchie tříd vychází z logické struktury, kdy třída `Evolution` obsahuje seznam generací (`List<Generation>`). Třída

`Generation` je tvořena číslem generace a nejlepším chromozomem (chromozomem, který bude použit jako rodič pro vytvoření další generace). Chromozom, reprezentovaný třídou `Chromosome`, je pak tvořen hodnotou fitness, zpětným odkazem na mateřskou třídu `Generation` a třemi seznamy: `List<Input>`, `List<Output>`, `List<Gate>`, které obsahují vstupy, výstupy a hradla chromozomu. Třídy `Input` a `Output` obsahují jen jméno respektive číslo zapojení. Třída `Gate` uchovává informace o zapojení hradla a jeho funkci.



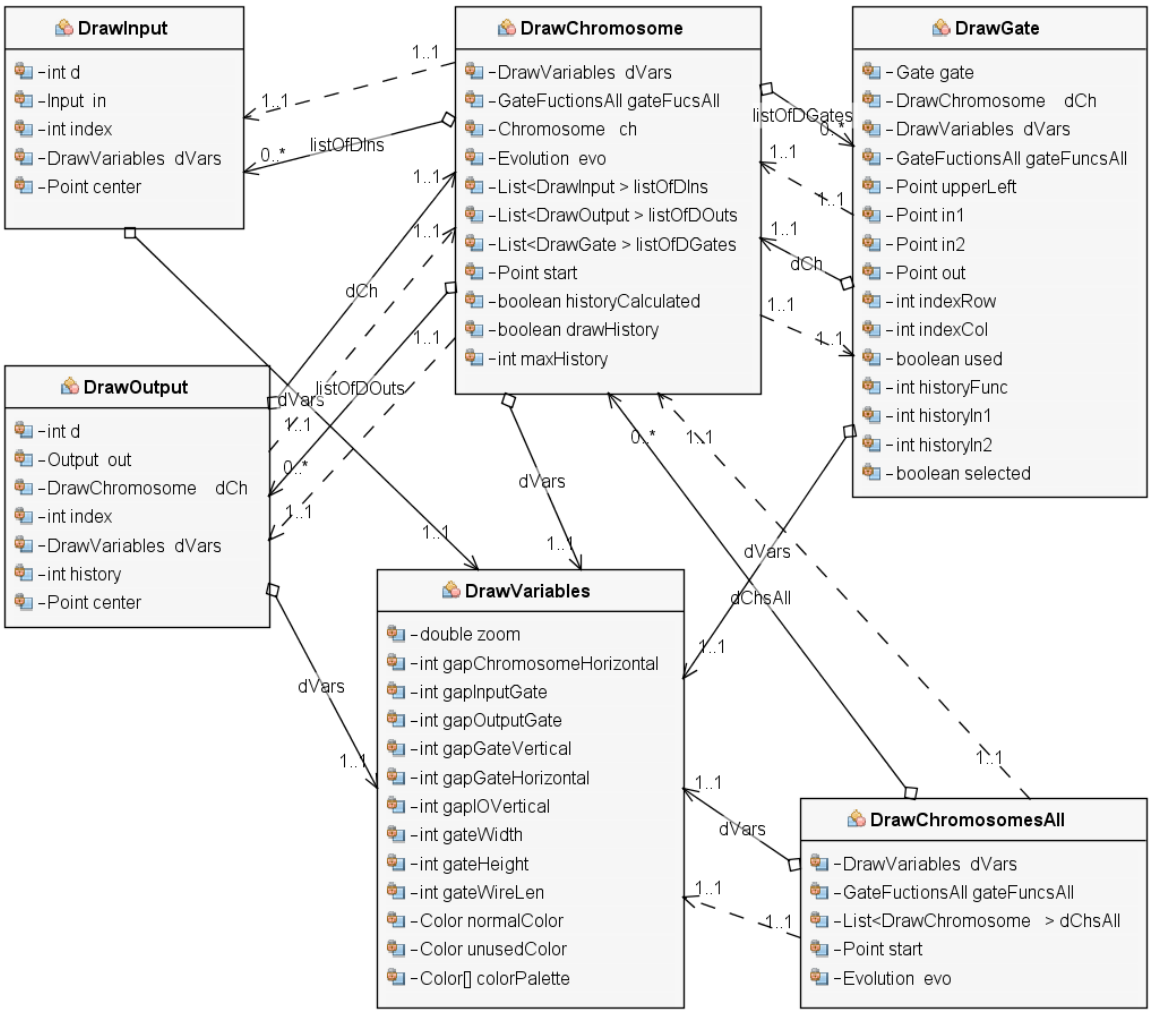
Obrázek 5.3: Diagram tříd záznamu evoluce.

Tato struktura tříd slouží pro uložení celého záznamu evoluce. Protože takový záznam může být velmi dlouhý (řádově i stovky milionů generací), je hierarchie tříd vytvořena velmi kompaktně, aby zabírala co nejméně paměti (například seznam vstupů `List<Input>` je v paměti uložen jen jednou a třída `Chromosome` na něj obsahuje pouze odkaz). Pro uchování dodatečných informací o chromozomu, jako je počet řádků, sloupců, vstupů a výstupů, slouží třída `ChromosomeInfo`. Dodatečné informace o celé evoluci uchovává třída `EvolutionInfo`.

Po načtení záznamu průběhu evoluce a jeho uložení do popsané hierarchie tříd přichází na řadu vizualizace. Protože však není možné zobrazit celou evoluci, předchází samotnému zobrazování proces vzorkování záznamu – konkrétně výběr zajímavých úseků evoluce. Tento úkol zajišťuje třída `GraphDataSampler`, kterou můžeme nalézt na obrázku obrázku 5.5.

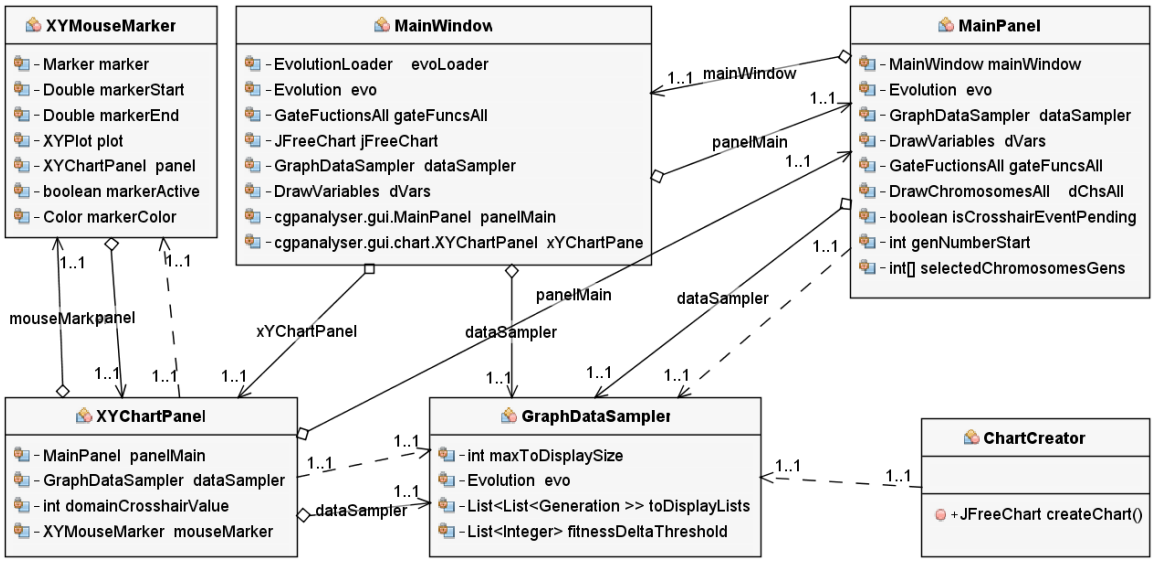
K zobrazení navzorkovaných dat slouží sada tzv. „draw“ tříd, které jsou na obrázku 5.2 reprezentovány třídou `DrawChromosomesAll`. Na obrázku 5.4 nalezneme konkrétnější strukturu tříd, které slouží pro vykreslování. Protože hierarchie tříd pro uložení záznamu evoluce musí být co nejméně paměťově náročná, byly k jednotlivým třídám z diagramu 5.3 vytvořeny tzv. „obalující“ třídy, které uchovávají data a metody sloužící pro vykreslo-

vání. Ke třídám `Input`, `Output`, `Gate`, `Chromosome` a `Evolution` tak existují grafické třídy `DrawInput`, `DrawOutput`, `DrawGate`, `DrawChromosome` a `DrawChromosomesAll`. Tyto třídy jsou vytvářeny a upravovány podle potřeby za běhu a v každém okamžiku jich existuje jen nezbytně nutné množství. Třída `DrawVariables` uchovává konstanty, použité při vykreslování, jako jsou základní rozměry hradel, zoom, mezery mezi elementy chromozomu atd.



Obrázek 5.4: Diagram tříd pro vykreslování.

Obrázek 5.5 zachycuje strukturu tříd sloužících pro vzorkování evoluce (již zmíněná třída `GraphDataSampler`), jejíž výstup využije třída `ChartCreator`, která vytvoří přehledný, interaktivní graf, který je zobrazen v hlavním okně (`MainWindow`) pomocí třídy `MainPanel`.



Obrázek 5.5: Diagram tříd pro vzorkování a hlavní okno.

Kapitola 6

Aplikace CGPanalyser

V této kapitole je popsána vytvořená aplikace CGPanalyser, její funkce, možnosti, omezení a některé implementační detaily. Aplikace je implementována v jazyce Java verze 8 s využitím knihovny Swing a vzhled grafických komponent, neboli „look and feel“, je nastaven na výchozí vzhled daného operačního systému. Obrázky použité v této kapitole jsou pořízeny v operačním systému Windows 7.

6.1 Hlavní okno

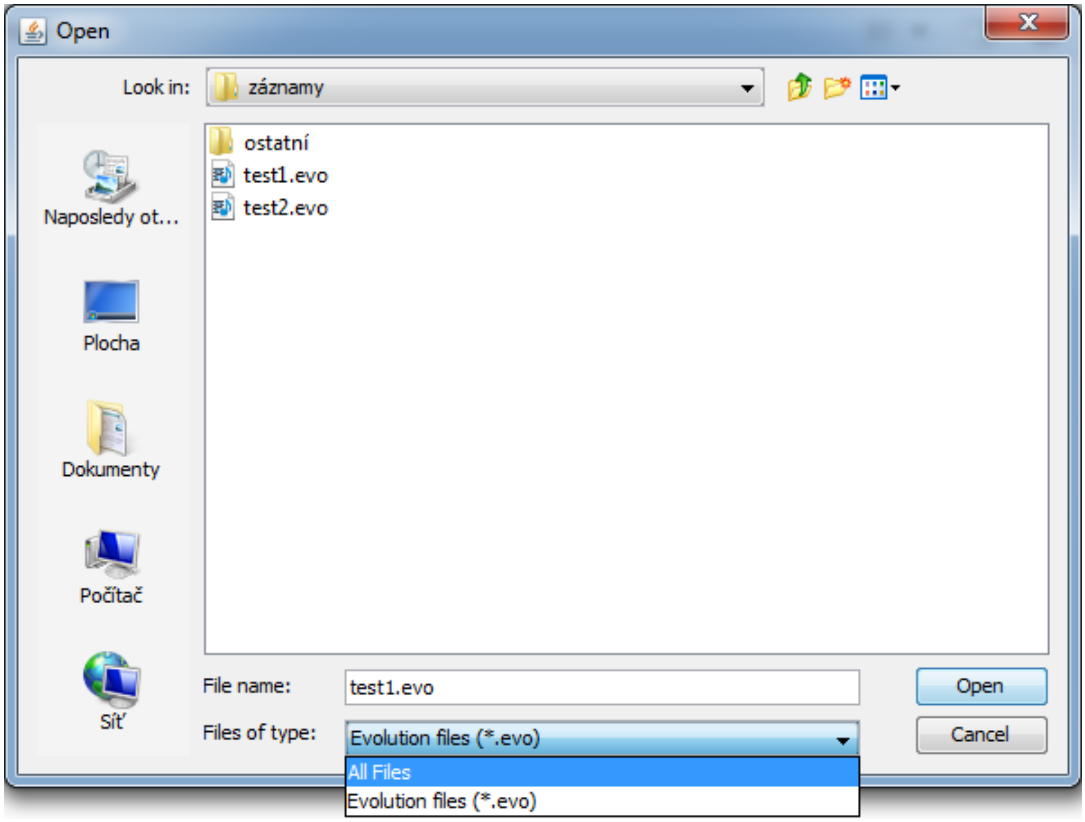
Po spuštění aplikace je zobrazeno prázdné hlavní okno a pomocí menu nebo klávesové zkratky `ctrl+o` je třeba otevřít záznam o průběhu evoluce obvodu ve formátu, který je popsán v kapitole 2.7.1. K výběru souboru je použit standardní dialog pro procházení souborové hierarchie. Pro zvýšení uživatelského pohodlí je do tohoto dialogu přidán souborový filtr, který zobrazuje jen soubory s koncovkou „.evo“, které slouží pro snadnou identifikaci záznamů průběhu evoluce pro vyvinutý program. Použití této koncovky je však dobrovolné a proto nabízí dialog pro otevření souboru i možnost zobrazení všech typů souborů, jak si můžeme všimnout na obrázku 6.1.

Po zvolení souboru se záznamem o průběhu evoluce je provedeno načtení celého záznamu a uložení do hierarchie tříd, která je blíže popsána v sekci 5.3 a zobrazena na obrázku 5.3. Po načtení záznamu do datových struktur programu dojde k jeho navzorkování (více viz sekce 7.2) a poté zobrazení v podobě grafu. Pro analýzu je použit pouze nejlepší chromozom z každé generace neboli ten, který je v příští generaci použit jako rodič.

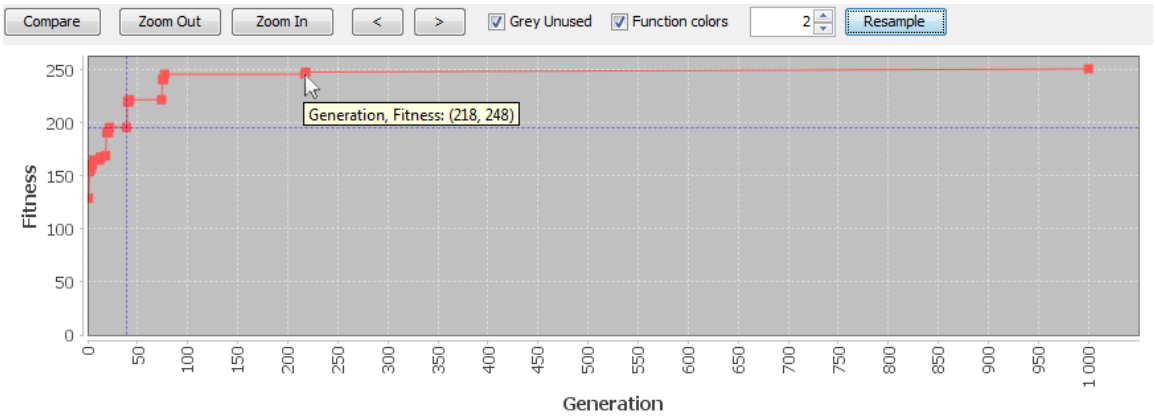
6.1.1 Graf vývoje fitness

Obrázek 6.2 zachycuje graf vývoje fitness, který je zobrazen po otevření a navzorkování záznamu evoluce. Na vodorovné ose „Generation“ jsou zaneseny generace a na svislé ose jsou zaneseny hodnoty fitness. Graf zobrazuje celý průběh evoluce, ne však každou generaci, ale pouze generace, které jsou potenciálně „zajímavé“. Tyto „zajímavé“ generace jsou vybírány na základě velikosti změny hodnoty fitness mezi generacemi (více viz sekce 7.2). Vybraní jedinci jsou v grafu znázorněni červenými čtverci. Po umístění kurzoru myši na libovolný čtverec je zobrazena fitness hodnota a generace příslušného jedince.

Vzorkování na základě změny hodnoty fitness probíhá automaticky a práh minimální změny je vypočten bez vstupu od uživatele. Přesto je uživateli umožněno tento práh změnit a docílit tak „hrubějšího“ vzorkování zvýšením prahu nebo naopak dosáhnout vzorkování jemnějšího volbou nižšího prahu. Tato volba je dostupná nad grafem prostřednictvím



Obrázek 6.1: Dialog pro otevření souboru se záznamem o průběhu evoluce.

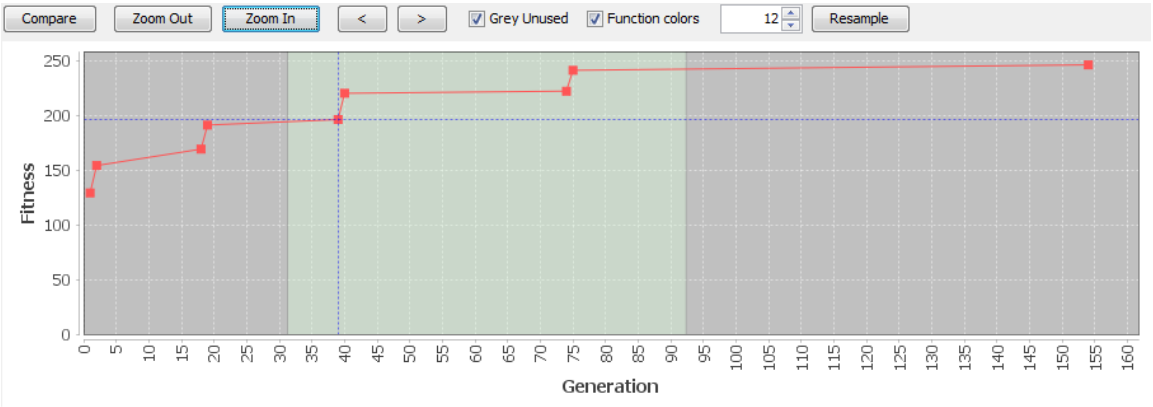


Obrázek 6.2: Graf vývoje fitness.

grafické komponenty „spinner“, která v tomto případě umožňuje zadávání celočíselných kladných hodnot velikosti prahu a rovněž sekvenční změny hodnoty pomocí šipek na pravé straně textového pole. Samotné převzorkování a zobrazení proběhne po stisknutí tlačítka „Resample“. Textové pole pro změnu hodnoty prahu zobrazuje aktuální velikost prahu nezávisle na tom, zda byla nastavena automaticky nebo uživatelem.

Vzhledem k možnému velkému množství generací nabízí graf, zobrazující změny fitness, možnost bližšího zkoumání vybraných úseků evoluce. Tato funkce je zobrazena na obrázku 6.3. Kliknutím a tažením myši v grafu je možno zvolit oblast evoluce pro bližší analýzu.

Zvolená oblast je v grafu podbarvena zeleně. Po označení daného úseku je možno stiskem tlačítka „Zoom In“ provést přiblížení. V takovém případě proběhne opětovné vzorkování evoluce s automaticky vypočteným prahem, ovšem pouze na zvoleném úseku. Navzorkovaný úsek je poté zobrazen místo původního grafu. Tlačítko „Zoom Out“ má opačnou funkci, než tlačítko „Zoom In“ – provede „oddálení“, jinými slovy návrat zpět o jednu úroveň přiblížení. Na obrázku 6.3 je zobrazena již přiblížená část evoluce (generace 0-160) a pro další přiblížení je označen úsek generací 32-94, ve kterém jsou nejvýraznější dvě velké změny hodnoty fitness. Po přiblížení dojde k „jemnějšímu“ navzorkování zvolené oblasti, což můžeme vidět na obrázku 6.4. Takto je umožněna efektivní navigace záznamem o průběhu evoluce.



Obrázek 6.3: Graf vývoje fitness – před přiblížením.

Obrázek 6.4: Graf vývoje fitness – po přiblížení.

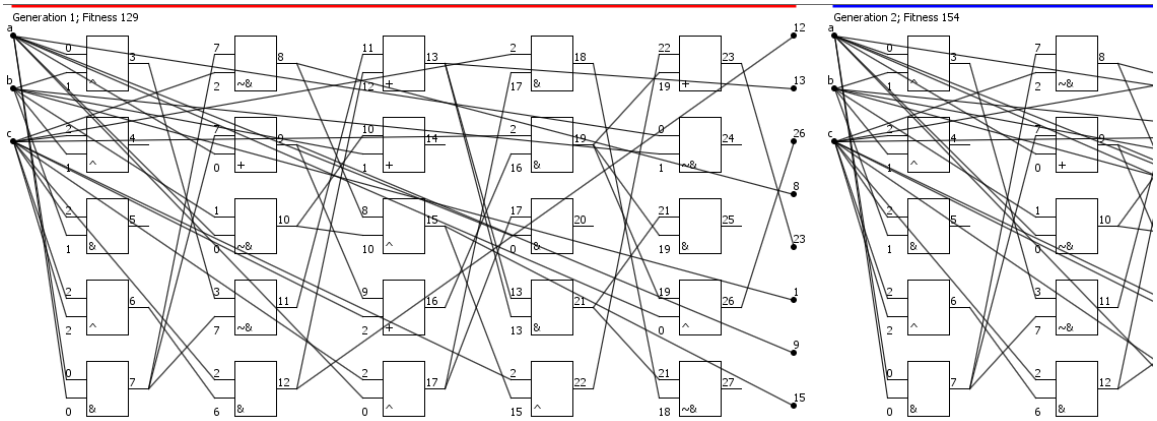
Graf byl implementován s použitím volně dostupné knihovny JFreeChart [13], která slouží pro práci s diagramy. Tato knihovna umožňuje, relativně snadno, vytvářet profesionální diagramy, není však určena pro zobrazování velkého množství dat. Z tohoto důvodu je počet vykreslovaných jedinců v grafu omezen konstantou 1000. Tuto hranici je možno překročit ručním nastavením prahu změny fitness v panelu nástrojů nad grafem, může však dojít k problémům s rychlostí vykreslování nebo velikostí paměti.

Dále si v grafu na obrázcích 6.2, 6.3 a 6.4 můžeme všimnout modrého kříže. Ten slouží pro výběr jedinců. Jedinec označený křížem je zobrazen v detailu pod grafem spolu s následujícími jedinci. Změna pozice označovacího kříže je možná buď kliknutím myši na čtverec označující jedince nebo sekvenčně pomocí tlačítek „<“ (zpět) a „>“ (vpřed), která jsou

umístěna nad grafem.

6.1.2 Detail jedinců

Obrázek 6.5 zobrazuje jedince, který byl v grafu označen křížem, a je proto zobrazen ve spodní části hlavního okna. Jedná se o kandidátní obvod z první generace, jehož fitness hodnota je 129 – tyto informace jsou zobrazeny v levé horní části nad každým zobrazovaným jedincem. Samotný chromozom sestává ze tří vstupů reprezentovaných černými kruhy, které jsou automaticky označeny písmeny abecedy (v tomto případě a, b, c), dále jsou na vstupy napojena hradla v matici o velikosti 5×5. Za tímto jedincem následuje tolik dalších chromozomů, kolik je na dané ploše možno zobrazit. Zobrazují se pouze chromozomy, které jsou aktuálně navzorkovány v příslušném grafu průběhu evoluce.

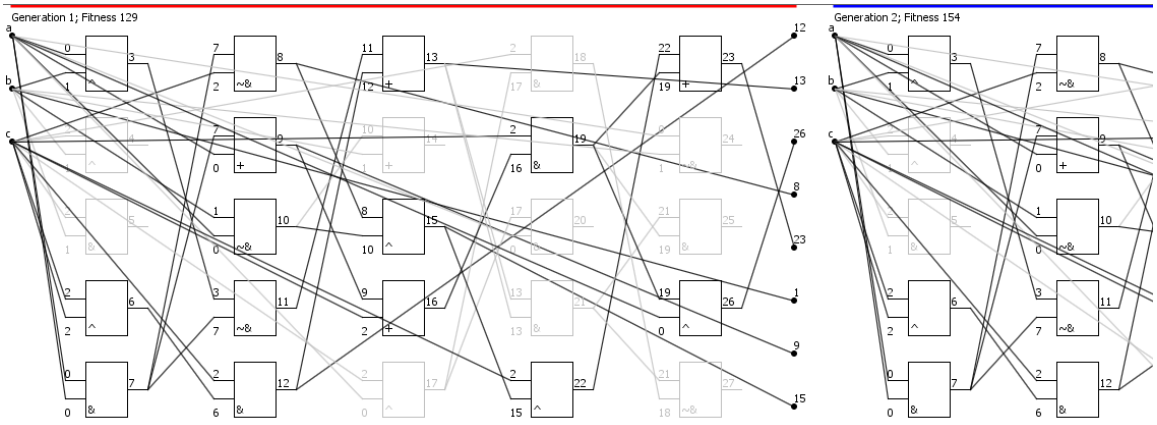


Obrázek 6.5: Zobrazení vybraných jedinců.

Každé hradlo má dva vstupy, jeden výstup a logickou funkci. U obou vstupů každého hradla je vykresleno číslo hradla, ke kterému je daný vstup připojen. Na výstupu hradel je pak vykresleno číslo daného hradla. Uvnitř každého hradla můžeme najít označení funkce a to buď číselné označení ze záznamu o průběhu evoluce nebo označení zvolené uživatelem prostřednictvím tzv. „function file“ (více o formátu tohoto souboru viz sekce 6.4). Poslední částí chromozomu jsou výstupy, které jsou, stejně jako vstupy, reprezentovány černými kruhy, u nichž je zobrazeno číslo hradla, ke kterému je daný výstup připojen.

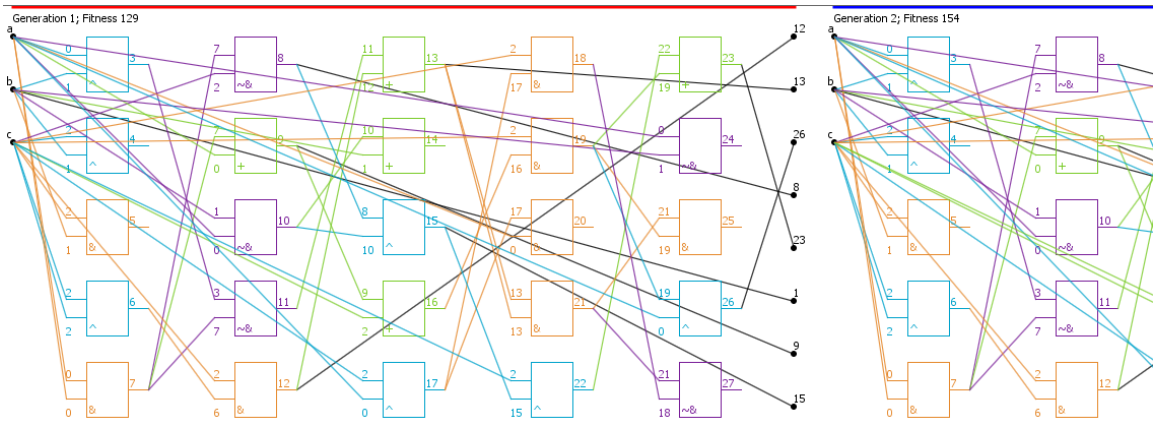
Ačkoliv jsou na obrázku 6.5 zobrazeny všechny informace o vybraném chromozomu (funkce hradel, zapojení, počet vstupů a výstupů), je obtížné z tohoto zobrazení vyčíst jakékoliv další informace jako je například identifikace nevyužitých hradel. Na obrázku 6.6 si můžeme všimnout, že některá hradla a spoje jsou vykreslena černou a jiná šedou barvou. Implicitně je v aplikaci nastavena šedá barva pro vykreslování nepotřebných hradel a černá barva pro využitá hradla a spoje. V hlavním okně má uživatel možnost vykreslit celý obvod jednotnou černou barvou prostřednictvím „žaškrtávacího“ tlačítka „Grey Unused“, které se nachází v panelu nástrojů nad grafem průběhu evoluce. S využitím této funkce lze velmi jednoduše zjistit která hradla jsou v chromozomu nevyužita.

Napravo od tlačítka „Grey Unused“ můžeme najít další tlačítko s názvem „Function Colors“. Pokud uživatel načetl tzv. „function file“, ve kterém specifikoval funkce jednotlivých hradel a jejich barevné označení, jsou pak hradla barevně odlišena právě těmito barvami na základě svých funkcí. Tuto funkci lze ovládat právě tlačítkem „Function Colors“. V případě, že je tlačítko „zaškrtnuté“, ale „function file“ není načten, je použita standardní černá, pří-



Obrázek 6.6: Zobrazení vybraných jedinců s šedými nevyužitými hradly.

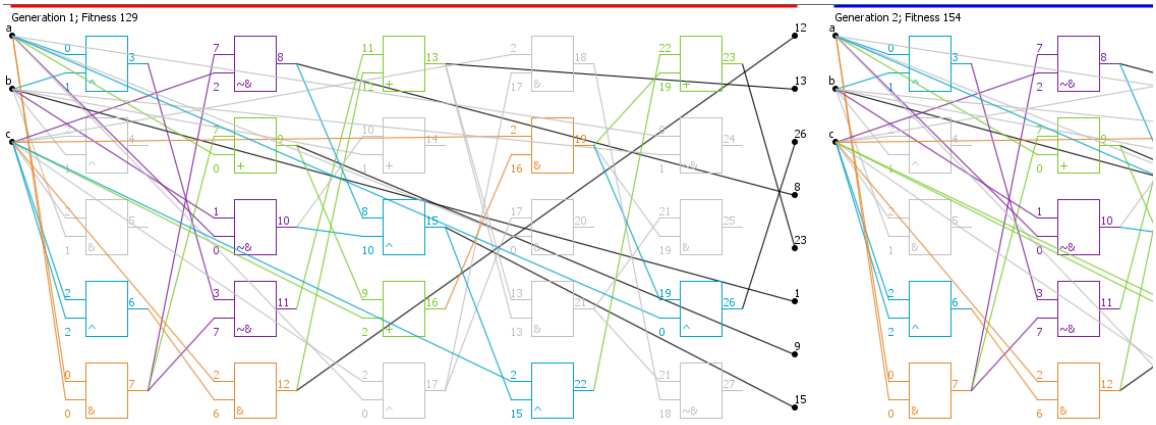
padně šedá barva. Na obrázku 6.7 je zobrazen obvod, pro který byl vytvořen a načten „function file“. Jak si můžeme všimnout, označení logických funkcí uvnitř hradel se změnilo z číselného na značení odpovídající definici v načteném souboru s funkcemi hradel – („function file“). Standardní černá barva hradel a spojů se také změnila na barvy, které jsou rovněž specifikovány ve „function file“. Použití rozdílných barev pro vykreslení jednotlivých hradel ještě více napomáhá k rozlišení jejich funkcí.



Obrázek 6.7: Zobrazení vybraných jedinců s barevným odlišením hradel.

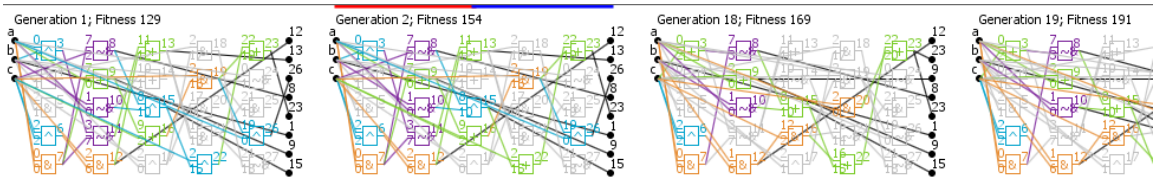
Podobně jako u standardního zobrazení černou barvou, kde existuje možnost odlišení nepoužitých hradel šedou barvou, je tato funkce dostupná i při použití vlastních barev. V případě, že je „zaškrtnuté“ jak tlačítko „Grey Unused“, tak „Function Colors“ a zároveň je načten soubor s funkcemi a barvami hradel, jsou použitá hradla vykreslena příslušnými barvami a nepoužitá hradla jsou vykreslena sedě. Jak se můžeme přesvědčit na obrázku 6.8, toto zobrazení nabízí nejlepší přehled jak o použitých a nepoužitých hradlech, tak o jejich funkcích. Volba vykreslovacího módu je však ponechána na uživateli.

Z důvodu podpory různě velkých chromozomů, zobrazovacích ploch a všeobecného komfortu uživatele, byla implementována možnost přiblížení a oddálení (zoom) chromozomů. Toto lze provést otáčením kolečka myši – otočením směrem od uživatele je provedeno oddálení a otočení kolečka směrem k uživateli způsobí přiblížení chromozomu. Aby funkce zoom fungovala, musí se kurzor myši při otáčení kolečkem nacházet v oblasti pro vykres-



Obrázek 6.8: Zobrazení vybraných jedinců s barevným odlišením hradel a šedými nevyužitými hradly.

lování chromozomů. Obrázek 6.9 ilustruje použití funkce zoom pro oddálení chromozomů a zobrazení více jedinců najednou na malé ploše.



Obrázek 6.9: Oddálení chromozomů.

6.1.3 Výsledné hlavní okno

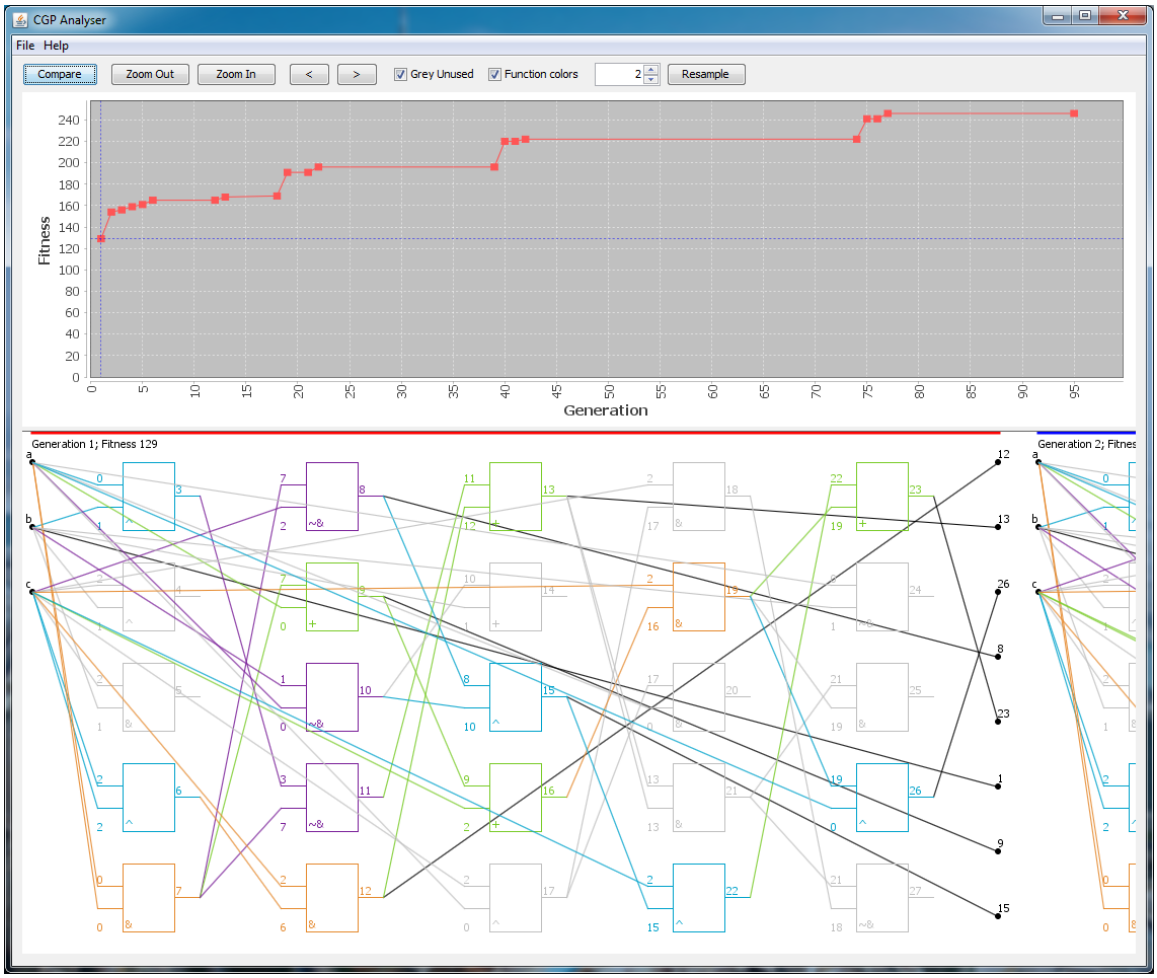
Samotné hlavní okno, vyobrazené na obr. 6.10, se skládá z menu, panelu nástrojů, grafu vývoje fitness, posuvného oddělovače a detailu chromozomů.

V menu lze najít následující položky:

- „Open Evolution“ (klávesová zkratka `ctrl+o`), sloužící pro otevření záznamu o průběhu evoluce,
- „Open Function File“ (klávesová zkratka `ctrl+i`), sloužící pro otevření souboru s funkcemi hradel (tzv. „function file“),
- „Create Default FF“ (klávesová zkratka `ctrl+p`), sloužící pro vytvoření standardního „function file“ například v případě jeho poškození,
- „Gate Usage“ (klávesová zkratka `ctrl+u`), sloužící pro zobrazení okna se statistikou využití hradel,
- menu „help“ obsahuje okno s informacemi o projektu a autorovi.

Pod menu se nachází panel nástrojů s tlačítky pro obsluhu hlavního okna. Dále nalezneme graf vývoje fitness, který je od detailu chromozomů oddělen grafickou komponentou

„split pane“. Jedná se o posuvný oddělovač, který umožňuje měnit velikost přidělenou grafu s vývojem fitness a detailu chromozomů. Tato komponenta, podobně jako funkce zoom, zlepšuje uživatelský komfort a umožňuje snadnější práci s programem, zvláště na menších obrazovkách nebo při práci s většími chromozomy. V hlavním oknu na obrázku 6.10 je využita tato funkce, přičemž je zvětšena plocha pro detail chromozomů na úkor plochy pro zobrazení grafu. Při zvětšování nebo zmenšování velikosti hlavního okna je rovnoměrně zvětšována, případně zmenšována velikost přidělená jak grafu, tak detailu chromozomů.



Obrázek 6.10: Hlavní okno.

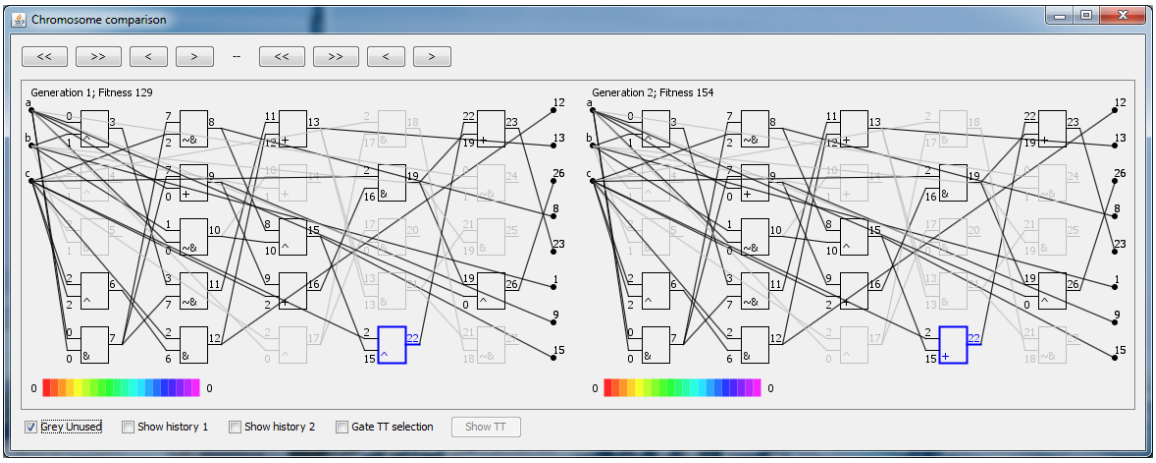
Můžeme si všimnout, že na obrázcích 6.5, 6.6, 6.7, 6.8 i 6.10 je nad některými chromozomy červená linka a nad jinými chromozomy je modrá linka. Toto označení slouží pro zvýraznění vybraných chromozomů. Výběr se provádí kliknutím myší na zvolený chromozom na vykreslovacím plátně. Po kliknutí levým tlačítkem je chromozom označen červenou čarou a kliknutí pravým tlačítkem způsobí označení chromozomu modrou čarou. Pokud uživatel označí levým i pravým tlačítkem myši jeden chromozom, je označen částečně červenou i modrou linkou, jak je vidět na obrázku 6.9. Označené chromozomy je poté možno podrobněji analyzovat v porovnávacím okně, kde jsou zobrazeny právě dva, uživatelem vybrané, chromozomy. Toto okno je možno otevřít pomocí tlačítka „Compare“ v panelu nástrojů nad grafem vývoje fitness.

6.2 Porovnávací okno

Porovnávací okno, jak vyplývá z názvu, slouží pro porovnávání chromozomů, není to však jediná funkce tohoto okna, o čemž se přesvědčíme dále. Po výběru chromozomů v hlavním okně a stisku tlačítka „Compare“ je zobrazeno porovnávací okno s příslušnými chromozomy.

6.2.1 Navigace a rozdíly mezi chromozomy

Obrázek 6.11 ilustruje použití porovnávacího okna pro nalezení rozdílů mezi dvěma chromozomy. Na levé straně okna je chromozom z první generace, jehož hodnota fitness je 129. V pravé polovině okna je chromozom z další generace (číslo 2), který má ovšem fitness 154. Takto velká změna fitness mezi generacemi byla vyhodnocena vzorkovacím algoritmem jako dostatečná a oba chromozomy byly přidány do grafu vývoje fitness v hlavním okně, odkud byly vybrány pro analýzu v porovnávacím okně. Pokud nás zajímá, co způsobilo tuto skokovou změnu fitness, porovnávací okno v základním módu vyznačuje právě rozdíly mezi dvěma chromozomy. V tomto případě nastala jediná změna ve funkce hradla číslo 22 z logické funkce „xor“ na funkci „or“. Změna funkce hradla je označena modrou barvou příslušných hradel.



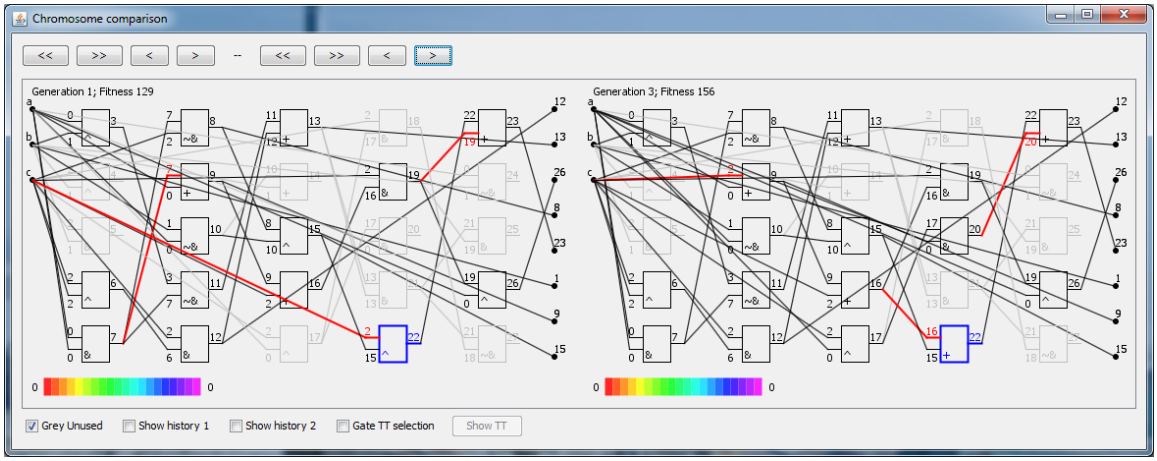
Obrázek 6.11: Porovnávací okno hned po otevření.

Všimněme si, že se v horní části porovnávacího okna nachází osm tlačítek ve dvou skupinách po čtyřech. Tyto dvě skupiny jsou odděleny pomlčkou. První skupina čtyř tlačítek patří k prvnímu (levému) chromozomu a skupina dalších čtyř tlačítek patří ke druhému (pravému) chromozomu. Tato tlačítka slouží změnu vybraného chromozomu, levého či pravého, bez nutnosti návratu do hlavního okna.

Tlačítka s označením „<“ a „>“ slouží ke změně aktuálního chromozomu na chromozom z předcházející generace. Pokud tedy použijeme funkci dopředu („>“) pro pravý chromozom v okně z obrázku 6.11, změní se pravý chromozom z generace číslo 2 na nejlepší chromozom z generace číslo 3. Výsledek po provedení této operace je zobrazen na obrázku 6.12.

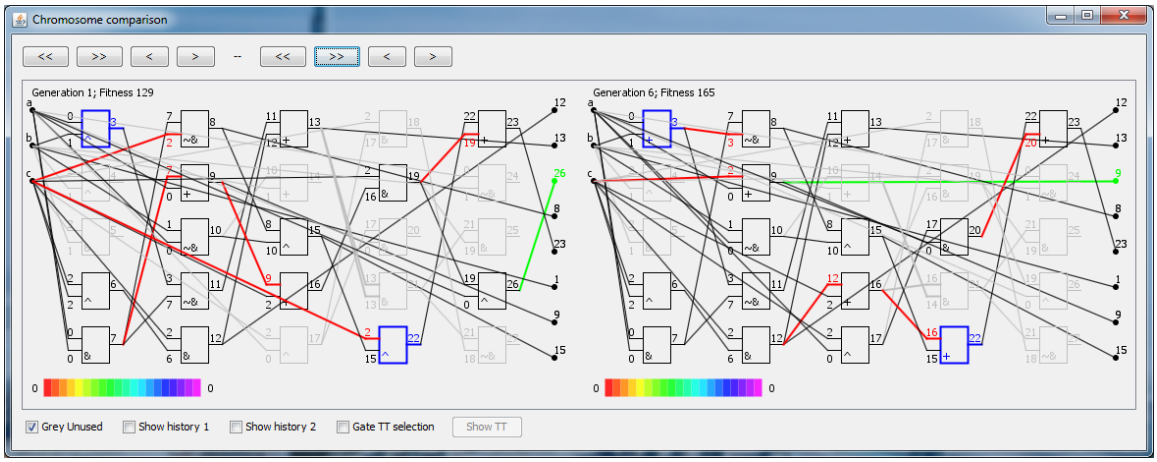
Můžeme si všimnout, že na obrázku 6.12 se zvýšil počet změn mezi chromozomy oproti obr. 6.11. Tyto nové rozdíly jsou vyznačeny červenou barvou proto, že se jedná o změny v zapojení. Díky barevnému rozlišení je tedy možné velmi jednoduše analyzovat různé typy změn.

Nyní přichází na řadu objasnění funkce zbývajícího typu tlačítek s označením „<<“ a „>>“. Ta slouží také pro posuv na předchozí respektive další chromozom. Použití tla-



Obrázek 6.12: Porovnávací okno po stisku tlačítka „>“ pro pravý chromozom.

čítek „<<“ a „>>“ však nezpůsobí posun na chromozom z předcházející nebo následující generace, ale na předcházející nebo následující chromozom, který je navzorkován v grafu průběhu evoluce. Pokud je tedy například vybrán chromozom generace číslo 3 a další chromozom, který je navzorkován v grafu vývoje fitness, je z generace č. 6, pak stisk tlačítka „>>“, způsobí přeskočení z generace č. 3 rovnou na chromozom z generace číslo 6. Použití této funkce je zobrazeno na obrázku 6.13.

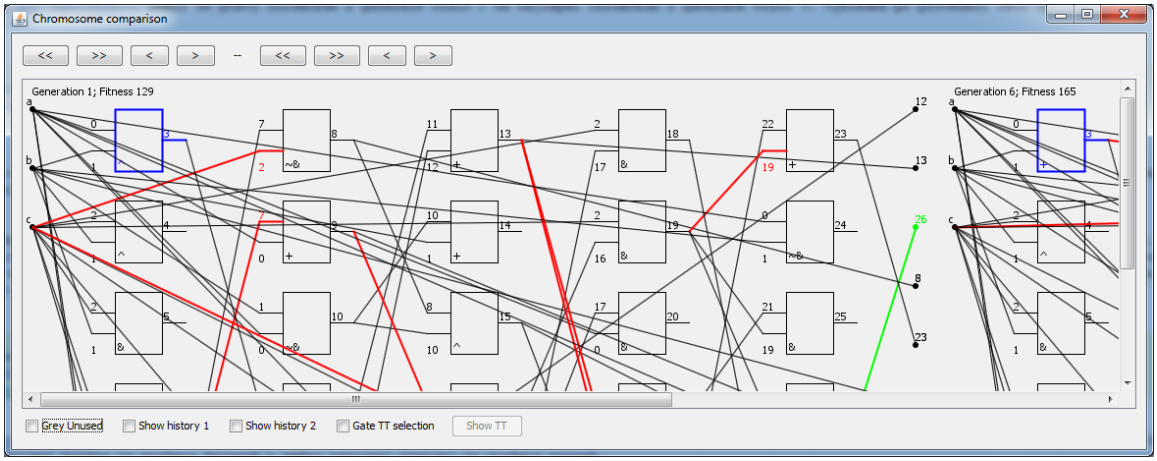


Obrázek 6.13: Porovnávací okno po stisku tlačítka „>>“ pro pravý chromozom.

Na obrázku 6.13 si také můžeme všimnout zelené barvy při zobrazování rozdílů mezi chromozomy. Takto je zvýrazněno rozdílné zapojení výstupů obvodu. Změna funkce hradla je tedy značena modře, změna zapojení hradla je značena červeně a změna zapojení výstupu je značena zeleně.

Podobně jako v hlavním okně je i ve srovnávacím okně dostupná funkce označení nepoužitých hradel šedou barvou – „zaškrtávacím“ tlačítkem „Grey Unused“, které je umístěno pod chromozomy u spodního okraje okna. Dále je možno použít, stejně jako v hlavním okně, funkci zoom otáčením kolečka myši, pokud se kurzor nachází na ploše s chromozomy. Pokud při větším přiblížení zasahuje chromozom mimo srovnávací okno, je přidán vertikální, případně horizontální posuvník pro možnost „skrolování“. Okno s přiblíženým chromozomem

a aktivními posuvníky je na obrázku 6.14.



Obrázek 6.14: Porovnávací okno – funkce zoom s posuvníky a neoznačení nevyužitých hradel.

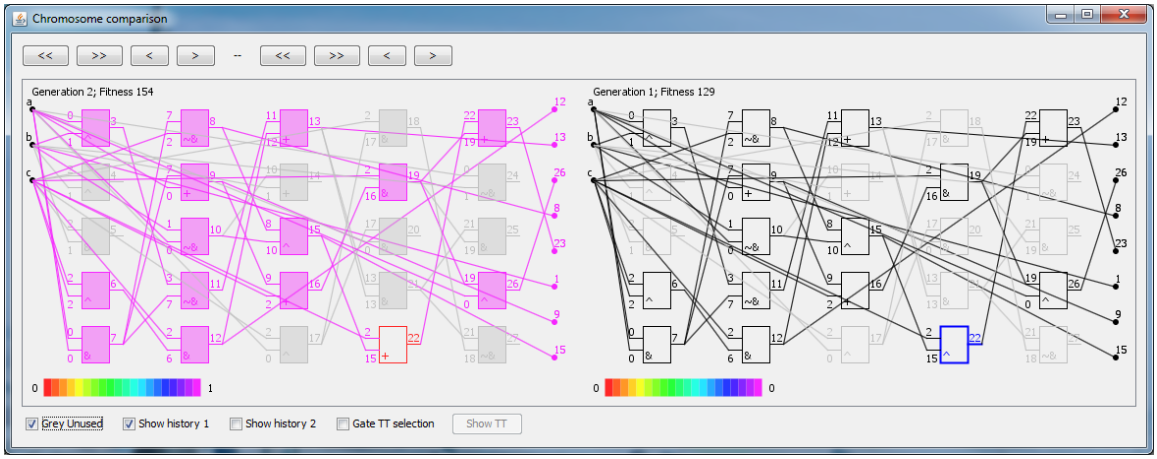
6.2.2 Historie

Další funkcí porovnávacího okna je zobrazení historie vybraných chromozomů. Tato funkce je aktivována „zaškrtávacím“ tlačítkem „Show history 1“ pro první (levý) chromozom a „Show history 2“ pro chromozom druhý. Tato tlačítka jsou umístěna ve spodní části okna. Účelem funkce historie je zjistit, jak dlouho byla každá komponenta chromozomu nezměněna, neboli jak „stará“ daná část chromozomu je.

Tzv. stáří komponenty chromozomu (hradla nebo spoje) je vyznačeno barevně. Aby bylo jasné, co přesně jednotlivé barvy znamenají, je pod chromozomem umístěna barevná paleta s jedním číslem na každé straně. Paleta se skládá z dvaceti barev barevného spektra. Bylo by sice možné použít mnohem více barev, což moderní monitory umožňují, ale odlišit lidským okem drobné barevné změny je velmi obtížné, a proto byla zvolena varianta menší barevné palety, zato s většími rozdíly mezi jednotlivými barvami. Číslo na levé straně palety je vždy nula a příslušná barva (červená) označuje relativně „nedávnou“ změnu (tzn. například v současné generaci). Na opačné straně palety je po aktivaci funkce historie zobrazeno maximální stáří některé části chromozomu, jinými slovy řečeno kolik generací nebyla nejstarší komponenta změněna. Tato nejstarší část potom bude vykreslena fialovou barvou a pro ostatní části bude proporcionálně přidělena jedna z dvaceti barev palety.

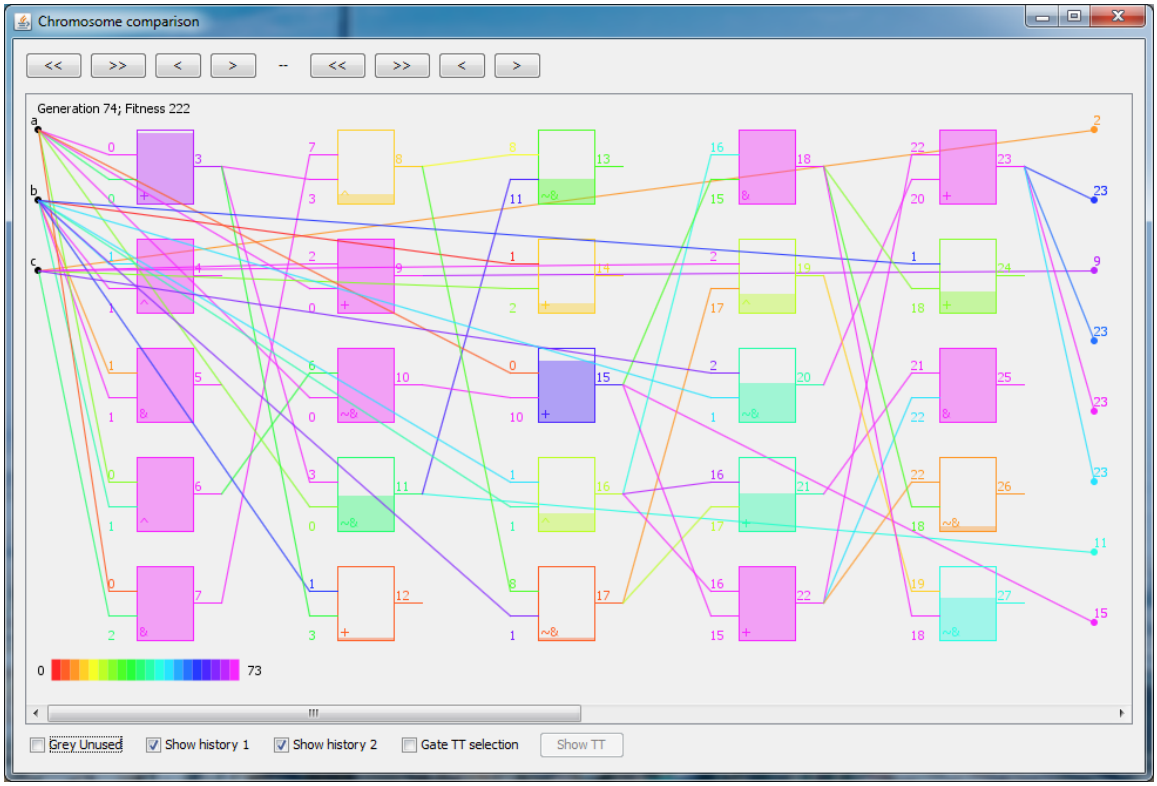
Nejjednodušší demonstrace funkce historie je na obrázku 6.15. Pro levý chromozom generace č. 2 je aktivována funkce historie a všechna hradla i spoje jsou vykreslena fialovou barvou, která značí stáří jedna. Pouze hradlo číslo 22 je červené, má tedy nulové stáří, protože toto hradlo bylo změněno právě v této generaci. Pro ověření se můžeme podívat na pravý chromozom, který je z generace číslo 1. Tento chromozom nemá aktivní funkci zobrazování historie, ale je ponechán původní mód zobrazování rozdílů. Zde pak vidíme, že rozdíl mezi chromozomy je skutečně jen ve funkci hradla číslo 22.

Obrázek 6.16 ilustruje použití funkce historie na chromozomu z generace 74. Jak si můžeme všimnout, je „stáří“ hradel vyznačeno kromě odpovídající barvy z palety také částečným vyplněním hradla. Čím je hradlo „starší“, tím je více (až zcela) vyplněno. Například hradla 4, 5, 6, 7, 9, 10, 18, 22, 23 a 25 jsou vykreslena fialovou barvou, značící



Obrázek 6.15: Zobrazení historie.

maximální „stáří“. Současně jsou zcela vyplněna, protože jsou nejdéle nezměněna. V tomto případě nejsou hradla změněna vůbec (jedná se o generaci 74 a maximální stáří je 73 - viz číslo na pravé straně palety). Ostatní hradla jsou různě „stará“, což lze jednoduše určit díky barevnému označení i částečnému vyplnění hradel.



Obrázek 6.16: Zobrazení historie.

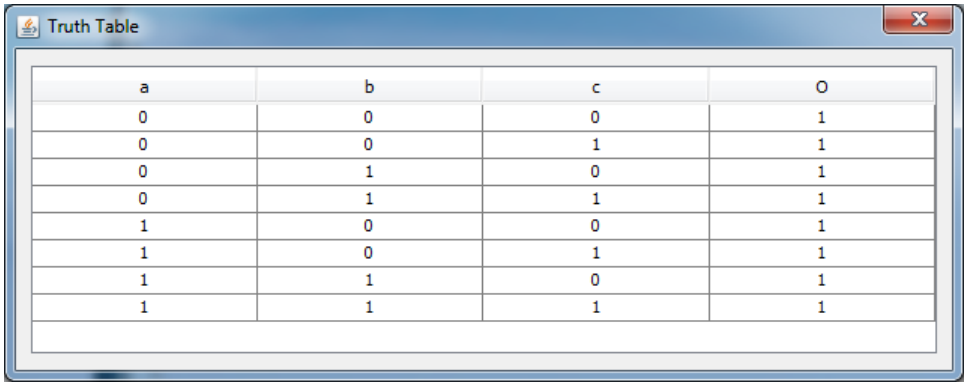
6.2.3 Pravdivostní tabulka

Poslední funkcí, dostupnou v porovnávacím okně, je pravdivostní tabulka části obvodu a porovnání dvou pravdivostních tabulek. Tato funkce je aktivována „zaškrtávacím“ tlačítkem „Gate TT selection“. Po zvolení tohoto tlačítka jsou deaktivovány funkce zobrazení rozdílů i historie (tlačítka pro zobrazení historie se navíc stanou nefunkčními) a chromozomy jsou vykresleny černou barvou (případně jsou šedou barvou vykreslena nevyužitá hradla). Výběr části obvodu je proveden kliknutím na koncové hradlo, jehož výstupní pravdivostní tabulka nás zajímá. Po kliknutí na toto hradlo je automaticky červeně zvýrazněn celý podobvod, který je spojený se zvoleným hradlem. Na obrázku 6.17 je zobrazen podobvod s koncovým hradlem číslo 23, na které bylo kliknuto. Díky automatickému označení příslušné části obvodu (tj. hradel i spojů), lze identifikovat všechna připojená hradla a použité vstupy.

Obrázek 6.17: Výběr části obvodu.

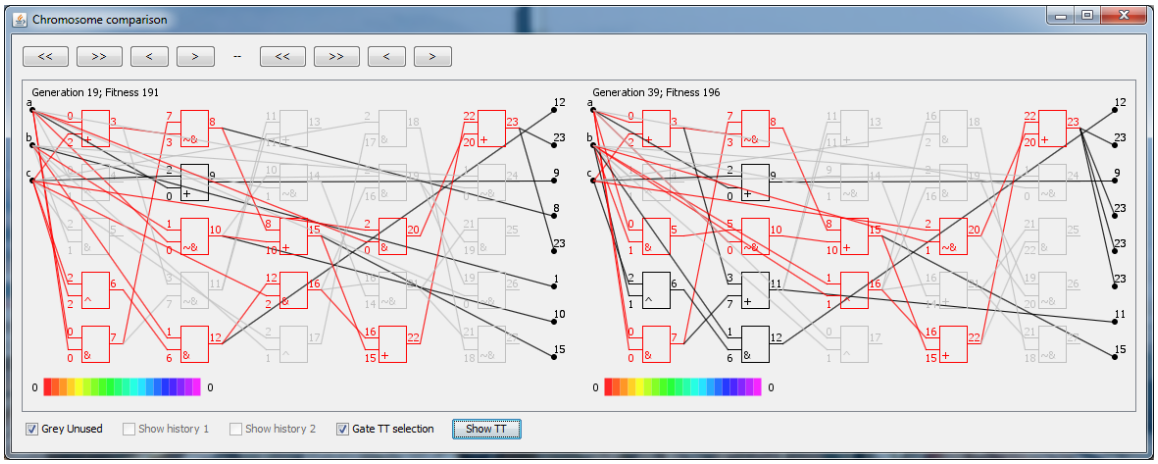
Po označení podobvodu lze použít tlačítko „Show TT“, které zobrazí příslušnou pravdivostní tabulku. Na obrázku 6.18 je pravdivostní tabulka vybrané části obvodu z obrázku 6.17. Jedná se o klasickou pravdivostní tabulku se vstupy „a“, „b“, „c“ a výstupem „O“.

Pokud je vybrána část obvodu v obou chromozomech srovnávacího okna (viz obr. 6.19), potom jsou po kliknutí na tlačítko „Show TT“ zobrazeny pravdivostní tabulky obou podobvodů a zároveň jsou barevně vyznačeny stejné a rozdílné výstupní hodnoty, jak je ukázáno na obrázku 6.20.

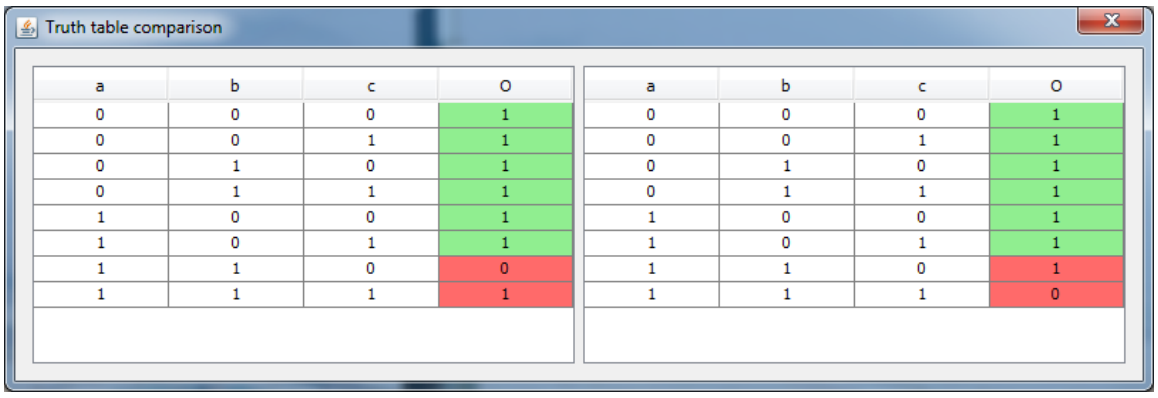


a	b	c	O
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Obrázek 6.18: Pravdivostní tabulka.



Obrázek 6.19: Výběr části obvodu dvou chromozomů.



a	b	c	O
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

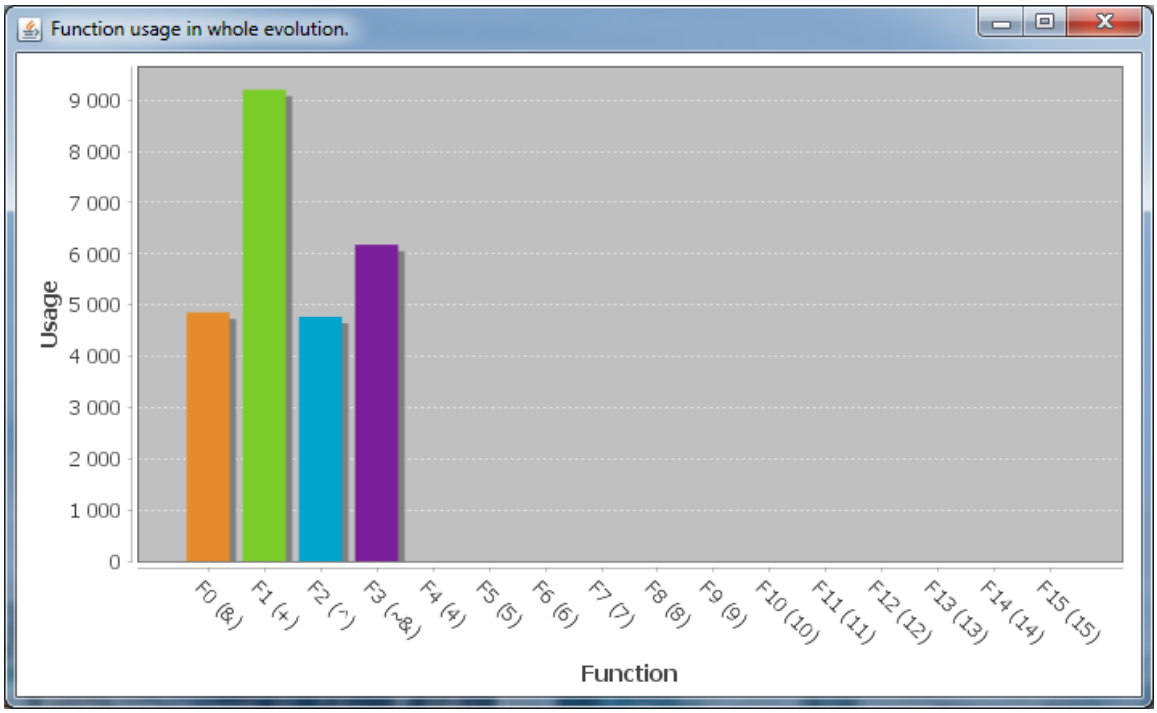
a	b	c	O
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Obrázek 6.20: Porovnání pravdivostních tabulek.

6.3 Statistika využití hradel

Poslední funkcí vyvinuté aplikace je statistika využití hradel podle jejich logické funkce. Okno s touto statistikou lze otevřít z menu v hlavním okně kliknutím na položku „Gate Usage“ nebo stiskem klávesové zkratky `ctrl+u`. Tato funkce vypočítá využití jednotlivých logických funkcí v hradlech každého nejlepšího chromozomu (neboli chromozomu, který je

použit jako rodič v další generaci) v celé evoluci a výsledky zobrazí v podobě grafu, což je znázorněno na obrázku 6.21. Můžeme si všimnout, že na vodorovné ose jsou zaneseny logické funkce a na svislé ose je počet použití příslušné funkce. Pro označení každé funkce je použita barva načtená ze souboru s funkcí hradel a také příslušný symbol. Pokud není soubor „function file“ načten nebo v něm není daná funkce specifikována, je použita černá barva a pouze číselné označení funkce.



Obrázek 6.21: Statistika využití hradel.

6.4 Function file

„Function file“, neboli soubor s funkcemi hradel, je speciální typ souboru, který obsahuje dodatečné informace o hradlech chromozomů. Konkrétně přiřazuje číselnému označení funkcí hradel, které je uvedeno v záznamu o průběhu evoluce, označení textové (lze použít jeden i více znaků) a navíc přiřazuje jednotlivým funkcím barvu, která potom může být použita pro vykreslování hradel pro grafické odlišení jejich funkcí. „Function file“ není vyžadován pro samotné prohlížení záznamu o průběhu evoluce, ale některé funkce programu nejsou dostupné, pokud není tento soubor načten – jedná se o vykreslení hradel různými barvami na základě funkce a generování pravdivostní tabulky části obvodu, případně porovnání pravdivostních tabulek.

Formát souboru „function file“ je velmi jednoduchý. Jeden záznam se skládá z čísla funkce, které odpovídá číslování v použitém záznamu o průběhu evoluce (typicky 0-15, protože program podporuje hradla se dvěma vstupy, která tedy mohou mít jednu z 16 logických funkcí). Následuje kódové označení funkce, podle kterého aplikace rozpozná, o jakou logickou funkci se jedná – výčet těchto kódů (jmen) je uveden v tabulce 6.1. Poté je třeba uvést symbol nebo řetězec označující danou funkci – toto označení bude použito při vykreslování hradel na místě pro uvedení funkce hradla (uvnitř každého hradla), místo implicitního čísel-

logická operace	**kód**	**alternativní kód**
F_0	F	0
F_1	AND	.
F_2	A.~B	.~
F_3	A	A
F_4	~A.B	~A.
F_5	B	B
F_6	XOR	^
F_7	OR	+
F_8	NOR	~+
F_9	XNOR	~^
F_{10}	~B	~B
F_{11}	A+~B	+~
F_{12}	~A	~A
F_{13}	~A+B	~A+
F_{14}	NAND	~.
F_{15}	T	1

Tabulka 6.1: Tabulka kódů pro logické funkce hradel.

A	B	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Tabulka 6.2: Pravdivostní tabulka logických funkcí 0-7.

ného označení. Poslední položkou je barva pro hradla s danou funkcí. Barvu je nutné zadat v hexadecimálním formátu. Části záznamu jsou odděleny mezerou a jednotlivé záznamy jsou odděleny novým řádkem. Kvůli možným nejasnostem jsou, v tabulkách 6.2 (funkce F_0 - F_7) a 6.3 (funkce F_8 - F_{15}), uvedeny i pravdivostní tabulky logických funkcí z tabulky 6.1.

Kódy logických funkcí nezávisí na velikosti písmen, tedy nejsou tzv. „case-sensitive“. Z toho vyplývá, že logickou funkci „nand“ (F_{14}) můžeme definovat například řetězcem „nand“ nebo „NAND“, ale i řetězcem „Nand“ atd.

Formát souboru „function file“ umožňuje velmi jednoduché použití komentářů: řádek začínající znakem „-“ je považován za komentář a aplikací je ignorován. Každý řádek s jiným počátečním znakem včetně prázdných řádků je však považován za záznam o funkci hradla

A	B	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Tabulka 6.3: Pravdivostní tabulka logických funkcí 8-15.

a bude dále zpracováván, proto je nutné, aby každý řádek, který není záznam o funkci hradla byl uvozen znakem „-“. Narozdíl od souboru se záznamem o průběhu evoluce, pro který je doporučeno používat nestandardní koncovku „.evo“, je pro „function file“ doporučena standardní koncovka „.txt“, která se používá pro soubory textového charakteru. Aplikace podporuje koncovku „.txt“ podobně jako koncovku „.evo“ u záznamu o průběhu evoluce – konkrétně nastavením filtru dialogu pro otevírání souborů (viz dialog pro otevírání souborů s koncovkou „.evo“ na obrázku 6.1). V případě poškození souboru s funkcí hradel umožňuje aplikace vygenerovat ukázkový „function file“. Tuto možnost nalezneme v menu aplikace. Položka menu „Create Default FF“ (klávesová zkratka `ctrl+p`) způsobí vytvoření nového souboru jménem „defaultTT.txt“ v pracovním adresáři programu. Tento soubor také obsahuje instrukce k vytváření a modifikaci souboru „function file“.

Příklad 1.
Zde je pro ilustraci uveden příklad jednoduchého souboru s funkcemi hradel. Jak můžeme vidět, jsou zde definovány funkce pro čtyři typy hradel („0-3“). Hradlu „0“ je přiřazena logická funkce „and“, symbol „&“ a barva „E68A2E“, což odpovídá oranžové barvě. V definici logické funkce „3“ – „nand“ je ilustrována možnost přiřazení více znaků, pro symbol funkce – „~&“.

```
-Number Name Symbol Color
0 and & E68A2E
1 or + 7ACC29
2 xor ^ 00A3CC
3 nand ~& 7A1F99
```

Kapitola 7

Zajímavé části implementace a omezení

Tato kapitola se zabývá zajímavými částmi implementace aplikace a vysvětluje postupy použité při implementaci. Rovněž jsou charakterizována omezení aplikace.

7.1 Nalezení nejlepšího chromozomu

Protože aplikace pracuje pouze s nejlepším chromozomem z každé generace, tzn. s chromozomem, který je v další generaci použit jako rodič, je nutné tohoto jedince v každé generaci identifikovat. Algoritmus pro nalezení tohoto chromozomu je stejný jako v implementaci CGP Zdeňka Vašíčka v sadě Tools4CGP [24], aby byla zachována konzistence se záznamem o průběhu evoluce, pro jehož generování je použita také tato implementace. Algoritmus 3 zachycuje postup nalezení rodiče v jedné generaci.

Proměnná `fitness` obsahuje hodnotu fitness aktuálního chromozomu, proměnná `bestFitness` uchovává maximální hodnotu fitness, které je možno dosáhnout. Tato proměnná je nezbytná pro správné určení nejlepšího jedince po dosažení maximální hodnoty fitness (tzn. po nalezení plně funkčního obvodu). Proměnné `chromosomeIndex` a `bestChromosomeIndex` obsahují index aktuálního chromozomu a nejlepšího chromozomu v dané generaci. Po dosažení maximální hodnoty fitness je rozhodující počet využitých hradel, který je uložen v proměnných `usedBlocks` a `usedBlocksBest`.

Algoritmus 3: Nalezení rodiče

Input: Generace
Output: Rodič

```
1:  for chromosomeIndex = 0 to populationSize do
2:      if isFirstGeneration then
3:          if fitness > bestFitness then
4:              bestFitness = fitness;
5:              bestChromosomeIndex = chromosomeIndex;
6:          end
7:      else if chromosomeIndex != bestChromosomeIndex then
8:          if fitness == maxfitness then
9:              if usedBlocks <= usedBlocksBest then
10:                 usedBlocksBest = usedBlocks;
11:                 bestFitness = fitness;
12:                 bestChromosomeIndex = chromosomeIndex;
13:             end
14:         else if fitness >= bestFitness then
15:             bestFitness = fitness;
16:             bestChromosomeIndex = chromosomeIndex;
17:             usedBlocksBest = ARRSIZE;
18: end for
```

7.2 Vzorkování

Algoritmus 4 popisuje průběh vzorkování záznamu o evoluci. Toto vzorkování se provádí po načtení evoluce a při každém přiblížení grafu průběhu evoluce, rovněž také při převzorkování s prahem, který zvolil uživatel (v tomto případě se neprovádí operace `guessThreshold(genFromIndex, genToIndex)` pro odhad prahu). Pokud velikost navzorkovaných dat překročí konstantu 1000, je zvýšen práh změny fitness a vzorkování se opakuje.

Algoritmus 4: Vzorkování

Input: Seznam generací, ohraničení
Output: Navzorkovaný seznam generací

```
1:  fitnessDeltaThreshold = guessThreshold(genFromIndex, genToIndex);
    List<Generation> toDisplay;
2:  for genFromIndex to genToIndex do
3:      currentFitnessDelta = (currentChromosomeFitness -
        previousChromosomeFitness);
4:      if currentFitnessDelta >= fitnessDeltaThreshold then
5:          addToDisplay(toDisplay, prevGeneration, currentGeneration);
6:      end
7:  end for
```

Proměnná `fitnessDeltaThreshold` obsahuje práh pro zařazení dané změny do seznamu `toDisplay`, jehož prvky jsou poté zobrazeny v grafu vývoje fitness v hlavním

okně. Proměnné `genFromIndex` a `genToIndex` ohraničují úsek evoluce, který je právě vzorkován. Pro zařazení chromozomů do seznamu `toDisplay` tedy musí rozdíl jejich hodnot fitness (`currentChromosomeFitness - previousChromosomeFitness`) překročit aktuální práh `fitnessDeltaThreshold`.

7.3 Omezení

Kromě omezení v podobě velikosti dostupné zobrazovací plochy existují i omezení velikosti vstupního souboru se záznamem o průběhu evoluce. Tady je omezujícím faktorem rychlost načítání dat ze souboru a především doba jejich zpracování a uložení do datových struktur aplikace. Po uložení vstupních dat do datových struktur programu je práce s aplikací plynulá. Například na počítači s procesorem Intel Core i5-3570 o frekvenci 3.40 GHz, se doba načtení záznamu evoluce o velikosti 200 MB pohybuje kolem 8 s, ale operace prováděné s již načteným záznamem, jako je vzorkování nebo zjištění historie, jsou provedeny bez jakékoliv uživatelem postřehnutelné prodlevy. V závislosti na velikosti vstupního souboru může být omezujícím faktorem velikost operační paměti.

Počet řádků a sloupců hradel není přímo omezen, existuje však omezení počtu elementů obvodu (tj. počet vstupů, výstupů a hradel) na 32 767, což je maximální hodnota datového typu `short`, který byl použit pro označení jednotlivých elementů obvodu. Spíše než velikostí tohoto datového typu je tak velikost obvodu omezena dostupnou zobrazovací plochou. Aplikace podporuje pouze hradla se dvěma vstupy.

Kapitola 8

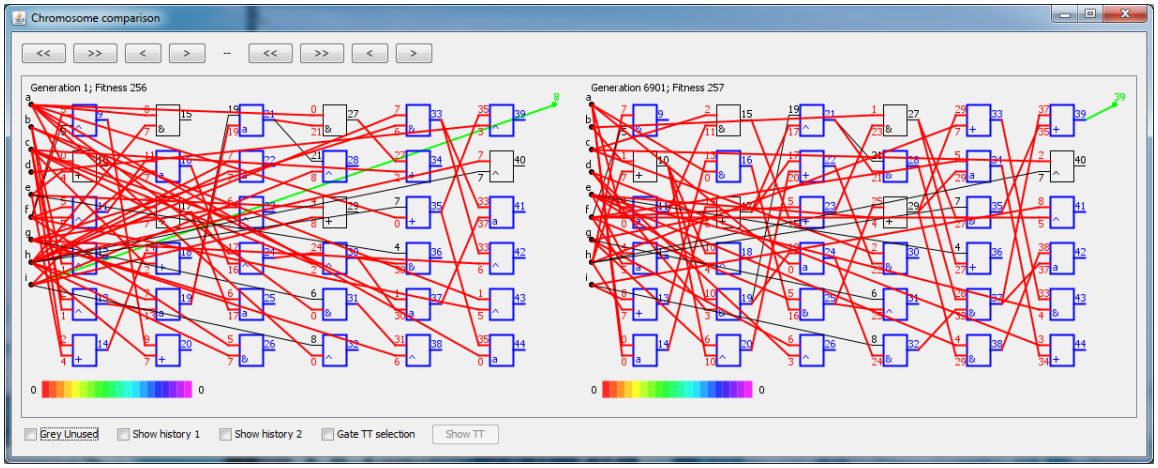
Analýza záznamů evoluce

V této kapitole je vytvořená aplikace využita pro analýzu zvolených záznamů o průběhu evoluce číslicového obvodu. Konkrétně jsem pomocí implementace CGP Zdeňka Vašíčka v sadě Tools4CGP [24] vygeneroval několik záznamů o průběhu evoluce devíti-bitové parity. Počet generací jsem nastavil na 10 000, rozměry pole hradel jsem zvolil 6×6 a evoluce může používat čtyři logické funkce: „a“ (funkce, jejíž výstup je identický jako první vstup), „and“, „or“ a „xor“. Maximální dosažitelná hodnota fitness je 512, protože má obvod devět vstupů a jeden výstup. Hodnota na výstupu je rovna 1, pokud je počet jedniček na vstupu lichý. Pokud je na vstupu sudý počet jedniček, objeví se na výstupu paritního obvodu nula. Z vygenerovaných záznamů jsem vybral tři úspěšné, které jsou níže analyzovány pomocí vytvořené aplikace. Pro ilustraci postupu práce a analýzy záznamu jsem použil obrázky z prvního záznamu.

Obrázek 8.1: Vývoj hodnoty fitness prvního běhu evoluce.

Na obrázku 8.1 je vývoj hodnoty fitness nejlepších chromozomů. Práh pro zobrazení změny fitness v grafu je nastaven na hodnotu 1, což znamená, že jsou zobrazeny všechny změny hodnoty fitness. Můžeme si všimnout, že již první generace má hodnotu fitness 256. Výstup obvodu je připojen k jednomu ze vstupů a výsledek je proto v polovině případů správný. Zajímavé ovšem je, že první změna fitness nastala až v generaci 6901 a to na hodnotu 257, tedy vyrostla pouze o 1. Prvním krokem je tedy srovnání obvodů z generací 1 a 6901, které je na obrázku 8.2. Volba „Grey Unused“ je záměrně ponechána „nezaškrtnutá“, aby bylo zřejmé, že se jedná o dva naprosto odlišné obvody. Z toho lze vyvodit, že počáteční

obvod nebyl příliš vhodný a jednalo se spíše o „slepou uličku“, ze které se evoluce snažila najít cestu, což trvalo vzhledem k celkové délce evoluce velmi dlouho – 6900 generací.



Obrázek 8.2: Srovnání obvodů z generací 1 a 6901.

Tuto teorii potvrzuje obrázek 8.3, který využívá funkci historie pro chromozom z generace 6901, ve kterém došlo k prvnímu zlepšení hodnoty fitness. Jak si můžeme všimnout, poslední změna v tomto obvodu nastala před 397 generacemi, což je relativně „brzy“. Navíc, takto „stará“, jsou pouze dvě hradla. Ostatní části chromozomu jsou mnohem „novější“. Původní obvod byl z velké části změněn aniž by došlo ke změně fitness. V tomto intervalu docházelo k tzv. neutrálním mutacím a pro evoluci bylo zřejmě velmi obtížné dospět ke zlepšení, protože nebylo přítomno „vedení“ průběhu evoluce určitým směrem prostřednictvím výběru jedinců s vyšší hodnotou fitness.

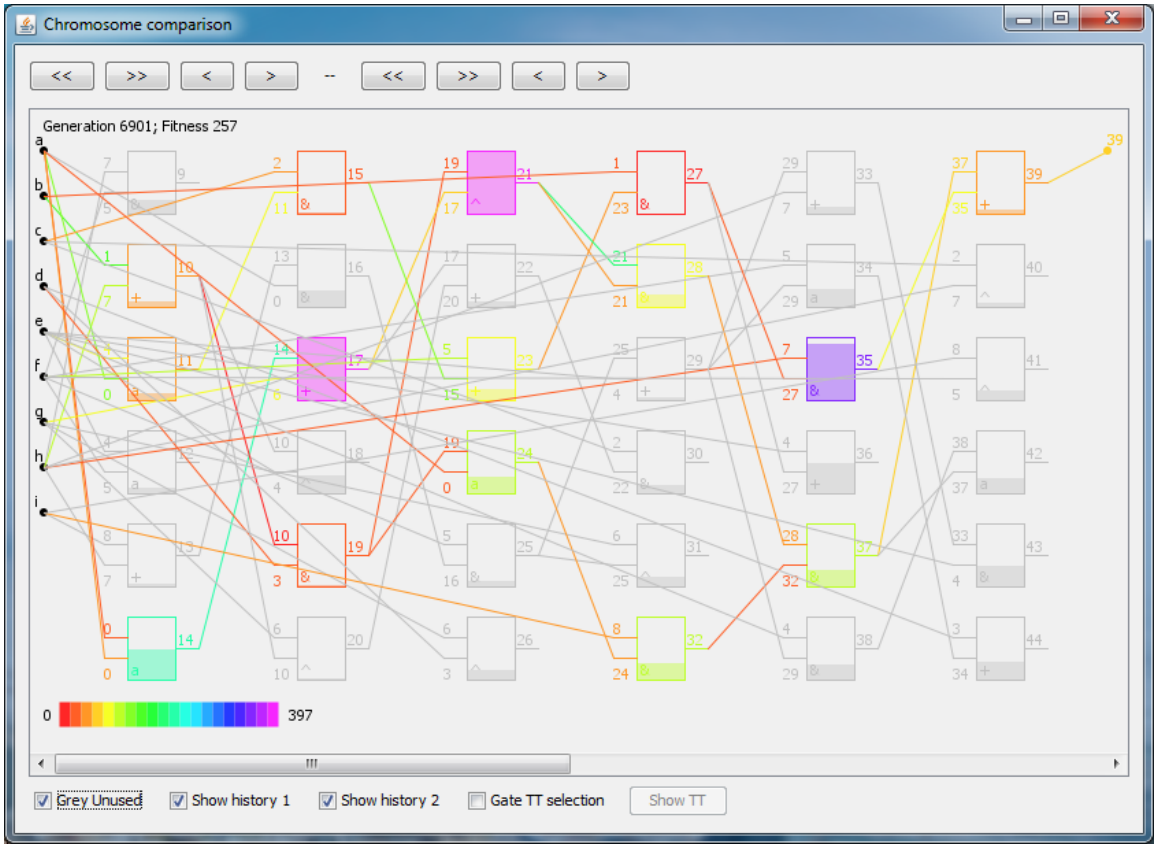
Po první změně hodnoty fitness v generaci 6901 na 257 došlo k poměrně rychlému vývoji, který se dočasně pozastavil v generaci 7189 na hodnotě fitness 320. Mezi těmito generacemi docházelo k nárůstu fitness především změnou funkce aktivního hradla. Za zmínku stojí změna mezi generacemi 7107 a 7108, kde došlo k aktivaci dosud nevyužité části obvodu a zvýšení hodnoty fitness z 272 na 288, jak je ukázáno na obrázku 8.4. Po posledním zvýšení fitness v generaci 7189 došlo ke stagnaci, podobně jako na počátku evoluce, která byla přerušena až v generaci 9392.

Mezi generacemi 9392 a 9393 došlo ke skokové změně hodnoty fitness z 320 na 384. Tato změna byla ovšem způsobena změnou funkce jediného hradla, jak je patrné z obrázku 8.5. Nejednalo se tedy o změnu způsobenou například aktivací dosud nepoužité části obvodu.

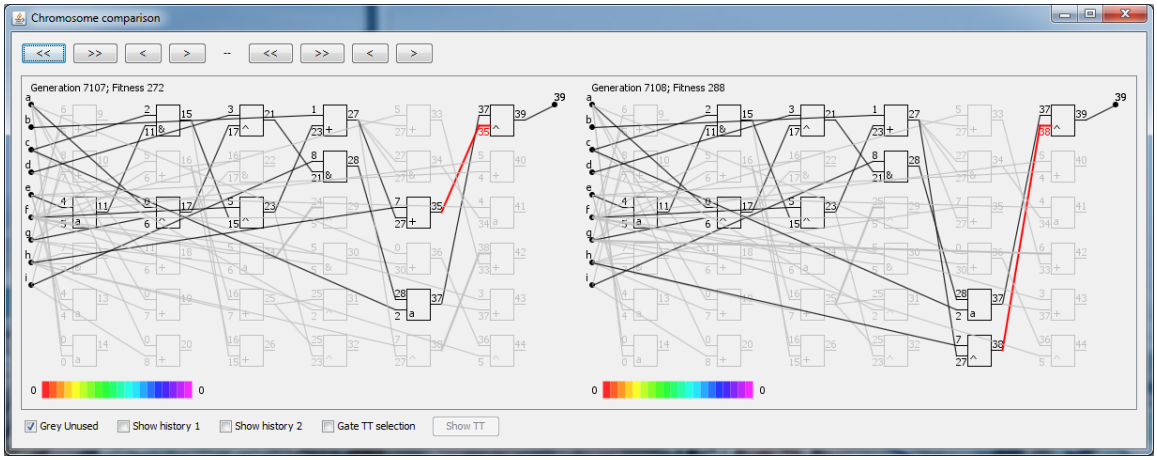
K další velké změně hodnoty fitness došlo mezi generacemi 9399 a 9400, přičemž se fitness změnila z hodnoty 384 na 512, takže vznikl již plně funkční obvod. S využitím porovnávacího okna je z obrázku 8.6 zřejmé, že opět došlo pouze ke změně funkce jediného hradla.

Potenciálně zajímavý by mohl také být vývoj obvodu mezi výše zmíněnými „skokovými“ změnami hodnoty fitness. Jak se tedy změnil obvod mezi generacemi 9393 (po první změně fitness na hodnotu 384) a 9399 (před druhou změnou fitness z hodnoty 384 na 512)? Obvod se měnil pouze v nevyužité části – aktivní část obvodu zůstala nezměněna. Obrázek 8.7 srovnává generaci 9392, po které nastala první velká změna fitness s generací 9400, ve které vznikl plně funkční obvod a ukazuje, že stačila pouze změna funkce dvou hradel.

Po dosažení maximální hodnoty fitness (v tomto případě 512), evoluce obvodu neskončila, ale pokračovala v hledání funkčně ekvivalentního obvodu s menším počtem hradel.



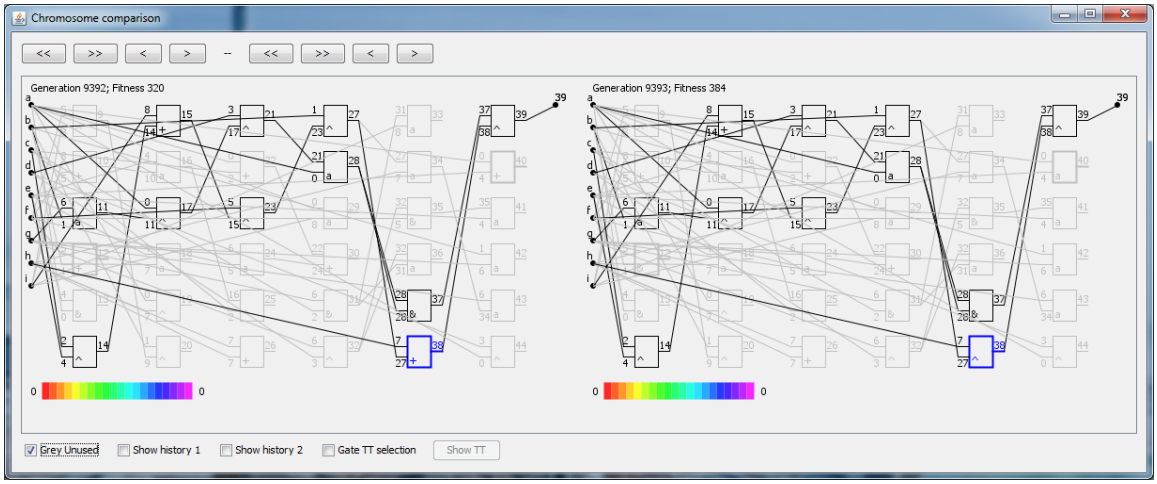
Obrázek 8.3: Historie chromozomu generace 6901.



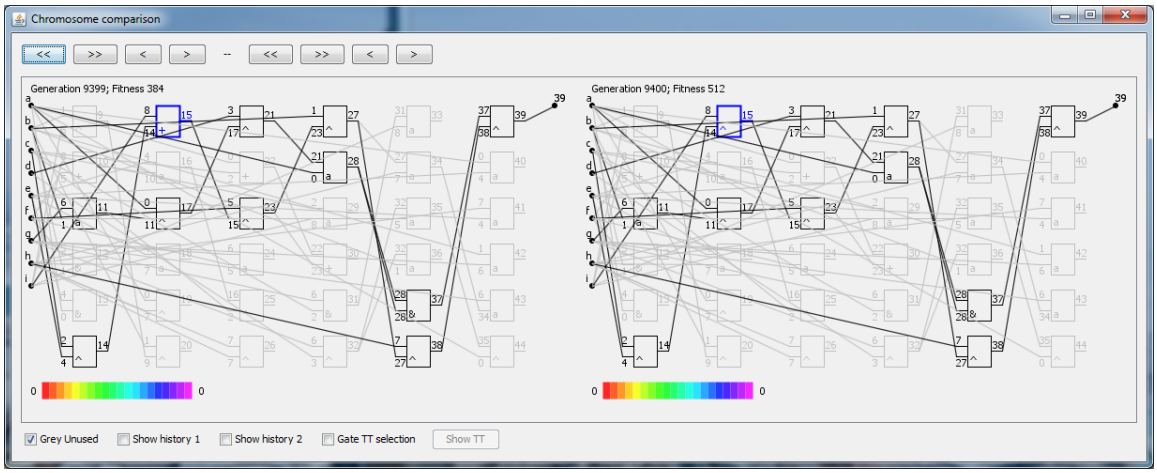
Obrázek 8.4: Aktivace nevyužité části obvodu.

Bylo dosaženo snížení počtu hradel z 11 na 10. Jak je ukázáno na obrázku 8.8, značná část funkčního obvodu je tzv. konzervovaná, tzn. že změny použitých hradel a spojů byly provedeny před relativně velkým množstvím generací a evoluci je „umožněno“ provádět změny především v nevyužité části chromozomu – viz obr. 8.9. Tyto konzervované části vznikly právě v období evoluce, kdy došlo k prvním změnám fitness.

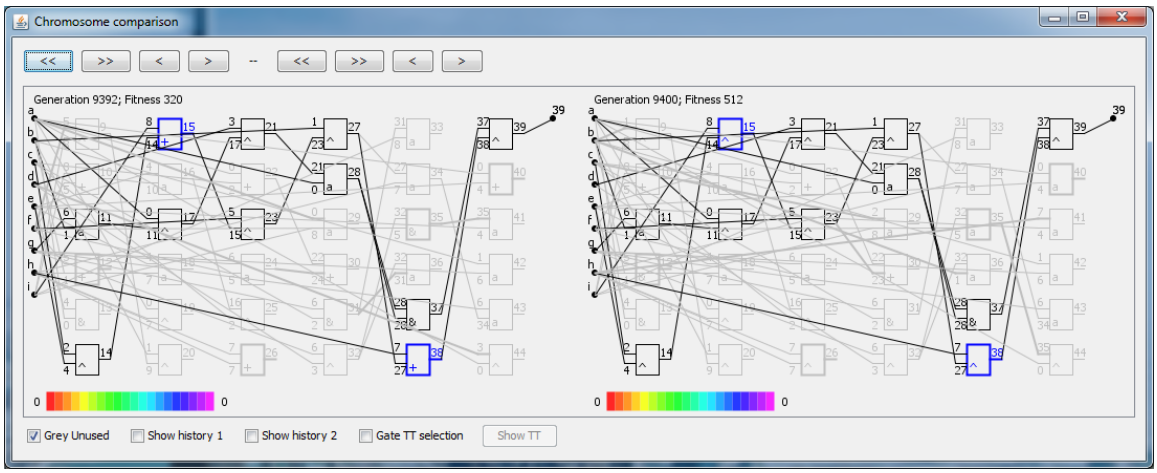
Vezmeme-li v úvahu statistiku využití funkcí hradel, je použití všech funkcí v celé evoluci



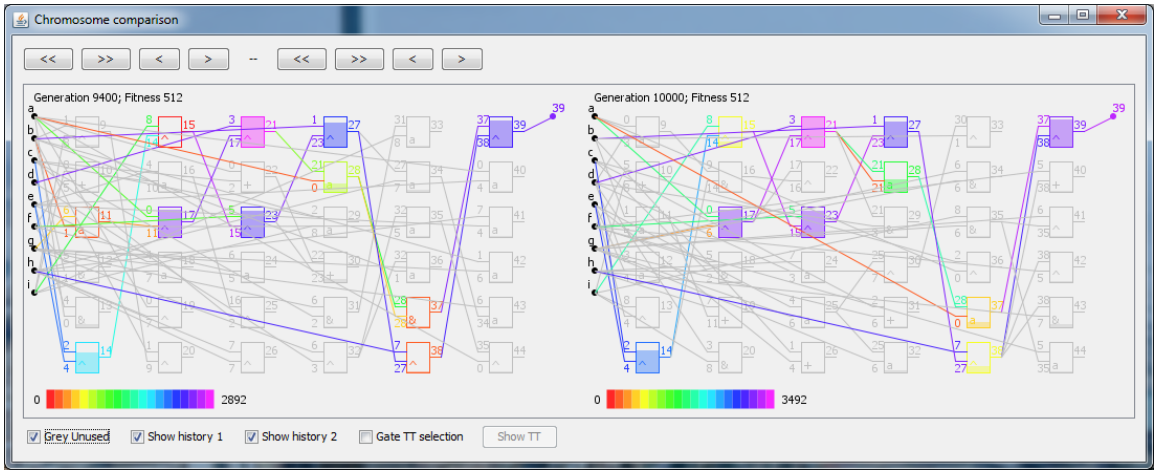
Obrázek 8.5: Srovnání chromozomů generací 9392 a 9393 – změna funkce jediného hradla.



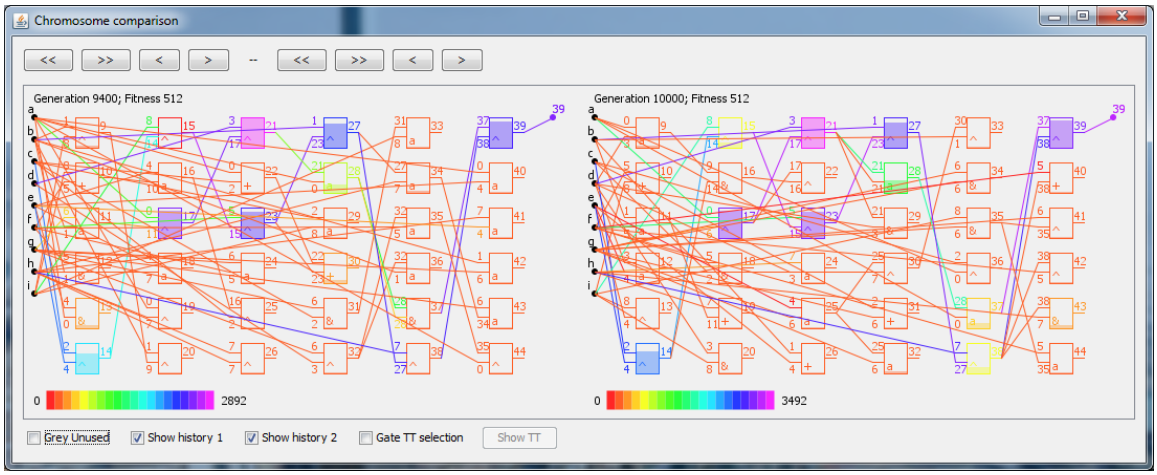
Obrázek 8.6: Srovnání chromozomů generací 9399 a 9400 – změna funkce jediného hradla.



Obrázek 8.7: Srovnání chromozomů generací 9392 a 9400 – změna funkce dvou hradel.



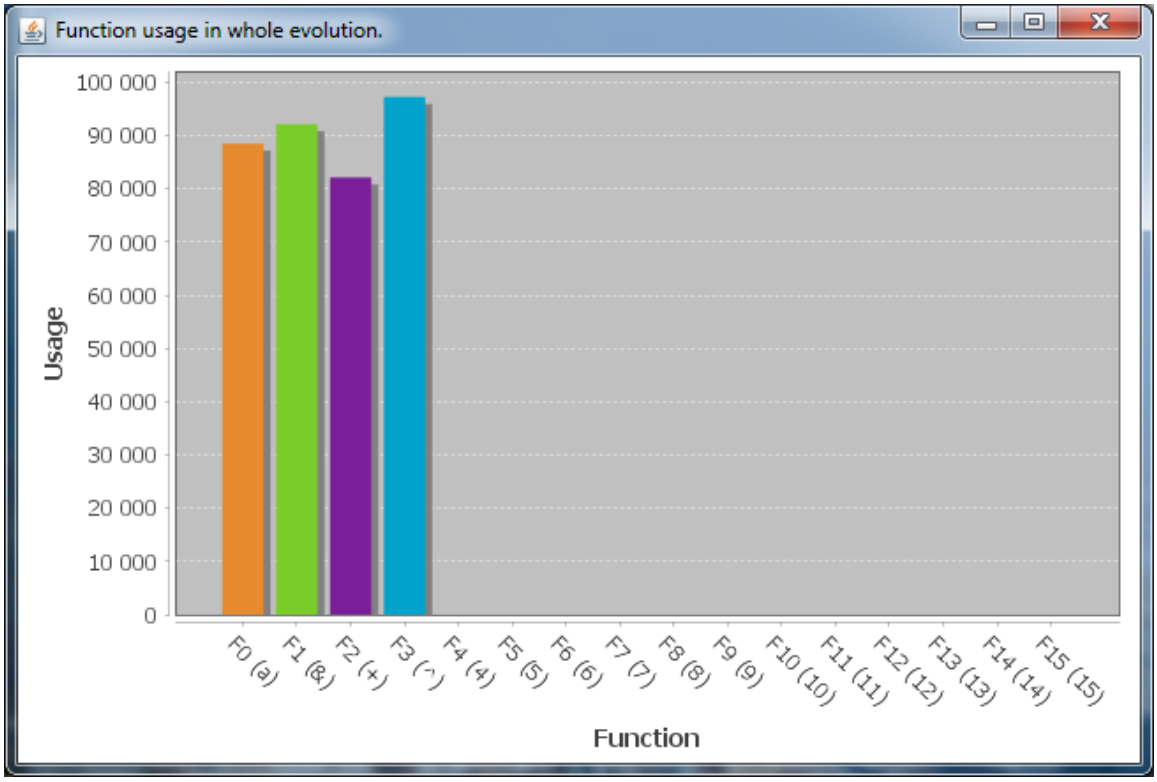
Obrázek 8.8: Historie využité části chromozomů z generací 9400 a 10 000.



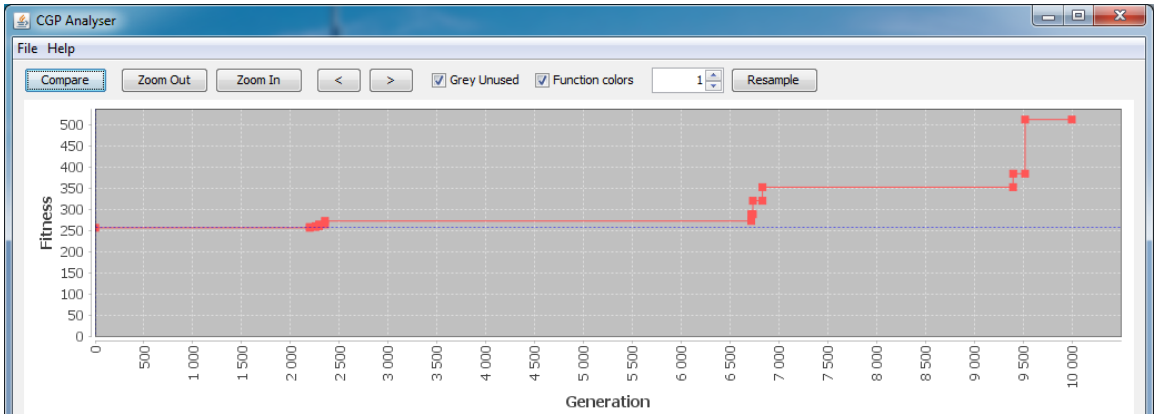
Obrázek 8.9: Historie nevyužité části chromozomů z generací 9400 a 10 000.

relativně vyrovnané, o čemž se můžeme přesvědčit na obrázku 8.10. Nejvíce je používána funkce „xor“. Funkce „xor“ převažuje i ve využité části plně funkčních obvodů, jak se lze přesvědčit například na obrázku 8.7.

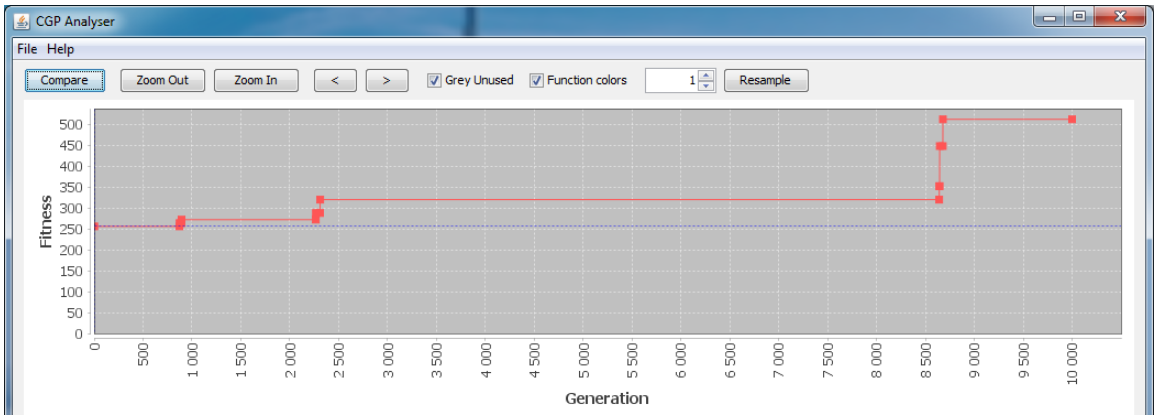
Evoluce druhého a třetího obvodu, jejichž vývoj hodnoty fitness je zobrazen na obrázcích 8.11 a 8.12, probíhala velmi podobně jako výše popsaná evoluce. V evoluci můžeme opět vidět relativně krátké oblasti z růstu fitness oddělené poměrně dlouhou dobou stagnace. Ve výsledných funkčních obvodech jsou části, jejichž poslední změny sahají až k oblastem prvních zlepšení hodnoty fitness. Využití funkcí hradel je podobné jako v prvním běhu, ale funkce „xor“ je ještě více dominantní.



Obrázek 8.10: Statistika využití hradel.



Obrázek 8.11: Vývoj hodnoty fitness druhého běhu evoluce.



Obrázek 8.12: Vývoj hodnoty fitness třetího běhu evoluce.

Kapitola 9

Závěr

Byly popsány stochastické optimalizační metody inspirované přírodou, které využívají populaci jedinců – konkrétně evoluční algoritmy. Podrobně bylo rozebráno kartézské genetické programování a byly zmapovány a navrženy nové metody pro vizualizaci průběhu evoluce CGP. Dále byly popsány existující nástroje pro vizualizaci průběhu evoluce, ale protože nebyl nalezen nástroj umožňující podrobnou analýzu celé evoluce, byl proveden návrh nové aplikace pro tento účel. Navržená aplikace byla implementována ve formě interaktivního nástroje s grafickým uživatelským rozhraním v jazyce Java. Funkce vytvořené aplikace byly detailně popsány, vysvětleny a poté použity k analýze zvolených evolučních záznamů. K možným rozšířením patří zpracování a možnost analýzy nepoužitých jedinců každé generace nebo přidání funkcí pro analýzu úseků evoluce, kde nedochází ke změnám hodnoty fitness.

Literatura

[1] Bäck, T.; Schwefel, H.-P.: An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, ročník 1, č. 1, 1993: s. 1–23.

[2] Benson, C.; aj.: *GNOME Human Interface Guidelines 2.2.3* [online]. `https://developer.gnome.org/hig-book/3.12/`, 2011 [cit. 2014-12-27].

[3] Bloch, J.: *Java efektivně: 57 zásad softwarového experta*. Grada, 2001, ISBN 80-247-0416-1.

[4] Boetticher, G. D.; Kaminsky, K.: The Assessment and Application of Lineage Information in Genetic Programs for Producing Better Models. In *Information Reuse and Integration, 2006 IEEE International Conference on*, 2006, s. 141–146.

[5] Burke, E. K.; Gustafson, S.; Kendall, G.; aj.: Is increased diversity in genetic programming beneficial? an analysis of the effects on performance. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, 2003, str. 1398–1405.

[6] Burlacu, B.; Affenzeller, M.; Kommenda, M.; aj.: Visualization of genetic lineages and inheritance information in genetic programming. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material Proceedings*, 2013, s. 1351–1358.

[7] Dong, H.-B.; Chen, J.: Improved Genetic Programming Based on Lineage Information. In *Management and Service Science, 2009. MASS '09. International Conference on*, 2009, s. 1–5.

[8] Eckel, B.: *Myslíme v jazyku Java: knihovna zkušeného programátora*. Grada, 2001, ISBN 80-247-0027-1.

[9] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer, 15 vydání, 2003, ISBN 978-3-642-07285-7, 300 s.

[10] Eiben, A. E.; aj.: Multi-parent Recombination. In *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, 1997, s. 3–3.

[11] Essam, D.; Mckay, R. I.: Heritage diversity in genetic programming. In *5th International Conference on Simulated Evolution and Learning*, 2004.

[12] Gajda, Z.; Sekanina, L.: *Recent Advances in Evolutionary Synthesis and Optimization of Ordinary and Polymorphic Circuits*. Brno: Faculty of Information Technology, Brno University of Technology, první vydání, 2011, ISBN 978-80-214-4417-1, 102 s.

[13] Gilbert, D.: JFreeChart. online.
URL http://www.jfree.org/jfreechart/

[14] Kennedy, J.; Eberhart, R. C.: *Swarm Intelligence*. San Francisco: Morgan Kaufmann Publishers, první vydání, 2001, ISBN 1-55860-595-9, 512 s.

[15] Mach, M.: *Evolučné algoritmy*. Košice: elfa s.r.o., prvé vydání, 2009, ISBN 978-80-8086-123-0, 250 s.

[16] McPhee, N. F.; Hopper, N. J.: Analysis of genetic diversity through population history. In *Proceedings of the Genetic and Evolutionary Computation Conference*, ročník 2, 1999, str. 1112–1120.

[17] Miller, J. F.: *Cartesian Genetic Programming*. Springer, první vydání, 2011, ISBN 978-3-642-17309-7, 346 s.

[18] Miller, J. F.; Smith, S. L.: Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, ročník 10, č. 2, 2006: s. 167–174.

[19] Pecinovský, R.: *Myslíme objektově v jazyku Java*. Grada, 2008, ISBN 978-80-247-2653-3.

[20] Pedroni, E.: JCGP. online, 2014.
URL https://bitbucket.org/epedroni/jcgp/

[21] Sekanina, L.; Pták, O.; Vašíček, Z.: Cartesian Genetic Programming as Local Optimizer of Logic Networks. In *2014 IEEE Congress on Evolutionary Computation*, IEEE Computational Intelligence Society, 2014, ISBN 978-1-4799-1488-3, s. 2901–2908.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=10504

[22] Sekanina, L.; aj.: *Evoluční hardware*. Praha: Academia, první vydání, 2009, ISBN 978-80-200-1729-1, 328 s.

[23] Staurovská, J.: Nástroj pro vizuální analýzu evoluce obvodů, diplomová práce, Brno, FIT VUT v Brně. 2012.

[24] Vašíček, Z.: Nástroje pro kartézské genetické programování Tools4CGP. online, 2008.
URL http://www.fit.vutbr.cz/~vasicek/cgp/tools/

[25] WWW stránky: The Java Tutorials.
`http://docs.oracle.com/javase/tutorial/index.html`.

Příloha A

Obsah CD

CD obsahuje následující adresářovou strukturu:

- aplikace – spustitelný soubor a potřebné knihovny,
- manuál – pdf verze manuálu k aplikaci,
- text práce – pdf verze textové části a zdrojové soubory pro systém LaTeX,
- ukázkové soubory – záznamy o evoluci a soubor s funkcí hradel pro vyzkoušení,
- zdrojové soubory – zdrojový kód aplikace.