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 STATISTICKÉ VYHODNOCENÍ KOEVOLUČNÍCH ALGORITMŮ

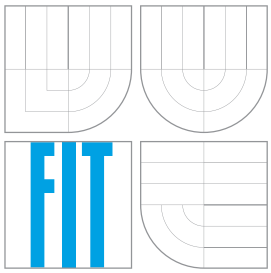
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL URBAN

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 STATISTICKÉ VYHODNOCENÍ KOEVOLUČNÍCH ALGORITMŮ

COEVOLUTIONARY ALGORITHMS STATISTICAL ANALYSIS TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL URBAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAELA ŠIKULOVÁ

BRNO 2015

Abstrakt

V této bakalářské práci se nachází teoretický základ, který nás seznámí s evolučními algoritmy, genetickým programováním, koevolučními algoritmy a metodami pro jejich statistické vyhodnocení. Dále se tato práce zabývá návrhem a implementací nástroje s grafickým uživatelským rozhraním, který umožnuje analýzu koevolučního algoritmu pro různá nastavení parametrů a jeho statistické vyhodnocení. Funkčnost implementovaného nástroje byla testována na datech získaných z externího programu umožňujícího evoluční návrh obrazových filtrů s využitím koevoluce prediktorů fitness. Výsledné grafy a statistiky umožnují jednoduché porovnávání průběhů a výsledků jednotivých běhů programu.

Abstract

This bachelor thesis contains a theoretical basis that introduces evolutionary algorithms, genetic programming, coevolutioanary algorithms and methods for statistical evaluation. Furthermore, this work deals with the design and implementation of tool with graphical user interface, which allows the analysis of coevolutioanary algorithm for various parameters and also its statistical evaluation. The functionality of the implemented tool has been tested on data obtained from an external program performing evolutionary design of image filters with the use of the coevolution of fitness predictors. The resulting graphs and statistics allow easy comparison of the progress and results for each program run.

Klíčová slova

Evoluční algoritmy, koevoluční algoritmy, genetický algoritmus, kartézské genetické programování, grafy, statistiky.

Keywords

Evolutionary algotithms, coevolutioanary algorithms, genetic algorithm, cartesian genetic programming, graphs, statistics.

Citace

Daniel Urban: Nástroj pro statistické vyhodnocení
koevolučních algoritmů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Nástroj pro statistické vyhodnocení koevolučních algoritmů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michaely Šikulové.

.....
Daniel Urban
14. května 2015

Poděkování

Tímto bych chtěl poděkovat své vedoucí Ing. Michaele Šikulové za odbornou pomoc při tvorbě této práce. Dále za kvalitní a časté konzultace, během nichž jsem dostal mnoho rad, doporučení a připomínek k práci. Také bych chtěl poděkovat Bc. Michalovi Wiglaszovi za poskytnutí programu, který vrací výsledná data pro implementován nástroj.

Obsah

Kapitola 1

Úvod

V dnešnej dobe sa snažíme čoraz viac zapájať stroje do nášho života. Vednou disciplínou, ktorá sa týmto problémom zaoberá je umelá inteligencia. Snahou umelej inteligencie je napodobniť sofistikované využívanie vedomostí v neživých organizmoch, predovšetkým v počítačových systémoch. Pre riešenie rôznych príkladov umelej inteligencie sa experimentuje s rôznymi algoritmami, metódami a ich kombináciami, ako sú napríklad neurónové siete, fuzzy logika, evolučné algoritmy a iné.

Táto bakalárska práca sa bližšie venuje evolučným algoritmom, ktoré predstavujú širšiu skupinu algoritmov, ktorej základnou vlastnosťou je inšpirácia princípmi z prírody.

Cieľom tejto práce je zoznámiť sa s princípmi evolučných algoritmov, genetického programovania, koevolučných algoritmov a metódami pre ich štatistické vyhodnocovanie. Po porozumení týchto princípov je potrebné navrhnúť a implementovať nástroj s grafickým užívateľským rozhraním, ktorý umožní analýzu koevolučného algoritmu pre rôzne nastavenia parametrov a jeho štatistické vyhodnotenie. Následne je potrebné na vhodných príkladoch demonštrovať činnosť tohto nástroja a zhodnotiť dosiahnuté výsledky.

Táto práca je rozdelená na šesť kapitol. Kapitola 2 sa venuje teórii potrebnej k správnemu pochopeniu evolučných algoritmov. Obsahuje podrobnejšie vysvetlenie evolučných algoritmov a pojmov s nimi spojenými, ako napríklad genetický algoritmus 2.1, genetické programovanie 2.2, kartézske genetické programovanie 2.3 a koevolučné algoritmy 2.4.

V kapitole 3 je popísaný návrh nástroja pre analýzu koevolučného algoritmu. Obsahuje požiadavky pre tento nástroj, spracovanie výstupov jedného či viacerých behov a ovládanie programu pre koevolučný návrh, čo zahŕňa nastavenie parametrov a nastavenie experimentov.

Implementácia je podrobnejšie popísaná v kapitole 4, kde je okrem zvolených nástrojov, knižníc a riešených problémov popísaná aj nadväznosť implementovaného nástroja na už implementovaný program umožňujúci prevádzať evolučný návrh obrazových filtrov s využitím koevolúcie prediktorov fitness.

Kapitola 5 popisuje, na vhodne zvolených príkladoch, činnosť nástroja v úlohe koevolučného návrhu obrazových filtrov.

Zhodnotenie dosiahnutých výsledkov a možné návrhy ďalšieho vývoja práce sú zhrnuté v záverečnej kapitole 6.

Kapitola 2

Evolučné algoritmy

Pomocou techník hľadania v prehľadávacom priestore sa riešia mnohé úlohy umelej inteligencie. Prehľadávací priestor reprezentuje všetky riešenia danej úlohy. V prípade, že je tento priestor malý, je možné preskúmať všetky riešenia a nájsť globálne optimum. Rozsiahle priestory sú prehľadávané rôznymi stochastickými algoritmami, medzi ktoré sa zaraďujú aj evolučné algoritmy.

Termín evolučný algoritmus (EA) vznikol začiatkom 90. rokov. Zastrešuje stochastické prehľadávacie algoritmy, ktoré nachádzajú inšpiráciu v Darwinovej evolučnej teórii a taktiež v rôznych teóriách neodarwinizmu. Hlavnou myšlienkou Darwinovej evolučnej teórie je predávanie rodičovského génu novým potomkom a následné uvoľnenie životného priestoru potomkom. Evolučné algoritmy sú trieda veľmi výkonných algoritmov, ktoré umožňujú riešiť zložité problémy efektívnym spôsobom. Vďaka schopnosti automatizovane riešiť optimalizačné problémy sa stali veľmi obľúbenými a používanými v mnohých inžinierskych odboroch.

Spočiatku boli evolučné algoritmy využívané hlavne k nájdeniu optimálnych hodnôt parametrov optimalizovaného systému, ale v posledných rokoch sa do popredia dostáva využitie evolučných algoritmov v úlohách návrhu. Tu sa hľadá okrem parametrov aj vlastná štruktúra cieľového systému [7, 10].

Pojmy

Predtým, ako bude možné začať pracovať s Evolučnými algoritmami, je potrebné definovať niekoľko základných pojmov, ktoré s nimi úzko súvisia. Pojmy a ich význam, čerpané z literatúry [7, 10], sú voľne inšpirované biologickou predlohou EA - biologickou evolúciou.

- **Gén** - základný prvok chromozómu. Často je to znak, ktorý je prvkom vopred definovanej abecedy a môže obsahovať v podstate čokoľvek, čo počítač umožní reprezentovať. Najčastejšie sa používa binárna, celočíselná alebo reálna reprezentácia.
- **Chromozóm** - tento pojem sa používa pre zakódovaného jedinca. Je to reťazec génov, ktoré majú premennú alebo pevnú dĺžku.
- **Kandidátne riešenie** - jedinec so svojimi vlastnosťami a prejavmi.
- **Genotyp** - pojem, ktorý sa používa v oblasti evolučných algoritmov pre zakódovaného jedinca. Skladá sa z jedného alebo viacerých chromozómov.
- **Fenotyp** - kandidátne riešenie skonštruované pomocou genotypov.

- **Populácia** - množina kandidátnych riešení. Jedinci v populácií sa v priebehu evolúcie menia. Jedna iterácia zmeny populácie sa nazýva *generácia.*
- **Fitness** - ohodnotenie jedinca z hľadiska evolúcie. Táto hodnota určuje schopnosť konkrétneho jedinca splniť požadované vlastnosti.
- **Fitness funkcia** - každému jedincovi priradí hodnotu fitness, pomocou ktorej môžu byť dvaja jedinci medzi sebou porovnávaní, ktorý z nich je kvalitnejší, to znamená, ktorý lepšie spĺňa požadovanú vlastnosť.

Princíp evolučného algoritmu

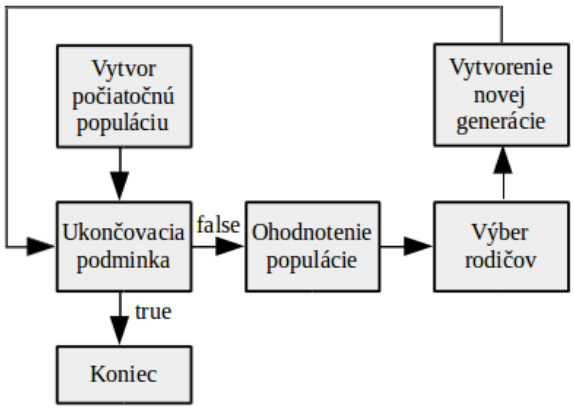
Na začiatku výpočtu evolučného algoritmu je potrebné vytvoriť počiatočnú populáciu, ktorá bude obsahovať dopredu zvolený počet kandidátnych riešení. Vytvorenie počiatočnej populácie prebieha pomocou vhodnej heuristiky, ktorá môže využívať už známe riešenia problémov alebo je populácia vytvorená úplne náhodne. Vo vytvorenej populácii je potrebné priradiť všetkým jedincom hodnotu fitness alebo ich ohodnotiť pomocou fitness funkcie.

Na základe fitness sa vyberú vhodní jedinci z predchádzajúcej populácie, ktorí vytvoria množinu rodičov pre novú populáciu. Na túto množinu sú aplikované genetické operátory, ako napríklad mutácia alebo kríženie. Pomocou nich vzniká množina potomkov a následne sa z týchto množín vyberajú jedinci pre novú populáciu. Výber rodičov a vytvorenie novej populácie prebieha pomocou rôznych selekčných mechanizmov.

Selekčné mechanizmy:

- **Proporcionálna selekcia** - pri tejto selekcii je pravdepodobnosť výberu priamo úmerná absolútnej hodnote fitness jedinca. Implementácia prebieha pomocou algoritmu rulety. Výber prebieha vygenerovaním náhodného čísla z intervalu $[0, 1]$, pomocou ktorého je identifikovaný jeden takzvaný úsek rulety a tým aj zodpovedajúci jedinec. Ak je vyberaných x jedincov, tento krok opakujeme x-krát. V prípade tejto selekcie môže nastať aj prípad výberu nekvalitného jedinca. Problém nastáva aj v prípade, keď jeden jedinec má veľmi vysoké ohodnotenie - populácia zdegeneruje, pretože pravdepodobnosť výberu tohto jedinca je príliš vysoká a je vyberaný častejšie.
- **Turnajová selekcia** - z populácie sa náhodne vyberie a porovnáva x jedincov. Typicky je x 2 až 8 jedincov. Víťaza turnaja predstavuje jedinec, ktorý má najvyššiu hodnotu fitness a postupuje do ďalšieho kroku algoritmu. Proces je opakovaný, kým nie je dosiahnutý počet jedincov, ktorí majú byť vybraní.
- **Deterministická selekcia** - k reprodukcii je deterministicky vybraných x jedincov s najvyššou hodnotou fitness. Jedná sa o najjednoduchší prípad selekcie.

Z vybraných rodičov sú následne vytvorení potomkovia. To sa deje pomocou rekombinačných operátorov. Medzi dva najzákladnejšie rekombinačné operátory patrí kríženie a mutácia. Evolučný algoritmus je ukončený, ak je nájdený dostatočne kvalitný jedinec, alebo ak je vyčerpaný zadaný počet generácií [7].

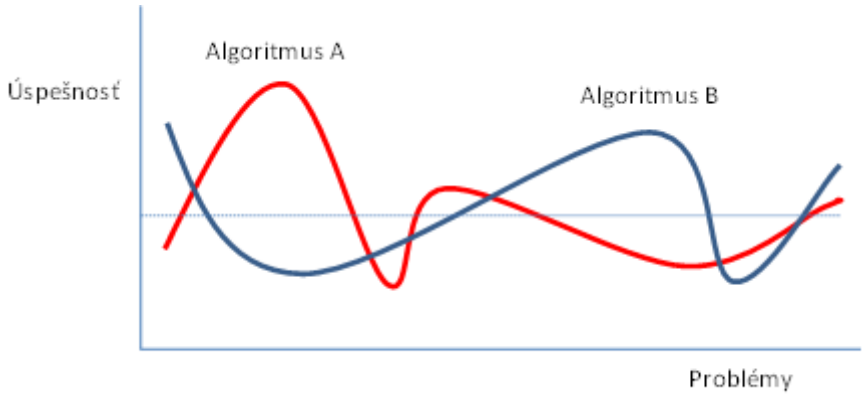


Obrázok 2.1: Evolučný algoritmus.

No free Lunch Teorém

Jedná sa o teorém, v ktorom sa tvrdí, že neexistuje algoritmus, ktorý by dokázal riešiť všetky problémy lepšie ako iné algoritmy alebo existuje podmožina problémov, pre ktoré je algoritmus A lepší ako algoritmus B a naopak. Všeobecne platí, že evolučné algoritmy sú v podstate ekvivalentné, lebo ich priemerná úspešnosť, počítaná zo všetkých možných riešených problémov, je rovnaká.

Ako je však znázornené na obrázku 2.2, ich výkon nebude rovnaký na všetkých problémoch. Algoritmy A a B majú rôznu úspešnosť, ale ich priemerná úspešnosť (priamka) je rovnaká. Na základe tohto teorému nie je možné povedať, že na riešenie ľubovoľného problému je možné použiť akýkoľvek algoritmus. Lepším vyjadrením je, že rôzne algoritmy sa hodia k riešeniu rôznych problémov [3].



Obrázok 2.2: NFL v grafickom poňatí.

2.1 Genetický algoritmus

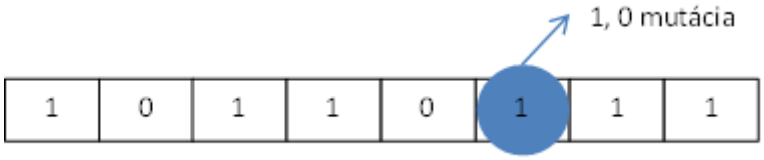
Genetický algoritmus (GA) patrí medzi základné stochastické optimalizačné algoritmy s výraznými evolučnými črtami. Bol navrhnutý v 70. rokoch Johnom Hollandom, ktorý skúmal možnosť adaptácie v umelých systémoch. Populárnym sa stal až v 80. rokoch keď David Goldberg preukázal jeho praktickú užitočnosť v rade obtiažnych úloh.

Základná varianta genetického algoritmu je veľmi podobná evolučnému algoritmu spomínanému v kapitole 2 a zobrazenému na obrázku 2.1. Kandidátny jedinec je reprezentovaný reťazcom génov, ktorý má konštantnú dĺžku a počiatočná populácia je vygenerovaná náhodne. Po výbere jedincov určitým selekčným algoritmom spomínaným v kapitole 2, je na nich aplikované kríženie a mutácia [7, 3].

V prípade GA sa pracuje s dvoma variantami evolučného algoritmu. S *generačnou variantou* pracujeme vtedy, ak je nová generácia vytvorená len z nových vytvorených potomkov. Ak je nová populácia z časti vytvorená z nových vytvorených jedincov a tiež z časti predchádzajúcej populácie, jedná sa o *prekrývanie populácií*. V tomto type GA je potom parametrom počet percent nových jedincov, ktorí boli zaradení do populácie. V prípade, že sa najlepší jedinec z predchádzajúcej populácie vždy dostane do novej populácie, hovorí sa o *elitizme* [7].

Mutácia

Vplyvom operátora mutácie vzniká z vybraného jedinca mutovaný jedinec. Dochádza k náhodnej zmene génu jedinca, ako je zobrazené na obrázku 2.3. Využíva sa najmä ako prostriedok pre dosiahnutie väčšej rozmanitosti v generácii. V prípade GA je skôr doplnkovým operátorom tvorby potomkov.



Obrázok 2.3: Mutácia.

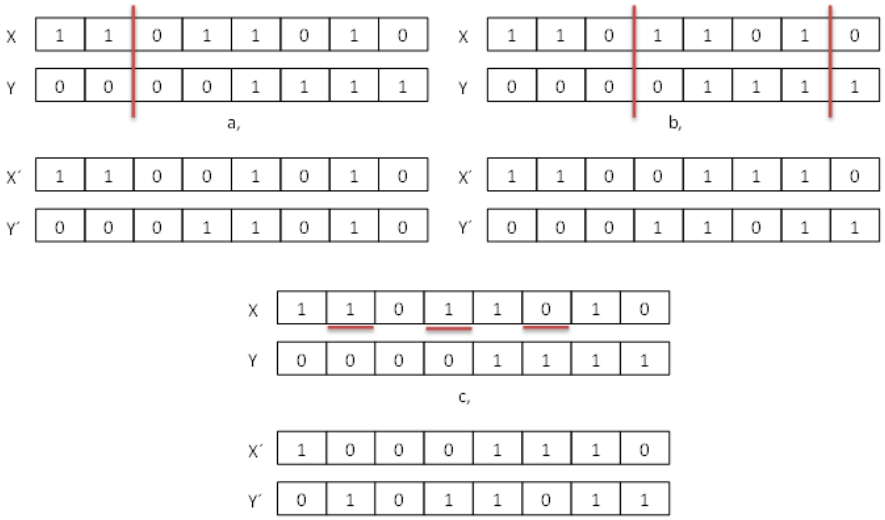
Kríženie

Vplyvom operátora kríženia dochádza k výmene častí génov medzi dvoma alebo viacerými chromozómami. Pri krížení sa využíva časť génov jedného rodiča a časť génov druhého rodiča. U základných variant kríženia vznikajú z dvoch rodičov dvaja potomkovia.

Existujú tri základné varianty kríženia, ktoré sú zobrazené na obrázku 2.4:

- **Jednobodové kríženie** - v tomto prípade kríženia je náhodne vygenerovaný bod kríženia a dôjde k prehodeniu génov, ktoré sa nachádzajú vľavo od bodu kríženia.
- **Viacbodové kríženie** - umožňuje lepšie premiešať genetický materiál medzi rodičmi, keďže je možné definovať viac bodov kríženia.

- **Uniformné kríženie** - o každom géne sa rozhoduje na základe masky či dôjde k jeho výmene.



Obrázok 2.4: Kríženie: a) jednobodové, b) viacbodové, c) uniformné.

2.2 Genetické programovanie

Na prelome 80. a 90. rokov John Koza navrhol originálnu modifikáciu genetického algoritmu, ktorú nazval genetické programovanie (GP).

Ako je uvedené v [8], GP definuje ako systematickú metódu, ktorá umožňuje počítačom automaticky riešiť problém spočívajúci vo výraze na vysokej úrovni, ktorý má byť vyriešený.

Cieľom tejto metódy nie je len evolučne hľadať optimálne hodnoty parametrov, ktoré sú zakódované v chromozóme, ale automaticky generovať celé programy. John Koza implementoval genetické programovanie v jazyku LISP, ktorý má účinné programovacie prostriedky pre kódovanie stromov a manipuláciu s nimi.

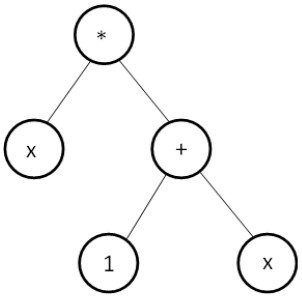
Chromozómy - znakové reťazce sú nahradené zložitejšími štruktúrami - funkciami. V najjednoduchšej verzii genetického programovania sú funkcie reprezentované výrazmi obsahujúcimi premenné, základné aritmetické operácie, elementárne funkcie a konštanty.

Na obrázku 2.5 je pomocou syntaktického stromu reprezentovaná jednoduchá funkcia. Horný vrchol sa nazýva koreň stromu a pri postupe zdola-hore sa od koncových vrcholov smerom ku koreňu stromu postupne vykonáva výpočet funkcie [7, 3].

Základné princípy genetického programovania

Reprezentácia - Genetické programovanie pracuje so spustiteľnými štruktúrami. Najčastejšie sa jedná o reprezentáciu pomocou stromov.

Genetické operátory - pracujú nad spustiteľnými štruktúrami. Existuje množstvo pokročilých operátorov, ktoré umožňujú napríklad evolučne vytvárať podprogramy v rámci vyvíjaných programov.



Obrázok 2.5: Syntaktický strom popisujúci jednoduchú funkciu: $x \cdot (1 + x)$.

Pri zisťovaní fitness hodnoty je vykonaný kód kandidátneho programu pre definovanú množinu vstupov a sú vyhodnotené výsledky.

Pri riešení úlohy pomocou genetického programovania, je pred začiatkom riešenia potrebné vykonať niekoľko prípravných krokov:

- Definovať množinu terminálov.
- Definovať množinu funkcií.
- Definovať spôsob výpočtu fitness.
- Definovať parametre genetického programovania.
- Definovať spôsob určenia výsledku a spôsob ukončenia evolúcie.

Množina terminálov

Množina terminálov reprezentuje vstupy do programu, konštanty a funkcie bez argumentov. Nová konštanta môže byť vytvorená buď kombináciou základných konštánt a aritmetických funkcií, alebo je možné generovať ju úplne náhodne.

Množina funkcií

Množina funkcií môže byť všeobecná alebo špecifická pre oblasť, v ktorej je genetické programovanie použité. Vhodné je zaraďovať menej náročné funkcie kvôli náročnosti výpočtu. Potrebné je definovať aj výstupy funkcií, aby nedošlo k chybám počas behu.

Reprezentácia programov

Programy sú chápané ako štruktúry, ktoré pozostávajú z funkcií a terminálov. Tiež sú definované pravidlá, ktoré určujú, ako budú jednotlivé funkcie a terminály použité. Základné štruktúry, ktoré sa používajú v genetickom programovaní sú stromy, lineárne štruktúry a všeobecné grafové štruktúry. K vykonaniu programu dochádza tým, že sa prechádza strom, ktorý daný program reprezentuje. V poslednej dobe sa v genetickom programovaní začali častejšie používať všeobecné grafy.

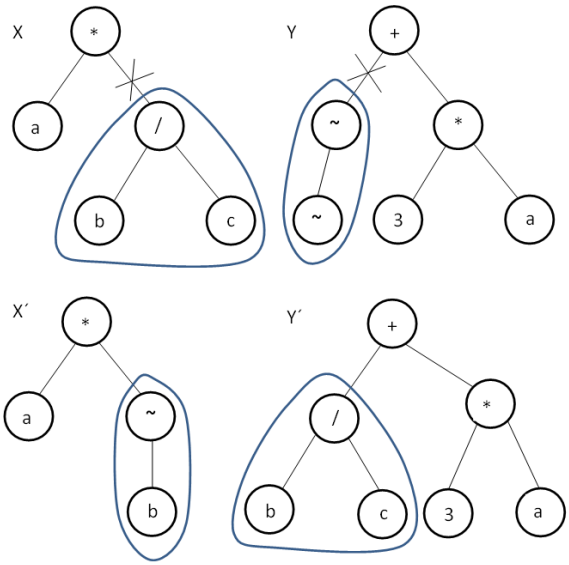
Inicializácia počiatočnej populácie

Počas inicializácie počiatočnej populácie sa do programu náhodne vyberajú terminály a funkcie z množiny symbolov.

Operátory genetického programovania

Za základný operátor genetického programovania sa považuje kríženie, ale tiež sa využíva aj mutácia.

Existujú rôzne varianty kríženia pre stromovo orientované genetické programovanie. Základný operátor kríženia pracuje s dvoma rodičmi, ktorí sú náhodne vybraní pomocou selekcie. Ako je zobrazené na obrázku 2.6, náhodne sa označí uzol, jeho podstrom a prehodí sa označená časť genetického materiálu jedného rodiča X s genetickým materiálom druhého rodiča Y. Vzniknú dvaja noví potomkovia X' a Y'.



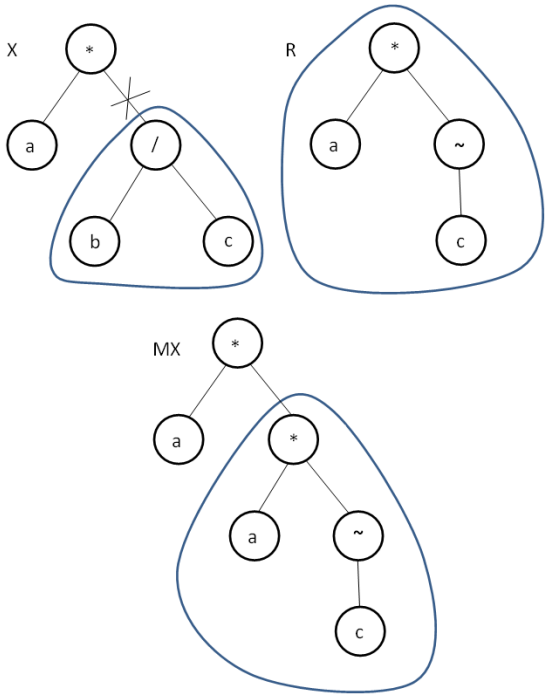
Obrázok 2.6: Kríženie v genetickom programovaní.

Operátor mutácie pracuje s jedným jedincom, kde sa náhodne označí jeden uzol v strome. Tento uzol sa spolu s príslušným podstromom nahradí stromom, ktorý je vygenerovaný. To je zobrazené na obrázku 2.7, kde zo stromu X vzniká použitím náhodne vygenerovaného podstromu R, strom MX.

Fitness funkcia v genetickom programovaní

Každému programu v populácii je pridelená fitness hodnota, ktorá reprezentuje stupeň riešenia požadovanej úlohy daným programom. Táto hodnota je vypočítaná na základe dobre stanoveného postupu. Medzi najviac používané metódy na určenie fitness hodnoty v GP sa radí *hrubá fitness* a *štandardizovaná fitness*.

- **Hrubá fitness** je najjednoduchším a najprirodzenejším spôsobom ohodnotenia, ktoré reprezentuje stupeň riešenia požadovanej úlohy daným programom. Často, ale nie vždy, je hrubá fitness hodnota vyčíslená pomocou množiny *prípadov fitness*. Fitness prípad sa zhoduje s reprezentatívnou situáciou, v ktorej môže byť vyčíslená schopnosť



Obrázok 2.7: Mutácia v genetickom programovaní.

programu riešiť danú úlohu. Napríklad, ak vezmeme do úvahy problém generovania aritmetického výrazu, ktorý aproximuje polynóm $x^4 + x^3 + x^2 + x$ nad množinou prirodzených čísel menších ako 10. Prípadom fitness je potom jedno z týchto prirodzených čísel. Hrubá hodnota fitness f_R jedinca i, vypočítaná nad množinou N prípadov fitness, môže byť definovaná ako:

$$f_R(i) = \sum_{j=1}^{N} |S(i,j) - C(j)|^k \tag{2.1}$$

$S(i,j)$ je hodnota vrátená výpočtom jedinca i nad prípadom fitness j. $C(i)$ je cieľová výstupná hodnota priradená k prípadu fitness j a k je nejaké prirodzené číslo (zvyčajne $k = 1$ alebo $k = 2$). Prípad fitness je typicky malá vzorka z celého doménového priestoru. Koľko a ktoré prípady fitness zvoliť je dôležité vedieť kvôli tomu, aby bolo možné určiť či sú evolované riešenia schopné sa všeobecne prispôsobiť na celý doménový priestor. V praxi je toto rozhodnutie na základe znalostí riešeného problému a na zvážení spotreby výpočetných prostriedkov. Ak je využívaná väčšia trénovacia množina, je na určenie hodnoty fitness potrebný dlhší výpočetný čas [8].

- **Štandardizovaná fitness** je prevedená hrubá fitness hodnota. Menšie numerické hodnoty znamenajú lepšiu kvalitu. Vyplýva z toho, že najlepšia hodnota v štandardizovanej fitness by mala byť rovná nule [8].

2.3 Kartézske genetické programovanie

Kartézske genetické programovanie (CGP) vzniklo z metódy rozvíjajúcej číslicové obvody, ktorú vyvinul v roku 1997 J. F. Miller. CGP však bolo prvýkrát ucelene predstavené až v roku 1999.

CGP reprezentuje kandidátne riešenia vo forme orientovaných acyklických grafov. Tieto grafy sú reprezentované ako dvojrozmerné pole programovateľných elementov (uzlov grafu). V tomto poli je počet stĺpcov reprezentovaný premennou n_c a počet riadkov premennou n_r. Primárne vstupy n_i a výstupy n_o obvodu sú pevne určené ešte na začiatku evolúcie. Funkcia môže obsahovať až n_n argumentov a je vybraná z množiny funkcií Γ. Každý uzol je realizovaný práve jednou takou funkciou.

Vstupy uzlu, ktorý je v danom stĺpci, sú pripojené na primárne vstupy obvodu alebo na výstupy uzlov umiestnených v L predchádzajúcich stĺpcoch, pričom uzly rovnakého stĺpca nesmú byť prepojované. Hodnota L reprezentuje mieru prepojiteľnosti uzlov v obvode, pričom minimálna hodnota L je 1. Primárne vstupy obvodu môžu byť pripojené k ľubovoľnému uzlu v obvode [4, 7].

Zhrnutie parametrov CGP

- n_c počet stĺpcov poľa
- n_r počet riadkov poľa
- n_i počet primárnych vstupov obvodu
- n_o počet primárnych výstupov obvodu
- n_n počet argumentov funkcie
- n_f počet dostupných funkcií
- Γ množina dostupných funkcií
- L (L-back) miera prepojiteľnosti uzlov v obvode

Kódovanie chromozómu v CGP

V kódovaní chromozómu môžu nastať situácie, kedy niektoré uzly nemusia byť vo fenotype využité. Niektoré môžu byť, naopak, využité viac krát. Existujú tiež prípady, kedy nie sú použité všetky vstupy uzlov alebo všetky primárne vstupy vo fenotype.

Tieto prípady pri kódovaní znamenajú redundanciu, keďže dĺžka chromozómu je konštantná, ale veľkosť fenotypu je variabilná [4].

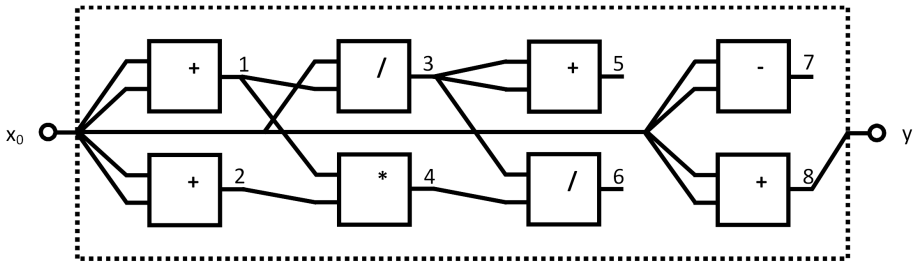
Postup kódovania chromozómu:

- Primárnym vstupným uzlom a výstupným uzlom sú postupne po stĺpcoch priradené indexy, počnúc indexom 0.
- Všetky uzly v obvode sú kódované pomocou $n_n + 1$ celočíselných hodnôt. Posledná hodnota určuje, akú funkciu uzol vykonáva nad svojimi n_n vstupmi.
- Na konci chromozómu sa nachádza n_o hodnôt definujúcich indexy uzlov, ku ktorým sú pripojené primárne výstupy obvodu.

Parametre obvodu uvedeného na obrázku 2.8:
$n_c = 4$, $n_r = 2$, $n_i = 1$, $n_o = 1$, $n_n = 2$, $n_f = 4$, $\Gamma = +(0), -(1), *(2), /(3)$, $L = 3$

Kódovanie chromozómu na obrázku 2.8:
$((0,0,0),(0,0,0),(0,1,3),(1,2,2),(3,3,0),(3,4,3),(0,0,1),(0,0,0),(8))$.

Posledná hodnota chromozómu je zapojenie výstupu. Uzly $5, 6, 7$ sú vo fenotype nevyužité, keďže výstupy nie sú zapojené.



Obrázok 2.8: Kandidátny obvod v CGP.

Operátor CGP

Štandardná metóda CGP využíva jediný typ operátora - mutáciu. Princíp mutácie v CGP je výber náhodného génu a náhodné vygenerovanie jeho novej hodnoty.

Mutácia môže spôsobiť:

- Zmenu funkcie uzla.
- Zmenu pripojenia vstupného signálu k uzlu.
- Zmenu pripojenia primárneho výstupu obvodu.

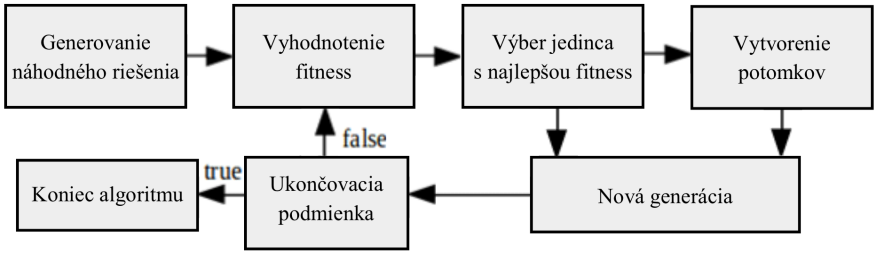
Mutácia taktiež môže spôsobiť, že sa z aktívnych uzlov (uzly, ktoré sú použité vo fenotype) stanú neaktívne alebo naopak. Ak mutácia nemá vplyv na podobu fenotypu, jedná sa o neutrálnu mutáciu. V inom prípade ide o adaptívnu mutáciu [7].

Algoritmus CGP

Typicky pozostáva z nasledujúcich krokov:

- Vygenerovanie náhodného riešenia $(1 + \lambda)$. Typicky je $\lambda = 4(1 + 4)$.
- Ohodnotenie všetkých jedincov pomocou fitness funkcie.
- Výber jedinca s najvyššou fitness hodnotou - rodič.
- Vytvorenie λ mutantov z najlepšieho jedinca pomocou mutácie - potomkovia.
- Rodič spolu s potomkami tvoria novú generáciu.

Ak je splnená ukončovacia podmienka, algoritmus končí. V inom prípade sa pokračuje opäť od ohodnotenia všetkých jedincov pomocou fitness funkcie. Evolúcia končí nájdením dostatočne kvalitného jedinca alebo vyčerpaním povoleného počtu generácií. Grafické znázornenie CGP algoritmu je na obrázku 2.9.



Obrázok 2.9: CGP algoritmus.

2.4 Koevolučné algoritmy

Kľúčovou časťou EA pre riešenie daného problému je fitness funkcia. Od tejto funkcie závisí rýchlosť konvergencie a tiež schopnosť nájdenia riešenia. Pre účely tejto kapitoly bude fitness funkcia popísaná ako funkcia, ktorá priraďuje reálnu hodnotu každému možnému genotypu z množiny všetkých genotypov G. Funkcia bude v tvare $f : G \to \mathbb{R}$ [6].

Pomocou tejto funkcie je možné porovnaním $f(g1)$ s $f(g2)$, kde $g1, g2 \in G$, zistiť vzťah medzi každými dvoma genotypmi. Tým je možné určiť, ktorý z genotypov je vhodnejší pre danú úlohu.

Naopak, v koevolučných algoritmoch (CoEA) sa nepoužíva takéto priame porovnávanie fitness hodnoty jedincov, čo je hlavným rozdielom voči EA. Dvaja jedinci sú porovnávaní na základe výsledku ich interakcie s inými jedincami a vďaka tomu môžu byť rôzne ohodnocovaní v priebehu evolúcie. Fitness hodnota jedincov sa môže v priebehu evolúcie meniť, čo sa v EA nemôže stať. Informácie v tejto kapitole sú čerpané z [11, 6].

Úlohy založené na teste a kompozičné úlohy

V roku 1990 predstavil Hillis úlohu generovania minimálnej radiacej siete pre koevolúciu založenú na testoch. *Úlohy založené na teste* sa vyznačujú tým, že kvalita potenciálnych riešení je určovaná na základe nejakej množiny testov. Hillis využil k riešeniu úlohy generovania minimálnej triediacej siete dve populácie, kde jedna reprezentovala triediace siete a druhá obsahovala prípady testov. Cieľom tohto algoritmu bolo vytvoriť čo najmenšiu sieť, ktorá bola schopná správne roztriediť akýkoľvek daný súbor dát. Touto koevolučnou technikou sa podarilo vytvoriť triediacu sieť, ktorá používa 61 porovnaní pre 16 vstupov. Podobná, ale nekoevolučná technika nedosiahla riešenie lepšej siete ako tej, ktorá používa 63 porovnaní.

Okrem úloh založených na teste sa v CoEA stretávame aj s *kompozičnými úlohami*, kde každý jedinec v každej populácii je komponent. Kvalita potenciálnych riešení je potom určovaná na základe interakcie viacerých komponentov, ktoré spolu tvoria tím. Použitím aspoň jedného komponentu z každej populácie je získané kandidátne riešenie.

Jedinci

Jedinci v tradičných evolučných systémoch reprezentujú potenciálne riešenie nejakého hľadaného problému. Evolučný algoritmus manipuluje s jedincami pomocou variačných operátorov ako mutácia alebo kríženie, pričom na výber lepších jedincov sa používajú selekčné algoritmy.

V prípade jedincov v koevolučných algoritmoch, platia rovnaké princípy ako v evolučných algoritmoch. V CoEA jedinci môžu reprezentovať len časti riešenia (*komponenty*) alebo môžu reprezentovať testy, ktoré pomáhajú zvyšovať kvalitu riešenia.

Populácia

Jeden z rozdielov medzi evolučnými algoritmami a koevolučnými algoritmami je v populácii. EA majú len jednu populáciu jedincov s výnimkou ostrovných modelov, pričom v mnohých CoEA sa používa viacero populácií. Ak je potenciálne riešenie v CoEA rozložené na komponenty, tak aj tieto podmnožiny sa nazývajú populácia.

Návrh populácie koevolučných algoritmov, ktorý má byť efektívny, by mal zahŕňať rozhodnutie o tom, ako aspekty problému mapovať na populácie. V úlohách založených na teste sa nachádzajú dve populácie. Jedna bude populácia testov a druhá populácia potenciálnych riešení. V prípade problému, ktorý je typický pre riešenie tímom s konkrétnym počtom rôznych komponentov, sú jedinci rozdelení do populácií pre každý z komponentov.

Populácia podľa [6] je:

- Množina (multimnožina) jedincov.
- Podmnožina jedincov izolovaných od ostatných jedincov nejakou prekážkou (neschopnosť sa úspešne krížiť, geografická prekážka a podobne).

Archívy

Koevolučné metódy často ponúkajú ďalší druh kolekcie, ktorý sa nazýva archív. Archívy môžu slúžiť ako pamäť prehľadávania, keďže obsahujú informácie o generáciách, ktoré prebehli. Riešenia, ktoré sú archivované nemusia byť práve všetky evaluované riešenia. Môžu to byť napríklad len najlepšie ohodnotené riešenia z generácie.

Riešenia

Potenciálne riešenie môže byť reprezentované jediným jedincom alebo môže byť zložené z množiny jedincov.

Pri reprezentácii jediným jedincom, sa môže jedinec nachádzať v *akejkoľvek populácii*, pričom musí byť vo všetkých použitých populáciách rovnaký typ jedincov. Ďalej sa môže nachádzať v *špecifickej populácii*. To je napríklad doména s dvoma rôznymi rolami, ktoré sú reprezentované dvoma populáciami. Okrem toho, sa môže jedinec nachádzať aj v *archíve*.

Ak je potenciálne riešenie zložené z *množiny komponentov*, tak môžu byť celým obsahom alebo podmnožinou archívu, respektíve populácie. Množina komponentov môže byť taktiež kolekcia jedincov, jeden z každého archívu, respektíve populácie.

Okrem toho, sa môžu vyskytnúť populácie alebo archívy v CoEA, ktorých jedinci sa nezúčastňujú na reprezentácii potenciálneho riešenia. Napríklad v úlohách založených na teste sa nachádzajú aj populácie, respektíve archívy, ktoré nie sú vôbec použité počas vyhodnotenia.

Interakcia

K interakcii jedincov sa využívajú dva hlavné prístupy pre výber interakcií zo všetkých možných interakcií. Sú to prístupy *zamerané na jedinca* a *prístupy zamerané na populáciu.*

- **Interakcia zameraná na jedinca** - v danom čase je pre daného jedinca vybraná množina interakcií a po ich vyhodnotení je danému jedincovi priradená fitness hodnota. Táto množina interakcií sa môže znova použiť na ohodnotenie iného jedinca.
- **Interakcia zameraná na populáciu** - v danej iterácii CoEA, používa pre výpočet fitness hodnoty všetkých jedincov v populácii množinu interakcií.

Veľkosť vzorky je počet interakcií na jedinca, potrebných pre vyhodnotenie fitness hodnoty. Môže byť rovnaká pre celú populáciu alebo sa môže líšiť pre ohodnotenie každého jedinca. Na základe toho sa môže líšiť aj počet interakcií pri ohodnocovaní jedinca.

Agregácia

Agregácia sa používa v prípade, ak je jedinec ohodnotený na základe viacerých interakcií, je potrebné určiť, akým spôsobom budú agregované výsledky.

Jednou z možností je, že sa hodnoty zlúčia do jednej, pomocou matematických funkcií a výsledná hodnota je používaná ako bežná hodnota fitness. V tomto prípade nie je vždy jasné, ktorá metóda slúžiaca k zlúčeniu hodnôt poskytne výber najkvalitnejšieho jedinca, keďže rôzne metódy môžu poskytnúť rôzne ohodnotenia.

Ďalšou možnosťou je uchovávanie fitness hodnoty ako n-ticu pre každého jedinca a následne je výber rodičov riadený pomocou metód multi-kriteriálnych EA.

Komunikácia

Ak sú používané CoEA s viacerými populáciami tak, aby jedinci z jednej populácie interagovali s jedincami z inej populácie. Je potrebné, aby informácie o obsahu boli poskytované medzi populáciami. Toto je riešené pomocou použitia princípov z distribuovaných systémov:

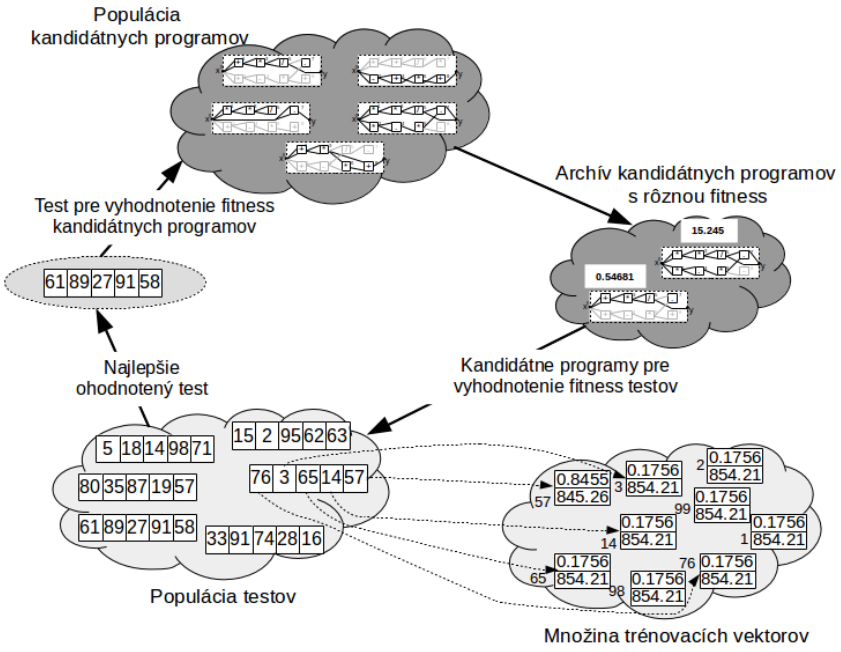
- **Koordinácia** - v prípade koordinácie môže byť komunikácia synchrónna alebo asynchrónna. Pri synchrónnej určujú výmenu informácii medzi populáciami centrálne hodiny. Pri asynchrónnej sa každá populácia vyvíja vlastnou rýchlosťou a komunikuje s ostatnými populáciami pomocou zdieľanej pamäti.
- **Tok** - ak sa jedná o synchrónny model, môže byť paralelný alebo sériový. Ak je paralelný, populácie sa určitú dobu súbežne vyvíjajú, potom sa na určitú dobu vývoj pozastaví a nastáva výmena informácií medzi ostatnými populáciami, aby sa mohlo pokračovať vo vývoji. V prípade asynchrónneho modelu sa vždy jedná o paralelný prípad, v každom časovom okamihu pracuje viac ako jedna populácia.
- **Frekvencia** - môže byť jednotná pre všetky populácie alebo sa môže líšiť od populácie k populácii. Udáva počet iterácií evolučného cyklu, ktoré prebehnú medzi dvoma komunikačnými udalosťami.

Koevolúcia v CGP

Cieľom koevolučného CGP je povoliť kandidátnym programom a testom automaticky sa navzájom vylepšovať dovtedy, kým nie je nájdené uspokojivé riešenie problému. Model tejto metódy je zobrazený na obrázku 2.10, kde je možné vidieť, že model využíva dve súbežne sa vyvíjajúce populácie. Jedná sa o populáciu kandidátnych programov, ktoré používajú k vývoju CGP a o populáciu testov, ktoré k vývoju používajú jednoduchý genetický algoritmus. K evolúcii sú využívané dve vlákna. Koevolúcia je implementovaná nasledovne.

Prvé vlákno má na starosti evolúciu kandidátnych programov, kde na začiatku je náhodne vygenerovaný archív kandidátnych programov s rôznou fitness a populácia kandidátnych programov. Zo zdieľanej pamäti je načítaný test, pomocou ktorého sa určí fitness hodnota kandidátnych programov. Na základe najlepšie ohodnoteného kandidátneho programu je vybraný rodič. Ak hodnota fitness súčasného rodiča nespadá do intervalu predošlej hodnoty fitness rodiča, rodič je uložený do archívu kandidátnych programov. Následne je vytvorená nová generácia kandidátnych programov s použitím rodiča.

V druhom vlákne, ktoré sa stará o populáciu testov, je na začiatku náhodne vygenerovaná populácia testov. Jedinci z populácie testov sú ohodnotení fitness hodnotou, k čomu sú použité kandidátne programy z archívu. Najlepšie ohodnotený test je vybraný a uložený do zdieľanej pamäte. Následne je vytvorená nová generácia testov s použitím jednobodového kríženia a turnajovej selekcie. Druhé vlákno čaká na signál od prvého vlákna, ktoré ho vysiela periodicky a na základe tohto signálu je buď vykonaná ďalšia iterácia, alebo je nájdené riešenie a GA je ukončený.



Obrázok 2.10: Interakcia populácií.

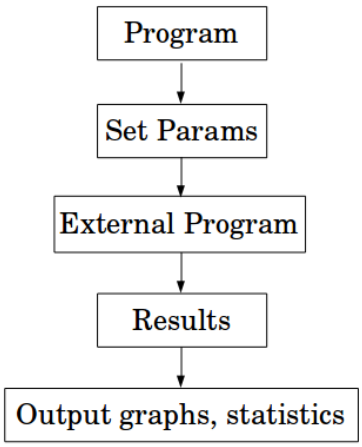
Kapitola 3

Návrh nástroja pre analýzu koevolučného algoritmu

V tejto kapitole sa nachádza bližší popis návrhu nástroja pre analýzu koevolučného algoritmu. Sú tu spomenuté požiadavky, ovládanie programu pre koevolučný návrh a tiež spracovanie výstupu.

Medzi základné požiadavky, ktoré má tento program spĺňať, patrí aj možnosť nastavenia parametrov. Jedná sa o všeobecné nastavenie parametrov programu, nastavenie typu algoritmu a nastavenie vlastností daných algoritmov.

Samozrejmosťou tohto nástroja je spolupráca s externým programom[1], ktorý umožňuje evolučný návrh obrazových filtrov s využitím koevolúcie prediktorov fitness. Pre účely experimentov bol vybraný program, ktorý vykonáva evolučný návrh pomocou *štandardného CGP*, využívajúci koevolúciu prediktorov fitness s *fixnou veľkosťou prediktora* a s *adaptívnou veľkosťou prediktora*, kde adaptácia prebieha na princípe súbežného učenia s koevolúciou. Externý program bol prevzatý z diplomovej práce Michala Wiglasza [9]. Z neho sú získavané údaje, ktoré budú použité pre vykresľovanie grafov a štatistík.



Obrázok 3.1: Základná schéma návrhu.

[1]Externý program sa nachádza na `githube` Michala Wiglasza: https://github.com/kacer/coco.

- Graf by mal byť schopný zobrazovať predikovanú fitness, reálnu fitness a zmenu prediktora v reálnom čase, vzhľadom k zmene v danej generácii.
- V prípade viacerých behov je potrebné štatistické vyhodnotenie pomocou BoxPlot diagramov.

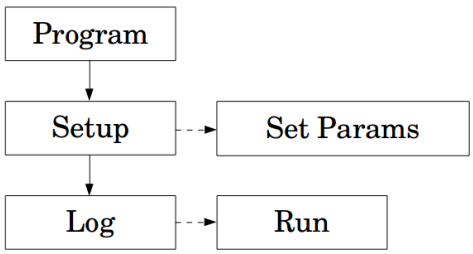
Jednou z funkcií by malo byť zaznamenávanie, zobrazovanie a následné prehrávanie priebehu aktuálne používaných trénovacích vektorov pri zmene prediktora. Po skončení výpočtov by mala byť možnosť zobrazenia najpoužívanejších trénovacích vektorov, ktoré budú odlíšené intenzitou farby. Najviac používané trénovacie vektory budú mať najvyššiu intenzitu. Základnú schému návrhu je možné vidieť na obrázku 3.1.

3.1 Ovládanie programu pre koevolučný návrh

Všetky vlastnosti, nastavenia a parametre bude možné nastavovať v hlavnej ponuke. Konkrétne prvky budú rozdelené do viacerých záložiek, medzi ktorými bude možnosť jednoduchého prepínania. Činnosť programu bude zahájená až stlačením tlačidla *Run*, ktoré by sa malo nachádzať v záložke *Log*. Okrem tejto záložky by mali byť dostupné aj ďalšie - *Setup*, *Statistics* a *Predictors*.

Nastavenie parametrov:

Pre úspešné spustenie programu je potrebné nastaviť parametre evolúcie / koevolúcie. Toto nastavenie umožňuje špecifikovať vlastnosti behu. Pri koevolúcii s pevnou veľkosťou prediktora a koevolúcii s adaptívnou veľkosťou prediktora sa jedná napríklad o nastavenie veľkosti prediktora. Toto nastavenie bude možné nájsť v hornej lište ako záložku s názvom *Setup* v hlavnej ponuke. Okrem iného, je potrebné zadať cestu k trénovacím dátam. Schéma nastavenia parametrov v nástroji je na obr. 3.2.



Obrázok 3.2: Schéma pre nastavenie parametrov.

Nastavenie experimentov:

Nástroj by mal taktiež podporovať viacnásobné spustenie, ktoré je potrebné na zobrazenie objektívnych výsledkov, vytvorenie štatistík a minimalizovanie rizika vychýlenia hodnôt od normálu.

Pri tomto nastavení bude program spustený n-krát s rovnakými nastaveniami. Hlavným dôvodom použitia viacnásobného spustenia bude možnosť zobrazenia grafov a štatistík zo získaných dát.

S viacnásobným spustením je úzko spojené paralelné spracovanie procesov, ktoré by malo byť tiež podporované.

- **Sekvenčné spracovanie** - jednotlivé behy programu sú spustené po jednom za sebou. To spôsobuje neefektívne využitie stroja a tiež vyššiu časovú náročnosť na beh programu.
- **Paralelné spracovanie** - umožňuje podľa zadaného počtu spustiť n procesov súčasne. Nielenže sa tým dosiahne rýchlejší beh programu, ale aj efektívnejšie využitie pamäte stroja.

3.2 Spracovanie výstupov

Jednou z najdôležitejších súčasti nástroja pre analýzu koevolučného algoritmu je získanie a spracovanie výstupu. Výstup bude dôležité spracovávať s dôrazom na to či sa jedná o jeden alebo viacero behov, kedže v oboch prípadoch je potrebné sledovať niečo iné.

Ak sa jedná o jeden beh, pozornosť je sústredená na vývoj grafu v reálnom čase. V prípade viacerých behov sú získavané dáta zo všetkých behov a výstup je zobrazený v podobe štatistického grafu BoxPlot.

Jednotlivé dáta by mali byť získavané z externého programu, ktorý vykonáva výpočty a mal by vrátiť výsledné dáta. Tie budú následne prebrané a spracovávané do podoby vhodnej na použitie v implementovanom nástroji.

Jeden beh:

V prípade ak bude zadaný jeden beh, budeme sledovať vývoj reálnej fitness, predikovanej fitness a výber prediktorov počas vopred zadaného počtu generácii.

Čo sa týka fitness funkcie v kartézskom genetickom programovaní, výpočet prebieha podobne ako v genetickom programovaní. Fitness hodnota je priradená kandidatnému riešeniu na základe rozdielu medzi očakávaným výstupom a výstupom kandidatného programu, pričom kandidátne riešenie bolo spustené nad množinou trénovacích vektorov.

V koevolučných algoritmoch sa nepoužíva priame porovnávanie fitness hodnoty jedincov, to je hlavným rozdielom voči evolučným algoritmom. Dvaja jedinci sú porovnávaní na základe výsledku ich interakcie s inými jedincami. Vďaka tomu môžu byť jedinci rôzne ohodnocovaní v priebehu evolúcie. Fitness hodnota jedincov sa môže v priebehu evolúcie meniť.

Dôležitou úlohou bude spracovávať výstupy v reálnom čase a zobrazovať ich v podobe grafu, aby bolo možné sledovať pre nás zaujímavé dáta, ale aj celý priebeh vybraného algoritmu. Jednotlivé skupiny dát budú od seba farebne odlíšené tak, aby bola zabezpečená lepšia orientácia v grafe. Sledovať priebeh je možné v rôznych algoritmoch či už je to CGP, koevolúcia s pevnou veľkosťou prediktora alebo koevolúcia s adaptívnou veľkosťou prediktora.

V prípade, ak bude vybraných viacero typov algoritmov alebo ak bude nastavených viacero behov, bude graf generovaný pre každý beh s možnosťou prepínania medzi nimi. Nemala by chýbať možnosť uloženia výsledných grafov.

Opakované experimenty - viac behov:

Evolučné algoritmy sa zaraďujú medzi stochastické, preto nie je vhodné vykonávať len jeden beh. Ak chceme vyhodnotiť či je metóda úspešná, alebo ju porovnať v danej úlohe s inou, je nutné vykonať rad behov a z nich vytvoriť štatistiky. Je možné porovnávať rôzne evolučné algoritmy medzi sebou a sledovať ich odlišnosti. Taktiež je možné porovnávať jeden evolučný algoritmus s radou rôznych nastavení, z čoho nám automaticky vznikajú rôzne štatistiky. Informácie v tejto podkapitole sú čerpané z [2].

Pre testovanie štatistických hypotéz je potrebné určiť či predpokladu, ktorý bol daný pred uskutočnením testu, vyhovujú experimentálne získané dáta.

- **Štatistická hypotéza** je predpoklad o rozdelení náhodných veličín.
- **Testovanie hypotéz** znamená určiť pravdepodobnosť nameraných dát v prípade, že hypotéza je platná.

Samotný postup testovania hypotéz sa označuje ako **štatistický test**. Pri testovaní štatistických hypotéz porovnávame vždy dve hypotézy. Jedna je **nulová hypotéza**, ktorou testujeme a značíme ju ako H_0. Druhú, **alternatívnu hypotézu**, značíme ako H_1.

Posudzovanie nulovej hypotézy je založené na:

- Predpokladáme, že nulová hypotéza H_0 platí.
- Rozhodneme, ktorý náhodný pokus použijeme a ktorá náhodná veličina bude výsledkom pokusu.
- Stanovíme hladinu spoľahlivosti α - miera rizika, že H_0 nesprávne zamietneme.
- Určíme kritický obor zvolenej náhodnej veličiny, do ktorého, za platnosti nulovej hypotézy, patrí výsledok veličiny s pravdepodobnosťou α.
- Ak hodnota náhodnej veličiny patrí do kritického oboru, zamietneme H_0 a prijmeme H_1.

K otestovaniu nulovej hypotézy H_0 oproti alternatívnej hypotéze H_1 sa využíva **testovacie kritérium**, čo je štatistika T. Jedná sa o funkciu náhodného výberu, ktorej rozdelenie poznáme pri platnosti nulovej hypotézy.

Obor hodnôt testovacieho kritéria je rozdelený na dva disjunktné obory. Obor prijatia testovanej hypotézy V a kritický obor W. Tieto dva obory od seba oddeľujú takzvané **kritické hodnoty**, čo sú kvantily rozdelenia testovacieho kritéria pri platnosti hypotézy H_0.

Pravdepodobnosť, že testovacie kritérium T za platnosti nulovej hypotézy odpovedá získanej alebo ešte extrémnejšej hodnote, sa nazýva **p-hodnota**.

Chyby testu sa delia na:

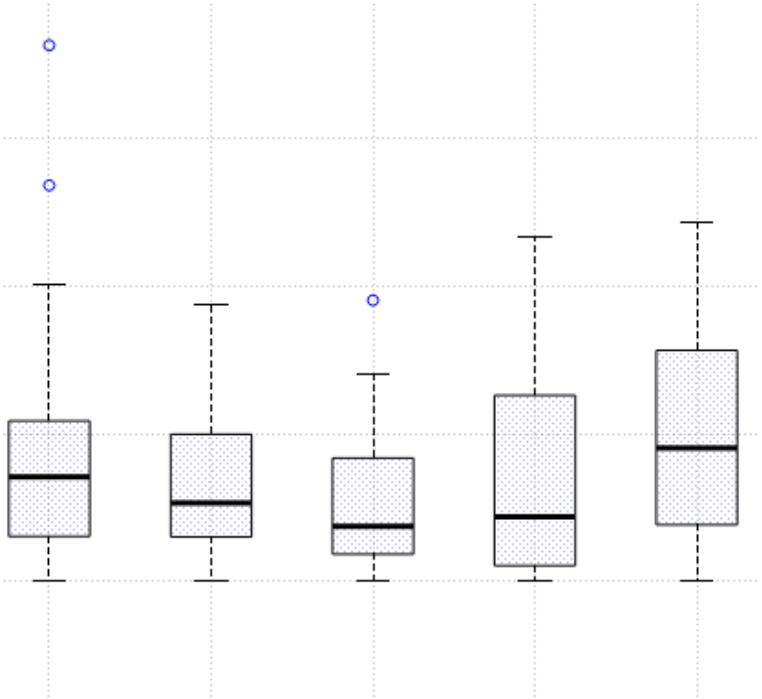
- Chyby $I.$ druhu - hladina významnosti testu ($\alpha = P(T \in W|H_0)$).
- Chyby $II.$ druhu - sila testu ($\beta = P(T \in V|H_1)$).

Znižovanie α vedie k rastu β a naopak.

Štatistické vyhodnotenie - BoxPlot diagramy:

V tomto nástroji bude možné z viacerých behov vytvárať štatistiky a zobrazovať ich v podobe BoxPlot diagramov. Jedná sa o jeden zo spôsobov vizualizácie numerických dát pomocou ich kvartilov.

Ako je možné vidieť na obrázku 3.3, stredná časť diagramu je zospodu ohraničená 1. kvartilom, zhora 3. kvartilom a medzi nimi sa nachádza čiara, ktorá vymedzuje medián. Prerušovaná čiara smerom hore či smerom dole, zakončená horizontálnou čiarou, môže znázorňovať minimum a maximum všetkých dát alebo najnižší údaj spodného kvartilu a najvyšší údaj horného kvartilu. Všetky dáta, ktoré sa nezmestia do tohto intervalu sú zaznamenané ako odchýlky.



Obrázok 3.3: Príklad BoxPlot diagramu.

Kapitola 4

Implementácia

V tejto kapitole sa nachádza podrobnejšie popísaná praktická časť tejto práce. Popisuje a zdôvodňuje výber nástrojov na implementáciu, použité knižnice, zvažované varianty, riešené problémy a taktiež nadväznosť na už implementovaný program, ktorý umožňuje prevádzať evolučný návrh obrazových filtrov s využitím koevolúcie prediktorov fitness.

Program pre analýzu koevolučného algoritmu bol napísaný v jazyku C++, využívajúc nástroj na implementáciu grafického rozhrania, `Qt Creator` verzie 3.0.1. Táto verzia je založená na `Qt` 5.2.1 a používa prekladač `GCC` 4.6.1. Operačný systém, na ktorom bola vyvíjaná táto aplikácia, bol `Linux Mint 16`.

4.1 Zvolené nástroje

Pred začiatkom implementácie programu bolo potrebné zvoliť, aký programovací jazyk bude využívaný. Rozhodovanie prebiehalo medzi `C`, `C++` alebo skriptovacím jazykom `Python`. Po dôkladnejšom preskúmaní problematiky, ktorú táto práca rieši, bol zvolený ako implementačný jazyk `C++`, ktorý využíva aj nástroj `Qt Creator`. `Qt Creator` ponúka širokú škálu možností, v tomto prípade aj implementáciu grafov a štatistík, čo bolo pre túto prácu smerodajné.

Tvorba grafov a štatistík

Dôležitou úlohou pri implementácii bolo vhodné zvolenie nástroja, respektíve spôsobu pre tvorbu a vykresľovanie grafov a štatistík.

Zvažované spôsoby tvorby grafov:

- **Vykresľovanie priamo v programe s použitím prídavnej knižnice** - prichádzajúce dáta sú predané triede využívajúcej prídavnú knižnicu a priamo vykresľované na výstup.
- **Vykresľovanie priamo v programe bez použitia prídavnej knižnice** - prichádzajúce dáta pre konkrétne body sú použité pre tvorbou spojitých čiar grafu. Táto varianta by zďaleka nedosahovala také vizuálne výsledky ako vyššie spomínaná varianta.
- **Volanie externého programu, respektíve skriptu, ktorý by podporoval vykresľovanie grafov** - tento spôsob by bol vhodný, no nebol zvolený z dôvodu vyššej

časovej a výkonnostnej náročnosti. Medzi takéto nástroje patrí napríklad `gnuplot`.

Po dôkladnejšom preskúmaní vyššie spomínaných možností, bol zvolený pre implementáciu nástroja v tejto práci prvý spôsob - vykresľovanie priamo v programe s použitím prídavnej knižnice. V tomto prípade je najdôležitejší prvok pri tvorbe grafov a štatistík `C++` widget pre `Qt Creator` s názvom `QCustomPlot` [1]. Táto knižnica je zameraná na vytváranie grafov a štatistík pre vizualizáciu dát. Pred jej použitím bolo potrebné sa podrobnejšie oboznámiť s dokumentáciou, ktorá popisuje nastavenia, tvorbu základných grafov a podobne.

Tvorba štatistík - BoxPlot diagramy

Po vykonaní každého behu je dostupná výsledná hodnota fitness. Po skončení všetkých behov, sú z výsledných hodnôt vypočítané údaje potrebné pre BoxPlot diagram. Ako už bolo spomínané v kapitole 4, je to 1. kvartil, 3. kvartil, medián a odchýlky. Výpočet a následné vykreslenie štatistických grafov prebieha v triede `boxplot.cpp`. V implementovanom nástroji sa nachádzajú BoxPlot štatistiky v hlavnom menu v záložke `Statistics`.

Vykresľovanie v reálnom čase

V návrhu bolo požadované, aby niektoré grafy prebiehali a boli vykresľované v reálnom čase. To je riešené v triede `realtimeprocess.cpp`, ktorá využíva knižnicu `<QProcess>`. Medzi požiadavky pre vykresľovanie v reálnom čase patrí ukončenie predchádzajúceho procesu a výstup z externého programu, ktorý musí obsahovať potrebné dáta pre aktuálny graf. Tieto podmienky musia byť splnené, pre úspešné vykreslenie následujúceho bodu grafu v reálnom čase. Konkrétne vykresľovanie prebieha volaním triedy na vykresľovanie grafov vždy, keď je spustený nový proces.

Paralelné spracovanie procesov

So spracovaním a vykresľovaním dát v reálnom čase je úzko spojená aj paralelizácia procesov. Potrebná je predovšetkým preto, aby dochádzalo k efektívnejšiemu využívaniu pamäte. Ďalším z dôvodov použitia paralelizácie je aj nižšia časová náročnosť na získanie výsledku programu. Dôležité je dbať aj na vhodné zvolenie súčasne spracovávaných procesov, aby nenastalo zahltenie pamäte.

Pred implementáciou boli zvažované dva spôsoby paralelizácie. Jedným z nich je využitie vlákien, v ktorých by bežali procesy nezávisle od seba. Druhým spôsobom je pri spustení vytvoriť a spustiť toľko paralelných procesov, koľko ich bolo zadaných na vstupe a vždy po skončení niektorého z procesov bude vytvorený nový proces. Tým je vždy zabezpečené, že bude bežať maximálne zadaný počet paralelných procesov. Z dôvodu menšej náročnosti a vyššej efektivity bol zvolený druhý spôsob.

4.2 Nadväznosť na už vytvorený program

Neoddeliteľnou súčasťou implementácie tohto nástroja je nadväznosť na už implementovaný program, umožňujúci prevádzať evolučný návrh obrazových filtrov s využitím koevolúcie prediktorov fitness [9]. Tento program je volaný vždy pri spustení nového behu

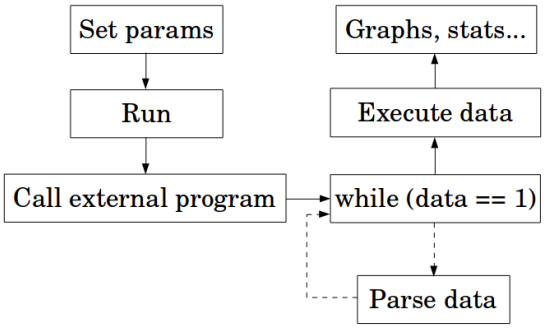
[1] Jedná sa o knižnicu pod licenciou GPL. Je voľne dostupná na: http://www.qcustomplot.com/.

implementovaného nástroja pre analýzu koevolučného algoritmu, aby boli dostupné nové dáta. Schéma jedného behu je zobrazená na obrázku 4.1.

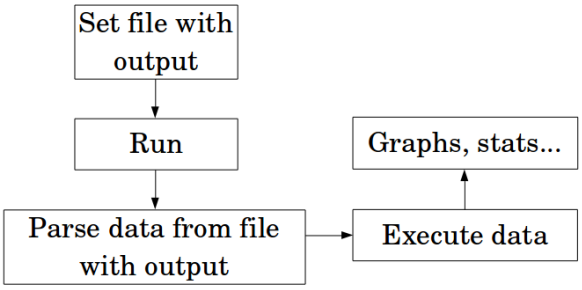
Štandardný výstup a štandardný chybový výstup sú presmerované do nástroja, kde prebieha parsovanie potrebných hodnôt pre grafy, štatistiky a podobne. Parsovanie prebieha v triede `realtimeprocess` a výsledky sú ukladané do vektorov, ktoré sú predávané ďalej na spracovanie.

Súbor s výstupom vo vhodnom formáte

Nástroj je možné spustiť aj bez využitia externého programu vtedy, ak pri nastavení parametrov zadáme už vytvorený súbor s výstupom vo vhodnom formáte. Ostatné parametre sa nenastavujú. Tento výstupný súbor je spracovaný v triede `runfromfile`. Dáta sú z výstupného súboru vyparsované a použité pri tvorbe grafov a štatistík. Schéma jedného behu sa nachádza na obrázku 4.2. V tomto prípade nie je možné spracovávať trénovacie vektory pri zmene prediktorov, keďže nemáme dostupné výstupné hodnoty na spracovávanie.



Obrázok 4.1: Schéma pre nástroj, ktorý využíva externý program.



Obrázok 4.2: Schéma pre nástroj, ktorý využíva súbor s výstupom.

4.3 Riešené problémy

Počas implementácie bolo potrebné vyriešiť mnoho problémov. Niektoré boli jednoduchšie, iné zase zložitejšie. Jedným z problémov bola potreba zabezpečiť prepínanie medzi jednot-

livými grafmi, ktoré boli vykresľované v reálnom čase. Problém nenastal, ak v programe nebežali paralelné procesy. Avšak, ak bol program spustený s paralelnými procesmi, pri prepínaní medzi súčasne bežiacimi grafmi neboli vykresľované už spočítané hodnoty. To bolo vyriešené takým spôsobom, že všetky dáta do daného priebehu boli ukladané do jednotlivých vektorov, ktorých počet je rovný počtu paralelných procesov. Pri prepínaní medzi grafmi sa na začiatku načítajú a vykreslia všetky hodnoty, ktoré boli do tohto okamihu spracované. Priebeh potom pokračuje ďalej od posledného bodu.

Ďalší problém nastal pri spracovaní najčastejšie používaných trénovacích vektorov pre dané nastavenie. Problém bol v tom, že bolo potrebné spracovávať veľké množstvo dát. Okrem spracovania bolo potrebné k týmto dátam pristupovať a určiť množstvo výskytov jednotlivých trénovacích vektorov. To spôsobovalo výrazné spomalenie programu. Riešením bolo vykresľovanie hneď pri spracovaní. Tu však nastal ďalší problém. Nebolo možné rozlíšiť stupeň intenzity farby. Finálnym riešením je preto nastavenie priehľadnosti farby trénovacích vektorov a ich vykresľovanie na seba, čo spôsobuje odlíšenie najviac a najmenej používaných trénovacích vektorov na základe intenzity farby. Tým sa výrazne znížilo už spomínané spomalenie programu.

Problém nastal aj pri načítaní výstupu zo súboru, kde sa nástroj nevie vysporiadať s rôznym rozložením jednotlivých dát. Najlepším riešením sa zdá byť zvolenie jednotného formátu súboru s výstupom. To znamená, že výstupný súbor musí obsahovať rovnaký počet behov pre jednotlivé typy algoritmov a jednotlivé algoritmy musia byť usporiadané za sebou v nasledujúcom poradí: štandardné CGP, koevolúcia s fixnou veľkosťou prediktora a koevolúcia s adaptívnou veľkosťou prediktora. Nepočíta sa tiež s paralelným spracovaním pri načítaní dát zo súboru s výstupom.

Mnoho menších problémov nastalo aj s tvorením paralelizácie, ktoré boli po niekoľkých úvahách nad daným problémom vyriešené.

Kapitola 5

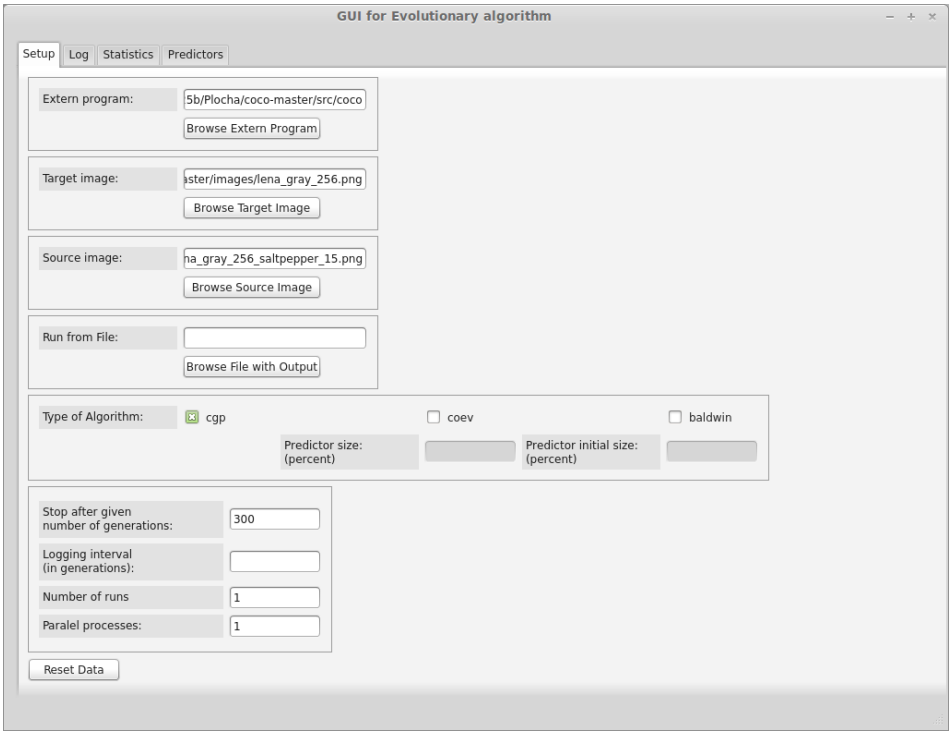
Činnosť nástroja

Táto kapitola bližšie popisuje činnosť nástroja v úlohe koevolučného návrhu obrazových filtrov. Okrem toho sa tu nachádza popis jednotlivých činností, pomocou vhodných príkladov z daných častí implementovaného nástroja.

5.1 Popis jednotlivých záložiek programu

Po spustení programu sa vychádza z hlavnej ponuky, ktorá obsahuje štyri záložky, medzi ktorými sa dá jednoducho prepínať.

Ako predvolená záložka po spustení je nastavená `Setup` viď. obrázok 5.1, kde prebieha nastavenie rôznych parametrov. Potrebné je nastaviť externý program, ktorého dáta budú

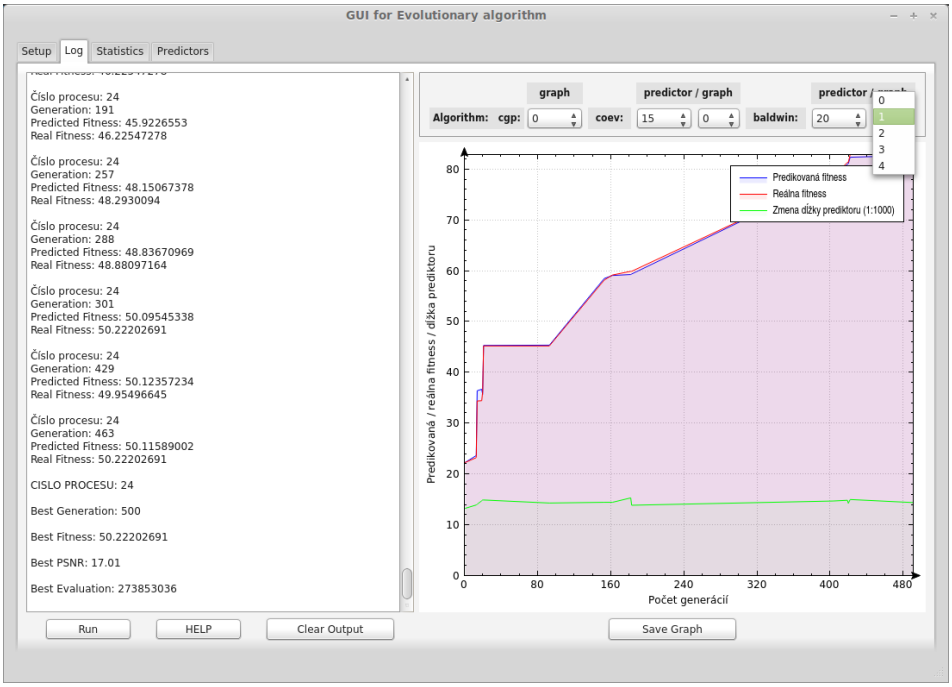


Obrázok 5.1: Ukážka z implementovaného nástroja - záložka Setup.

spracovávané. Tiež je nevyhnutné zadať cieľový a zdrojový obrázok a zaškrtnúť algoritmy, ktoré chceme používať. V prípade koevolúcie s fixnou veľkosťou prediktora a koevolúcie s adaptívnou veľkosťou prediktora je možné nastaviť aj veľkosť prediktora v percentách. Ako je možné vidieť na obrázku 5.1, v dolnej časti nástroja sú nastavenia, ktoré sa tykajú behu programu. Je to nastavenie počtu generácií, nastavenie intervalu vypisovania logov a nesmie chýbať nastavenie počtu behov programu a paralelných procesov. V prípade, že nie je nastavený interval vypisovania logov, logy sú vypisované len ak nastane zmena. Ak je zadaná cesta k súboru, v ktorom je zaznamenaný výstup v danom formáte, ostatné parametre sa nenastavujú. V takomto prípade, nie je možné pri zmenách prediktorov spracovávať obrázky s vyznačenými trénovacími vektormi.

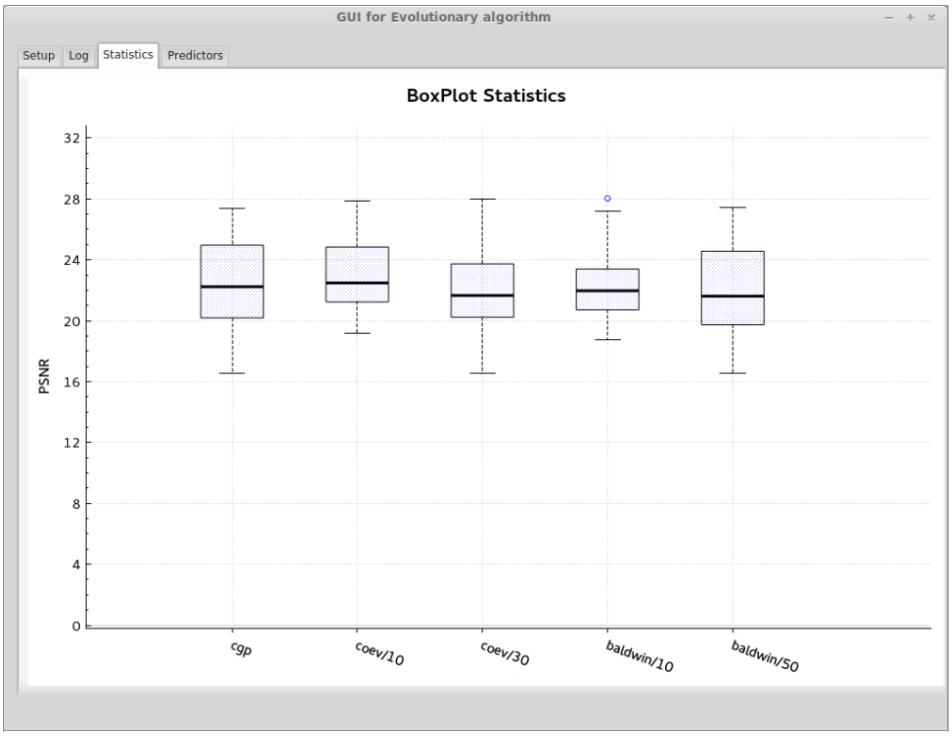
V záložkách `Log`, `Statistics` a `Predictors` už neprebieha žiadne nastavovanie. Sú to záložky, kde nájdeme grafy, štatistiky a vizualizáciu používaných trénovacích vektorov v prípadoch, keď nastanú zmeny prediktorov.

Na obrázku 5.2 vidíme výsledok po spracovaní dát. Je možné prepínať medzi jednotlivými grafmi a sledovať priebeh a zmeny dát. Vybraný graf môže byť uložený. V ľavej časti nástroja sa nachádzajú informácie o priebehu v textovej podobe. Dôležitým tlačidlom v tejto záložke je `Run`, ktorým sa spúšťa celý proces spracovávania dát. Nachádzajú sa tu tiež tlačidlá pre výpis nápovedy a vyčistenie plochy s informáciami o priebehu.



Obrázok 5.2: Ukážka z implementovaného nástroja - záložka Log, zobrazuje priebeh fitness s možnosťou prepínania medzi jednotlivými behmi.

Záložka `Statistics` ponúka výslednú štatistiku - BoxPlot diagram. Počas prebiehajúcich výpočtov výsledný diagram nie je k dispozícii, objaví sa až po ukončení všetkých behov. Na obrázku 5.3, je možné vidieť, že výsledný diagram obsahuje porovnanie na základe hodnoty PSNR (peak signal-to-noise ratio). Tento diagram bol vygenerovaný pri 5000 generáciách a pri 50 behoch pre každé nastavenie.



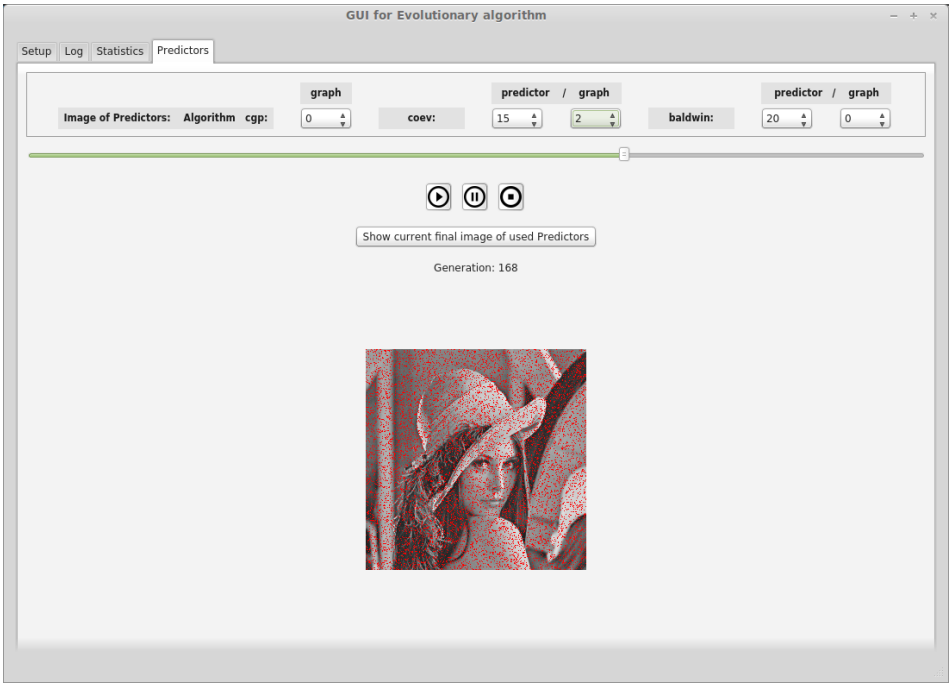
Obrázok 5.3: Ukážka z implementovaného nástroja - záložka Statistics, umožňuje porovnávanie testovaných algoritmov alebo nastavení.

Sledovať priebeh aktuálne používaných trénovacích vektorov pri jednotlivých zmenách prediktora v reálnom čase je možné na poslednej záložke `Predictors`. Pokiaľ bežia výpočty, zobrazuje sa vždy obrázok, ktorý prislúcha aktuálne prebiehajúcemu výpočtu. Ako je možné vidieť na obrázku 5.4, po dokončení všetkých behov, podobne ako v záložke `Log`, sa dá prepínať medzi jednotlivými algoritmami, nastaveniami a príslušnými grafmi. Pre každý graf je dostupná sada obrázkov s vyznačenými trénovacími vektormi a informáciou, v ktorej generácii došlo k zmene prediktora. Medzi jednotlivými obrázkami je možné ručne prepínať pomocou posuvníka alebo ovládať priebeh pomocou tlačidiel `spusti`, `pozastav` a `zastav` priebeh.

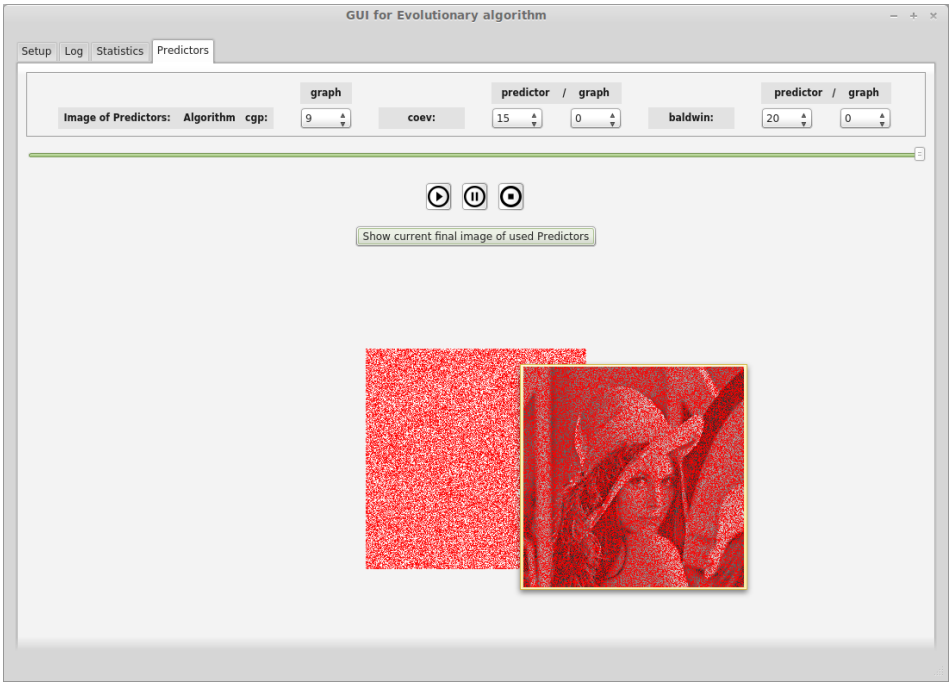
Pre každý výber je tiež možné zobraziť finálny obrázok, kde sú zakreslené všetky používané trénovacie vektory počas celého daného behu. Na obrázku 5.5, sa nachádzajú trénovacie vektory vykreslené na bielom pozadí. Ak prejdeme myšou na obrázok, ako pozadie sa zobrazí pôvodný obrázok.

5.2 Testovanie

Pre zaistenie stabilnej a funkčnej aplikácie bolo potrebné vykonať rad testov. Testovanie prebiehalo pri veľkom množstve behov a generácií, čo bolo potrebné aj pre štatistické vyhodnocovanie jednotlivých behov. Maximálny počet generácií sa pohyboval v desaťtisícoch a počet behov pre jednotlivé nastavenia v stovkách. Paralelné spracovanie bolo nastavované v závislosti od výkonu stroja.



Obrázok 5.4: Ukážka z implementovaného nástroja - záložka Predictors, pri behu zobrazuje aktuálne vyberané trénovacie vektory. Po skončení je možné priebeh prejsť.



Obrázok 5.5: Ukážka z implementovaného nástroja - záložka Predictors - výsledný obrázok početnosti výberu daného trénovacieho vektoru.

Kapitola 6

Záver

Hlavným cieľom tejto práce bolo navrhnúť a implementovať nástroj s grafickým užívateľským rozhraním, ktorý umožní analýzu koevolučného algoritmu pre rôzne nastavenia parametrov a tiež jeho štatistické vyhodnotenie.

Nástroj implementovaný v tejto práci umožňuje rôzne nastavenia parametrov pre kartézske genetické programovanie, koevolúciu s fixnou veľkosťou prediktora a koevolúcia s adaptívnou veľkosťou prediktora. Mimo iné podporuje aj nastavenie počtu generácií, behov programu a súbežne spustených procesov. Z výsledných dát sú generované grafy v reálnom čase pre jednotlivé behy. Po skončení výpočtov je k dispozícii aj štatistický graf BoxPlot, kde je možné prehľadne porovnávať výsledky jednotlivých nastavení. Nástroj bol testovaný v úlohe koevolučného návrhu obrazových filtrov a tiež podporuje vykresľovanie aktuálne používaných trénovacích vektorov na výpočet. Na konci výpočtov je možné si prezerať alebo prehrávať obrázky všetkých použitých trénovacích vektorov pre jednotlivé nastavenia a grafy. Samozrejmosťou je aj výsledný obrázok najčastejšie používaných trénovacích vektorov pre dané nastavenie. Okrem spracovania v reálnom čase je možné načítať textový súbor, obsahujúci výstup a vykresliť k nemu grafy, štatistiky.

Nástroj bol testovaný a implementovaný na základe získaných dát z implementovaného externého programu, ktorý umožňuje evolučný návrh obrazových filtrov s využitím koevolúcie prediktorov fitness. Bolo vykonaných množstvo testov s rôznymi nastaveniami pre jednotlivé algoritmy. Okrem porovnávania rovnakých algoritmov s rôznymi nastaveniami, prebiehalo aj porovnávanie medzi rôznymi algoritmami. V prípade upravenia výstupov by bolo možné tento nástroj použiť aj pre iné už implementované algoritmy.

Prínos tejto práce je v jednoduchom spustení implementovaného algoritmu pomocou grafického užívateľského rozhrania, ktoré umožňuje nastavenie parametrov, prepočítanie dát a následné vykreslenie grafov a štatistík, kde je možné jednoducho preskúmať priebeh jednotlivých behov. To výrazne sprístupňuje a uľahčuje prácu užívateľom, ktorých cieľom je získať a porovnávať vizualizované dáta z rôznych behov koevolučných algoritmov. Implementácia bola orientovaná na optimalizáciu jednotlivých častí tak, aby grafy a priebeh výberu prediktorov mohli byť zobrazované v reálnom čase aj pri súčasnom spustení viacerých behov, ktoré program umožňuje na počítači s viacerými procesormi.

Možným rozšírením nástroja, ktoré už nebolo implementované môže byť možnosť prihlásenia sa na server priamo z nástroja, na ktorom by prebiehalo spracovanie dát a vykonávanie výpočtov.

Počas práce som sa zoznámil s problematikou evolučných algoritmov, genetického programovania a koevolučných algoritmov, ktoré boli pre mňa dovtedy neznáme. Musel som tiež preskúmať rôzne možnosti a metódy štatistického vyhodnocovania a vizualizácie dát.

Prínosom bolo aj lepšie zoznámenie sa s nástrojom pre tvorbu užívateľského rozhrania `Qt Creator` a nástrojom pre tvorbu technickej správy `Kile`, ktorý využíva systém LaTeX.

Literatúra

[1] Blanchette, J.; Summerfield, M.: *C++ GUI Programming with Qt 4, Second Edition*. Prentice Hall, 2010, ISBN 978-0-13-235416-5, 752 s.

[2] Hrabáček, R.: *Statistické srovnání evolučních metod*. Materiál pro počítačové cvičení z předmětu BIN, Brno, FIT VUT v Brně, 2015.

[3] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. Vydavateľstvo STU v Bratislave, 2000, ISBN 80-227-1377-5, 223 s.

[4] Miller, J.: *Cartesian Genetic Programming*. Natural Computing Series, Springer Berlin Heidelberg, 2011, ISBN 978-3-642-17309-7, 17-34 s.

[5] Nist/Sematech: *Handbook of Statistical Methods*. 2012.
URL http://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm

[6] Popovici, E.; Bucci, A.; Wiegand, R.; aj.: *Coevolutionary Principles*. Springer Berlin Heidelberg, 2012, ISBN 978-3-540-92909-3, 987-1033 s.

[7] Sekanina, L.; a kolektív: *Evolunčí hardware: Od automatického generování patentovatelných invencí k sebemodifikujícim se strojům*. Academia, 2009, ISBN 978-80-200-1729-1, 328 s.

[8] Vanneschi, L.; Poli, R.: *Genetic Programming - Introduction, Applications, Theory and Open Issues*. Springer Berlin Heidelberg, 2012, ISBN 978-3-540-92909-3, 709-739 s.

[9] Wiglasz, M.: *Souběžné učení v koevolučních algoritmech*. Diplomová práce, Brno, FIT VUT v Brně, 2015.

[10] Zelinka, I.; Oplatková, Z.; Šeda, M.; Ošmera, P.; Včelař, F.: *Evoluční výpočetní techniky - principy a aplikace*. BEN, 2009, ISBN 978-80-7300-218-3, 536 s.

[11] Šikulová, M.; Sekanina, L.: *Coevolution in Cartesian Genetic Programming*. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, ISBN 978-3-642-29138-8, 182-193 s.

Príloha A

Obsah CD

Technická správa

Technická správa vo formáte `PDF` s názvom `TechnickaSprava` sa nachádza na priloženom CD v adresári `text`. Zdrojové kódy pre LaTeXsa nachádzajú v adresári `text/src`, ktoré je možné preložiť pomocou príkazu `make`.

Ovládanie aplikácie

V adresári `documentation` sa nachádza návod k ovládaniu aplikácie vo formáte `PDF` s názvom `ovladanie`.

Zdrojové kódy

Zdrojové kódy implementovanej aplikácie sa nachádzajú v adresári `src/gui` a zdrojové kódy externej aplikácie, ktorej dáta sú spracovávané sa nachádzajú v adresári `src/coco`. Preklad je možné vykonať pomocou príkazu `make` v koreňovom adresári, ktorý vykoná preklad programu za pomoci programu `qmake` a preklad externej aplikácie za pomoci príslušného programu `make`. Ak sa preklad uskutoční na počítači s viacerými verziami Qt frameworku, s nastavením prioritne volaného programu `qmake` staršej verzie ako je verzia `5.2.0`, je potrebné zavolať program `qmake` explicitne pomocou absolútnej cesty. To zrealizujeme pridaním absolútnej cesty do súboru `Makefile` a zakomentovaním riadku s príkazom `qmake` (príklad absolútnej cesty: `/home/a05-0925b/Qt/5.2.1/gcc_64/bin/qmake`).

Preklad programu spustíme zadaním nasledujúceho príkazu do terminálu:

```
make
```

Preložený program spustíme dvojklikom na spustiteľný súbor v adresári `bin/gui` alebo zadaním nasledujúceho príkazu do terminálu:

```
make run
```

Aplikácia

V adresári `aplication/gui` sa nachádza preložená aplikácia, pričom v adresári `bin/gui` sa nachádza jej spustiteľný súbor `gui`. Preloženú externú aplikáciu je možné nájsť v adresári `aplication/coco` a jej spustiteľný súbor `coco` v adresári `bin/coco`. V spustenej aplikácii sa následne budú využívať súbory uložené v adresári `examples`.