VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VYUŽITÍ EVOLUČNÍCH ALGORITMŮ V ÚLOZE SYMBOLICKÉ REGRESE

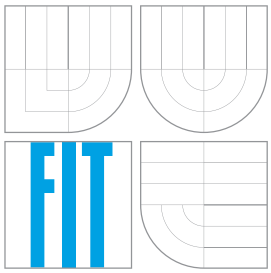
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL KOMADEL

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VYUŽITÍ EVOLUČNÍCH ALGORITMŮ V ÚLOZE SYMBOLICKÉ REGRESE

UTILIZATION OF EVOLUTIONARY ALGORITHMS IN SYMBOLIC REGRESSION PROBLEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL KOMADEL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2009

Abstrakt

Evoluční techniky jsou neustále se vyvíjející a progresivní část informatiky. Evoluční algoritmy se v praxi používají k řešení mnohých druhů problémů od optimalizace až k plánování. Tato práce se zabývá genetickým a kartézským genetickým programováním, které patří mezi nejčastěji používané algoritmy. Cílem práce je implementovat jednotlivé přístupy a vyhodnotit jejich účinnost v úloze symbolické regrese.

Abstract

Evolutionary algorithms are constantly developing and progressive part of informatics. These algorithms serve to solve many kinds of problems from optimal control to planning. This study discusses genetic and cartesian genetic programming, which belong among the most successful types of evolutionary algorithms. The goal of this work is to develop two aplications of genetic and cartesian genetic programming and evaluate efficiency of these two types of evolutionary algorithms in solving symbolic regression problems.

Klíčová slova

evoluční programování, genetické programování, kartézské genetické programování, symbolická regrese.

Keywords

evolutionary programming, genetic programming, cartesian genetic programming, symbolic regression.

Citace

Michal Komadel: Využití evolučních algoritmů v úloze symbolické regrese, bakalářská práce, Brno, FIT VUT v Brně, 2009

Využití evolučních algoritmů v úloze symbolické regrese

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Komadel
15. května 2009

Poděkování

Týmto děkuji mému vedoucímu Ing. Zdeňkovi Vašíčkovi za cenné rady a vedení počas tvorby této práce.

Obsah

1 Úvod **2**

2 Genetické programovanie **3**
2.1 Úvod do genetického programovania 3
2.2 Chromozóm 3
2.3 Populácia 5
2.4 Fitness hodnota 6
2.5 Algoritmus genetického programovania 7

3 Kartézske genetické programovanie **14**
3.1 Úvod do kartézskeho genetického programovania 14
3.2 Chromozóm 14
3.3 Algortimus kartézskeho genetického programovania 15

4 Návrh a implementácia programu **17**
4.1 Programovací jazyk 17
4.2 Aplikácia GP 17
4.3 Aplikácia CGP 26

5 Porovnanie GP a CGP **31**
5.1 Klasická symbolická regresia 32
5.2 Hľadanie iteračných predpisov 35
5.3 Vyhodnotenie experimentov 37

6 Závěr **39**

A Manual **42**

Kapitola 1

Úvod

Táto práca s zaoberá problematikou genetického a kartézskeho genetického programovania. Tieto postupy slúžia v praxi na riešenie mnohých druhov problémov od optimalizácie, plánovania, symbolickej regresie až po objavovanie matematických identít alebo evolúciu nečakaného správania[4].

Mojou úlohou bolo navrhnúť, zoptimalizovať a implementovať aplikácie genetického a kartézskeho genetického programovania, ktoré slúžia na riešenie problémov symbolickej regresie. Následne porovnať ich efektivitu a účinnosť na úlohách symbolickej regresie rôznych náročností a vyhodnotiť, ktorý spôsob je vhodnejší pre tento typ úloh.

V nasledujúcich kapitolách je postupne uvedená teória a fakty o genetickom a kartézskom genetickom programovaní, návrh a realizácia aplikácií spoločne s testami, na ktorých sa zakladali moje rozhodnutia pri ich navrhovaní. Na koniec sú uvedené testy na základe ktorých, som vyhodnotil úspešnosť prístupu genetického programovania a kartézskeho genetického programovania pri riešení úloh symbolickej regresie.

Kapitola 2

Genetické programovanie

V tejto kapitole budú uvedené základné fakty a informácie týkajúce sa genetického programovania.

2.1 Úvod do genetického programovania

Genetické programovanie (GP), predstavené koncom 80-tich rokov Johnom Kozom, vzniklo modifikáciou genetických algoritmov. Tak ako genetické algoritmy, genetické programovanie je inšpirované princípmi, ktorými sa riadi evolúcia v prírode. Špecificky Darwinovou tézou prirodzeného výberu, podľa ktorej prežívajú len tí najlepšie prispôsobení jedinci populácie[5]. V prírode sa proces evolúcie vyskytuje pokiaľ sú splnené nasledujúce pravidlá:

- jedinec má schopnosť sa reprodukovať
- existuje celá populácia takýchto jedincov
- jedinci v populácii sú v niektorých ohľadoch rôzny
- táto rôznosť má vplyv na prežitie jedinca v jeho prostredí

Po splnení týchto podmienok sa počas určitého časového obdobia a po veľkom počte generácií, pomocou sexuálnej rekombinácie (kríženia) a občasnej mutácii, v populácii vyskytnú jedinci, ktorí sú viac vhodní na vykonávanie určitej činnosti, čo im dáva väčšiu schopnosť prežiť v ich prostredí ako zvyšok populácie[4].

GP sa snaží tieto princípy aplikovať pri hľadaní riešenia daného problému. Ku genetickému programovaniu existuje niekoľko prístupov ako napríklad lineárne genetické programovanie, multivýrazové programovanie, kartézske genetické programovanie a infix forma genetického programovania[14].

2.2 Chromozóm

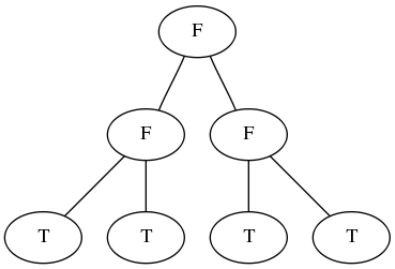
V každom systéme ktorý sa dokáže prispôsobovať svojmu prostrediu, musí aspoň jedna štruktúra prechádzať zmenami. V prípade genetického programovania je táto štruktúra hierarchicky usporiadaný počítačový program[4].

Tento program[1] sa vo všeobecnosti nazýva *chromozóm*. Biologickou terminológiou môžeme povedať, že jeden chromozóm reprezentuje genotyp[2] jedného riešenia skúmaného problému[5].

[1] prípadne funkcia – pomenovanie sa môže líšiť v závislosti na povahe riešeného problému

[2] súbor úplnej genetickej informácie týkajúcej sa skúmaného znaku alebo znakov

Táto štruktúra (chromozóm) je zostavená z množiny terminálov a z množiny funkcií. V biologickej terminológii môžme povedať, že funkcie a terminály sú gény chromozómu. Zvyčajne sa implementuje formou koreňového stromu.



Obrázek 2.1: Príklad chromozómu

Na obrázku 2.1 môžeme vidieť uzly stromu označené ako F reprezentujúce množinu funkcií. Uzly označené ako T sú terminály. Na základe povahy riešeného problému môže množina funkcií obsahovať:

- aritmetické operácie (+, -, *, ...)
- matematické funkcie (sin, cos, log, ...)
- operácie booleovskej logiky (AND, OR, NOT)
- podmienkové operátori (If-Then-Else)
- iteračné funkcie (Repeat-Until)
- rekurzívne funkcie
- iné funkcie špecifické pre povahu riešeného problému

Do množiny terminálov môžme zaradiť nasledujúce prvky:

- premenné – prvky reflektujúce prostredie (výstupy z meracích zariadení a pod.)
- matematické konštanty (celé/desatinné čísla)
- konštanty booleovskej logiky (TRUE, FALSE, NILL)

2.2.1 Funkcie a terminály

Pri výbere operácií a terminálov sa však musia vybrať také prvky aby boli splnené dve základné požiadavky, ktoré sú *uzavretosť* a *dostatočnosť*[4].

Uzavretosť

Množiny funkcií a terminálov sa považujú za uzavreté vtedy, ak akákoľvek funkcia z množiny funkcií je schopná akceptovať každú hodnotu vrátenú z ľubovolnej funkcie patriacej taktiež do množiny funkcií alebo ľubovolný prvok z množiny terminálov.

Takúto vlastnosť nemusí spĺňať napríklad operácia delenia (delenie nulou). V podobných prípadoch je nutné tieto funkcie nahradiť ich „bezpečnou verziou“. V prípade delenia môžme napríklad implemetovať funkciu, ktorá najprv skontroluje či je deliteľ rôzny od nuly a až potom vykoná samotné delenie[4].

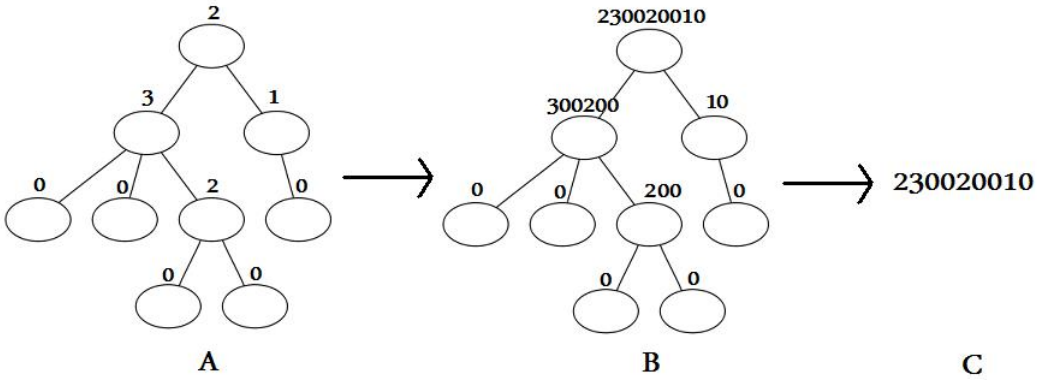
Dostatočnosť

Dostatočnosť vyžaduje aby použité funkcie a terminály boli schopné vyjadriť riešenie skúmaného problému.

Príkladom môže byť snaha o realizáciu funkcie XOR. Pokiaľ by množina terminálov pozostávala len z operácií AND a OR, nie je možné hľadaný výraz nájsť. Ak by ale bola do množiny funkcií zaradená aj operácia NOT, riešenie problému by sa už dalo vyjadriť[4].

2.2.2 Readov lineárny kód

Metóda Readovho lineárneho kódovanie koreňových stromov sa často využíva na konštruktívnu enumeráciu stromových štruktúr. Použitím tohto kódovania je možné obísť nutnosť použiť stromové štruktúri v GP a nahradiť ich lineárnymi reťazcami čísel, čo môže zjednodušiť operácie s chromozómami. Readov kód je reťazec celých čísel, ktorý určuje pre každý uzol koreňového stromu počet jeho podstromov.



Obrázek 2.2: Tvorba Readovho lineárneho kódu

Na obrázku 2.2 je znázornená tvorba Readovho lineárneho kódu. V diagrame A je každý uzol koreňového stromu ohodnotený číslom, ktoré udáva počet jeho podstromov. Diagram B ukazuje ako sa zostrojí výsledný Readov kód, tak že idúc postupne zdola nahor spájame ohodnotenie daného uzlu s kódmi nižších uzlov. Výsledný Readov kód (diagram C) je priradený koreňu stromu[5].

2.3 Populácia

V genetickom programovaní sa pojmom populácia myslí určité množstvo chromozómov, ktoré podliehajú zmenám v priebehu GP algoritmu. Toto množstvo sa nemení. To znamená,

že veľkosť počiatočnej populácie na začiatku algoritmu je rovnaká ako veľkosť populácie na konci.

Veľkosť populácie taktiež ovplyvňuje pravdepodobnosť nájdenia hľadaného riešenia. Čím väčšie je množstvo jedincov (chromozómov), z ktorých je populácia zložená, tým väčšia je pravdepodobnosť uspieť. Avšak existuje bod, za ktorým výpočtové nároky začnú prevyšovať prínos veľkej populácie[4].

2.4 Fitness hodnota

Fitness hodnota (vhodnosť) je hybnou silou genetického programovania[4]. Táto hodnota poskytuje spätnú väzbu algoritmu GP ohľadne toho, ktoré chromozómy sú vhodnejšie, poskytujúce lepšie riešenie skúmaného problému a majú tým väčšiu šancu zúčastňovať sa genetických operácií. Fitness hodnota sa získava pomocou *fitness funkcie*, ktorá túto hodnotu počíta na základe trénovacej množiny[2].

Napríklad pri hľadaní predpisu matematickej funkcie by v trénovacej množine mohli byť hodnoty x a k nim prislúchajúce hodnoty y. Fitness funkcia potom porovnáva výstupy chromozómu pre všetky x z trénovacej množiny s hodnotami y, pričom výsledná fitness hodnota by bola rovná súčtu absolútnych hodnôt odchyliek výstupov chromozómu od hodnôt y.

Z praktických dôvodov je však vhodné fitness hodnotu transformovať do normalizovaného tvaru, prípadne do intervalu (0,1)[14]. Nie vždy platí, že vyššia hodnota je aj lepšia. To záleží na type použitom pri GP. V [4] sú popísané štyri druhy, každý majúc svoje výhody a nevýhody:

- hrubá fitness hodnota
- štandardizovaná fitness hodnota
- regulovaná fitness hodnota
- normalizovaná fitness hodnota

2.4.1 Hrubá fitness hodnota

Meranie tejto hodnoty je dané problémom samotným. To znamená, že ak hľadáme predpis matematickej funkcie a fitness funkcia počíta súčet absolútnych hodnôt odchyliek od trénovacej množiny, tak nižšia fitness hodnota by bola lepšou. Platí:

$$r_i(t) = \sum_{j=1}^{N_e} |S_i(j) - C(j)| \tag{2.1}$$

kde $r_i(t)$ je fitness hodnota $i-$tého chromozómu v generácii t, N_e je počet hodnôt v trénovacej množine, $S_i(j)$ je výstup chromozómu pre $j-$tý vstup z trénovacej množiny a $C(j)$ hodnota v trénovacej množine prislúchajúca $j-$tému vstupu.

Hrubá fitness hodnota, hoci jej výpočet je zvyčajne najjednoduchší, je príčinou problému s prílišným uprednostňovaním jedincov, spomenutému v sekcii 2.5.9.

2.4.2 Štandardizovaná fitness hodnota

Je transformovaná hrubá fitness hodnota tak, že nižšia hodnota je vždy lepšia.

V prípade hľadania matematickej funkcie sú štandardizovaná a hrubá fitness rovnaké:

$$s_i(t) = r_i(t) \tag{2.2}$$

kde $s_i(t)$ je štandardizovaná fitness hodnota $i-$tého chromozómu v generácii t

V prípade ak vyššia hrubá fitness hodnota je lepšia je nutné túto hodnotu transformovať. Zvyčajne odčítaním dosiahnutej hrubej fitness hodnoty od jej maximálnej veľkosti pre daný problém.

$$s_i(t) = r_{max} - r_i(t) \tag{2.3}$$

2.4.3 Regulovaná fitness hodnota

Táto hodnota je vypočítaná zo štandardizovanej nasledujúcim spôsobom:

$$a_i(t) = \frac{1}{1 + s_i(t)} \tag{2.4}$$

Kde $a_i(t)$ je regulovaná fitness hodnota $i-$tého chromozómu v generácii t. Táto hodnota vždy leží v intervale $(0, 1)$.[4] Narozdiel od hrubej fitness hodnoty, neutralizuje nepriaznivý vplyv príliš veľkého rozdielu hrubých fitness hodnôt jedincov, a tak je jedným zo spôsobov riešenia problému uvedenému v sekcii 2.5.9.

2.4.4 Normalizovaná fitness hodnota

Normalizovaná fitness hodnota sa počíta zo štandardizovanej podľa:

$$n_i(t) = \frac{a_i(t)}{\sum_{k=1}^{M} a_k(t)} \tag{2.5}$$

Kde $n_i(t)$ je normalizovaná fitness hodnota $i-$tého chromozómu v generácii t, $a_i(t)$ regulovaná fitness hodnota $i-$tého chromozómu v generácii t, M počet chromozómov v populácií a $a_k(t)$ regulovaná fitness hodnota $k-$tého chromozómu v generácii t.

Táto fitness hodnota leží v intervale $(0, 1)$. Pre vhodnejšie chromozómy je táto hodnota väčšia. Súčet všetkých fitness hodnôt v populácii je rovný jednej[4]. Normalizovaná fitness hodnota sa prideľuje chromozómom pri použití ruletového spôsobu výberu jedincov (viď 2.5.2). Výpočet tohto typu je najzložitejší, pretože je nutné najprv vypočítať regulovanú alebo hrubú fitness hodnotu všetkých jedincov v populácii.

2.5 Algoritmus genetického programovania

Tento algoritmus predstavuje hľadanie najvhodnejšieho počítačového programu v priestore všetkých možných počítačových programov pre daný problém. V algoritme GP sú populácie o veľkosti stoviek až tisícok týchto programov (chromozómov) geneticky pestované, pričom sa využíva Darwinova téza prirodzeného výberu(viď 2.1). Toto genetické pestovanie sa vykonáva pomocou genetických operácií. Primárne genetické operácie sú kríženie a reprodukcia. Medzi sekundárne patrí mutácia, permutácia, upravovanie, zapuzdrenie a decimovanie. [4] Algoritmus GP je nasledujúci:

1. Vytvor počiatočnú populáciu
2. Ak je splnené kritérium pre ukončenie prejdi k bodu 8 inak pokračuj
3. Vyhodnoť fitness hodnotu pre každého jedinca v populácii
4. Vyber genetickú operáciu
5. Vyber príslušný počet jedincov z populácie
6. Vykonaj genetickú operáciu
7. Ak je počet jedincov v novej populácii dostatočný prejdi k 2 inak k 4
8. Stanov výsledok a skonči

Na obrázku 2.3 je graficky znázornený vyššie uvedený postup, kde Gen je aktuálna generácia, i je počet potomkov v novej populácii, M celkový počet jedincov v populácii, P_r je pravdepodobnosť reprodukcie, P_m pravdepodobnosť mutácie a P_k pravdepodobnosť kríženia jedincov.

2.5.1 Vytváranie počiatočnej populácie

Existuje niekoľko spôsobov generovania jedinca v populácii. Hlavným kritériom pri jeho vytváraní je maximálna hĺbka počet uzlov stromovej štruktúry reprezentujúcej chromozóm. Existujú dve základné metódy „plniaca“ a „rastová“[4].

„Plniaca“ metóda

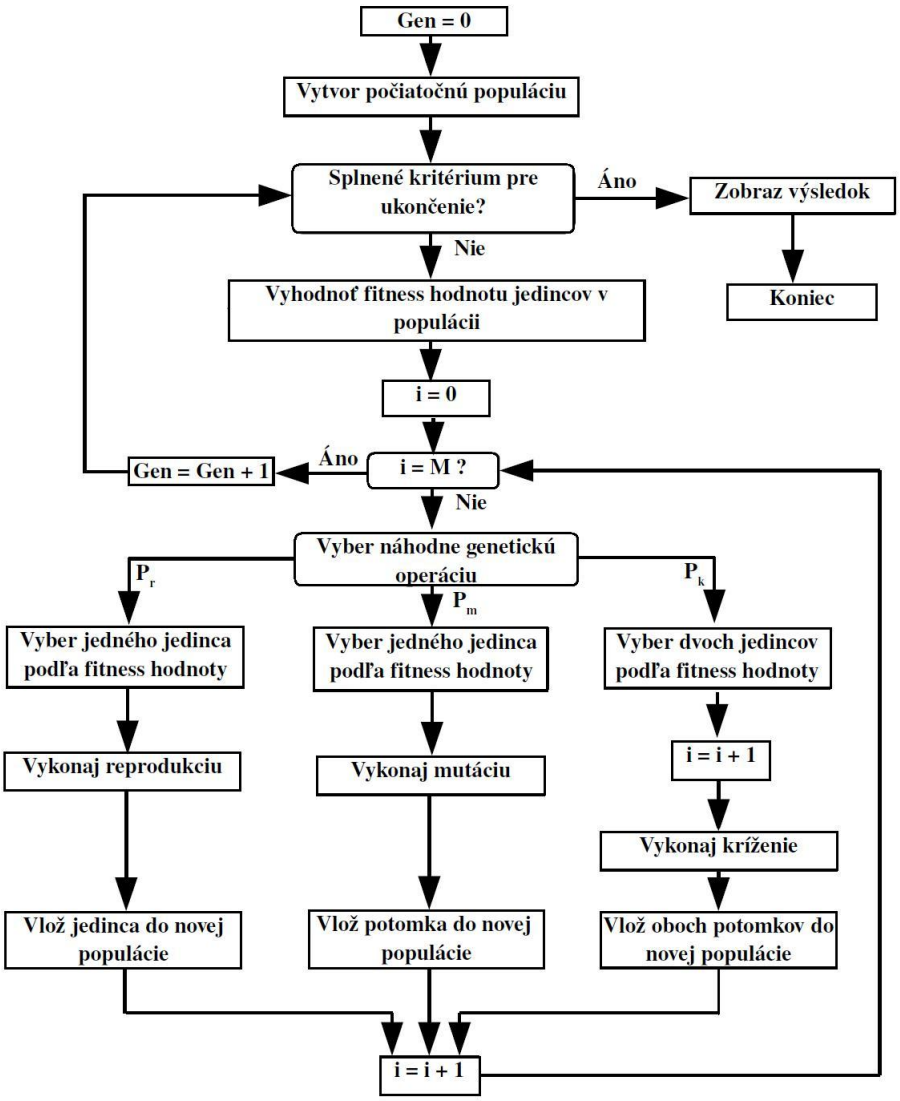
Metóda vytvára stromové štruktúry, pre ktorú platí, že dĺžka každej najkratšej cesty medzi koncovým bodom a koreňom stromu je rovná maximálnej povolenej hĺbke. Toto sa dá dosiahnuť zakázaním zvolenia terminálu pre uzly v hĺbke menšej ako maximálna hĺbka a pre každý uzol v tejto hĺbke zvoliť terminál[4].

„Rastová“ metóda

Týmto spôsobom sa vytvárajú stromy v populácii, ktorých tvar je rôznorodý. Avšak pre všetky stromy platí, že najkratšia cesta medzi koreňom a ľubovolným koncovým uzlom nie je dlhšia ako maximálna povolená hĺbka. Metóda povoľuje vybrať pre uzly v hĺbke menšej ako maximálna prvok zo zjednotenia množín operácií a terminálov. Pre uzly v maximálnej hĺbke je výber obmedzený na prvky z množiny terminálov. Táto metóda vytvorí väčšiu diverzitu ako plniaca, preto je aj častejšie používaná[4].

„Spádová pol na pol“ metóda

Táto metóda nedefinuje generovanie jedného stromu, ale celej populácie ako takej. Kombinuje plniacu a rastovú metódu, pričom vytvára rovnaký počet stromov pre všetky hĺbky od hodnoty 2 až po maximálnu povolenú hĺbku. Polovica stromov pre jednu hĺbku je vytvorená plnou a polovica rastovou metódou[2, 4].



Obrázek 2.3: Algoritmus GP

2.5.2 Výber jedincov pre genetické operácie

Určenie spôsobu výberu jedincov z populácie pre genetické operácie je jedno z najdôležitejších rozhodnutí GP. Výber je zodpovedný za rýchlosť evolúcie a často je obviňovaný v prípadoch, keď predčasná konvergencia znemožňuje úspech evolučného algoritmu. Metód pre výber jedincov je niekoľko. Rozdeľujú sa pritom do dvoch základných typov. Prvý typ sú metódy, ktoré vyberajú jedincov úmerne na veľkosti fitness hodnoty daného jedinca. Tieto metódy doposiaľ často používané, sa v poslednej dobe stretli s kritikou, pretože sú príčinou problému s tlakom na výber(viď 2.5.9)[2]. Preto je vhodné pri použití týchto spôsobov výberu prerozdeliť fitness hodnoty do intervalu [0,1].[14] Do tejto kategórie patrí napríklad ruleta. Na druhej strane metódy, ktoré nevyberajú jedincov úmerne na veľkosti fitness hodnoty sú pořadová selekcia, orezanie, turnaj.

Ruleta

Pokiaľ sa jedinci vyberajú ruletou je každému chromozómu v populácii na základe veľkosti jeho fitness hodnoty priradená určitá časť jednotkovej kružnice (prípadne jednotkovej úsečky). Potom sa vygeneruje náhodné číslo z intervalu [0,1]. Vyberie sa ten chromozóm, ktorému patrí interval na kružnici (úsečke), do ktorého toto číslo spadá[14]. Pravdepodobnosť P, že bude vybraný chromozóm s fitness hodnotou f je:

$$P = \frac{f}{\sum_{j=1}^{M} f_j}$$

kde M je počet chromozómov v populácii.

Pořadová selekcia

V pořadovej selekcii je pravdepodobnosť výberu chromozómu založená na poradí veľkosti jeho fitness hodnoty v celkovej populácii. Táto pravdepodobnosť je priradená chromozómu zvyčajne lineárnou funkciou, hoci je možné použiť aj exponenciálnu[2].

V prípade lineárnej funkcie sa napríklad chromozómu s najmenšou fitness hodonotou priradí hodnota 1, ďalšia väčšia hodnota dostane 2 atď., až po posledný chromozóm v populácii. Potom sa použije výber ruletou nad týmito priradenými hodnotami. Tento spôsob eliminuje problémy s klesaním genetickej diverzity v populácii[14].

Orezanie

Chromozómy sa zoradia podľa veľkosti a rozdelia na dve sekcie na základe parametru T, ktorý udáva z akej veľkej časti populácie sa budú vyberať rodičia. Pravdepodobnosť výberu chromozómu v sekcii s vyššou fitness hodnotou je rovnaká. Výber sa uskutočňuje buď deterministicky (každý jedinec sa stáva rodičom) alebo stochasticky (každý jedinec má rovnakú pravdepodobnosť selekcie)[1].

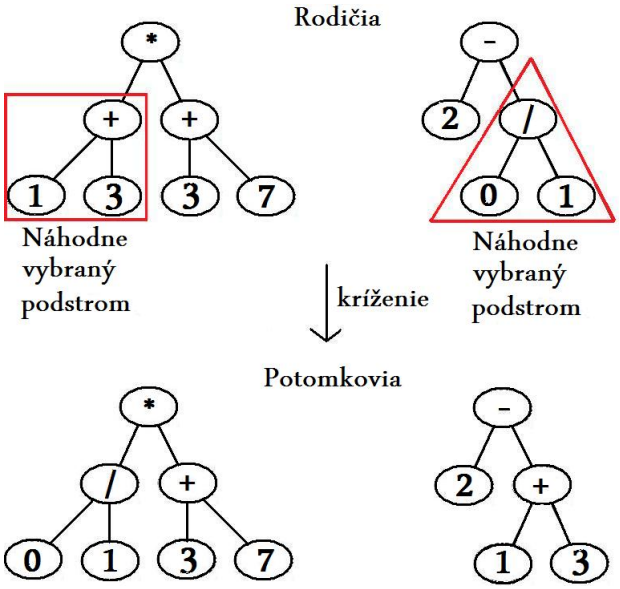
Turnaj

V poslednej dobe často využívaný spôsob, pretože nepotrebuje centralizované porovnávanie fitness hodnoty medzi všetkými chromozómami, čím podporuje paralelizáciu GP. V turnaji sa náhodne vyberie určitý počet jedincov z celej populácie (minimálne dvaja). Tento počet označuje veľkosť turnaju. Porovnajú sa ich fitness hodnoty a zvolí sa najlepší jedinec z tejto množiny[2].

2.5.3 Kríženie

Operácia kríženia (prípadne sexuálna rekombinácia) zabezpečuje pestrosť v populácii, tým že vytvára nových potomkov, ktorý obsahujú časti prevzaté od každého rodiča. Pre kríženie je nutné vybrať z aktuálnej populácie dvoch rodičov, pričom sa vytvoria dvaja potomkovia do novej populácie.

V oboch rodičovských chromozómoch, reprezentovaných koreňovým stromom, sa náhodne vyberie jeden podstrom. Prvý potomok vznikne nahradením zvoleného podstromu prvého rodiča, podstromom z druhého rodiča. A podobne druhý potomok nahradením zvoleného podstromu druhého rodiča, podstromom z prvého rodiča[4].



Obrázek 2.4: Kríženie

Kríženie je proces ktorý však môže pracovať tromi smermi: deštruktívne, neutrálne a konštruktívne. Pokiaľ pri operácii kríženia boli vytvorení potomkovia, ktorých fitness hodnoty sú menšie ako ich rodičov, tak táto operácia sa chovala deštruktívne. Naopak pokiaľ potomkovia mali fitness hodnoty väčšie ako rodičia kríženie bolo konštruktívne. Neutrálne správanie kríženia vytvára potomkov s podobnou fitness hodnotou ako ich rodičia.

Vo väčšine prípadov sa operácia kríženia správa deštruktívne a iba v malom množstve vytvára potomkov s fitness hodnotou väčšou ako ich rodičia. Napriek tejto skutočnosti, kríženie v konečnom dôsledku vedie k úspechu. Existujú však metódy ako tento deštruktívny efekt, znížiť. Najjednoduchší spôsob sa volá rekombinácia potomstva[2].

Rekombinácia potomstva

Metóda spočíva v tom, že namiesto jedného kríženia s dvoma rodičmi, sa vykoná väčší počet operácií sexuálnej rekombinácie a vytvorí sa viac potomkov. Z týchto potomkov sa vyberú dvaja najlepší a vložia sa do novej populácie. Môže sa však objaviť otázka prílišného spomalenia evolúcie, pretože sa je nutné vyhodnotiť 2N potomkov[3]. Tento problém sa rieši tak, že potomkovia sa vyhodnocujú len na zlomku trénovacích dát[2].

2.5.4 Reprodukcia

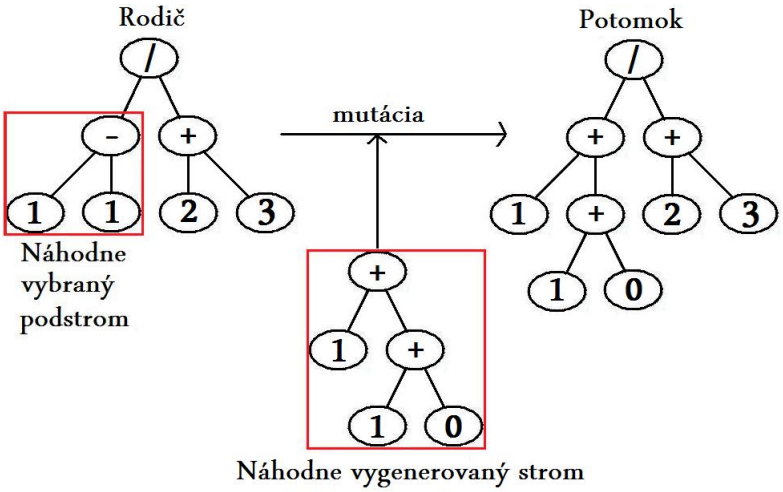
Operácia reprodukcie je základným kameňom Darwinovej tézy prirodzeného výberu a prežitia najlepšieho. V GP je táto operácia asexuálna – zúčastňuje sa jej len jeden rodič. Po zvolení chromozómu z aktuálnej populácie sa tento chromozóm bez zmeny skopíruje do novej.[4]

[3] N je počet operácií kríženia s rovnakými rodičmi

2.5.5 Mutácia

Hoci podľa Kozu[4] je táto operácia v GP nepotrebná, v [9] bolo dokázané, že mutácia dokáže priaznivo ovplyvniť evolúciu a zabezpečiť rôznorodosť génov(viď 2.2) v populácii.

Mutácia je asexuálna operácia. Postupuje tak, že v jednom chromozóme náhodne vyberie podstrom. Potom je vygenerovaný nový strom. Tento nový strom je následne vložený do chromozómu namiesto zvoleného podstromu[4].



Obrázek 2.5: Mutácia

Používa sa aj iný typ mutácie tzv. jedno-bodová mutácia. Táto mutácia nevyberá celý podstrom v chromozóme, ale len jeden uzol. Potom zamení hodnotu len tohto jedného uzla (terminál za iný terminál, funkciu za inú funkciu)[9].

2.5.6 Neutrálne zmeny v evolúcii

V GP poznáme dva druhy zmien, ktoré majú implicitne neutrálny vplyv na vhodnosť jedinca a to funkčná redundancia a intróny.

Pojmom funkčná redundancia sa označujú rôzne programy (genotypy), ktoré reprezentujú rovnaké funkcie. Napríklad nasledujúce programy reprezentujú rovnakú funkciu `x+2`:

- `P1: ((x+2)*x)/(x-0)`
- `P2: x+2-5*x+10-10*x/x+5*x`
- `P3: ((x+2)*(x+2))/(x+2+3*0)`

Genetická zmena z jedného programu do druhého (napr. z P1 do P2) má neutrálny efekt na funkčnosť. To znamená, že pokiaľ počas evolúcie dôjde k takejto premene, tak nemá žiadny efekt na chromozóm.

Intróny sú časti programu, ktoré sú sémanticky nadbytočné k správaniu programu. Napríklad:

- `P4: -x+x`

Pokiaľ by sa program P4 (intrón) odstránil nejakou genetickou operáciou, nemala by táto operácia efekt na správanie chromozómu[2, 13].

2.5.7 Elitizmus

Elitizmus je jednoduchá metóda, ktorá výrazne zlepšuje priebeh evolúcie a zamedzuje strate najlepšieho riešenia. Pred vytvorením novej populácie použitím genetických operácií sa najprv nájde najlepší jedinec v aktuálnej populácii a automaticky sa vloží do novej[14].

2.5.8 Ukončovacie kritériá

Evolučný proces v prírode nikdy nekončí. Evolúcia v GP však musí byť z praktických dôvodov ukončená, keď sú splnené ukončovacie kritériá. Zvyčajné kritérium je maximálny počet generácií. Ďalšie kritériá sa aplikujú špecificky na povahe problému[4]. Evolúcia sa napríklad môže ukončiť, ak niektorý chromozóm dosiahne určitej veľkosti fitness hodonty.

2.5.9 Problémy v genetickom programovaní

V genetickom programovaní sa stretáme s mnohými problémami. Jedným s ktorým sa treba vysporiadať v každom algoritme GP je „bloat“ efekt. Častým problémom je tiež tlak na výber jedincov z populácie.

Bloat efekt

Označuje sa ním rast dĺžky potomokov počas evolúcie. V GP použitím kríženie alebo mutácie môžu mať potomkovia väčšiu dĺžku ako ich rodičia a obvykle dĺžka potomkov narastá lineárne s počtom generácií.

Tento jav môže mať neželaný efekt na výkon GP (čím dlhší chromozóm, tým dlhší čas zaberie jeho vyhodnotenie). Riešiť sa dá penalizáciou príliš dlhých chromozómov alebo prispôsobiť operácie kríženia a mutácie tak aby vytvárali potomkov s len s určitou maximálnou dĺžkou[14].

Problémy tlaku výberu

Tento problém sa objavuje v aplikáciach genetického programovania využívajúcich fitness hodnote úmerný spôsob výberu rodičov z populácie. Jedná sa o problém, kedy niektorí jedinci v populacii majú príliš veľkú fitness hodnotu v porovnaní so zvyškom populácie. Toto spôsobuje, že sú vyberaný ako rodičia až príliš často a dochádza k úbytku genetickej diverzity v populácii a k predčasnej konvergencii[8].

Kapitola 3

Kartézske genetické programovanie

V tejto kapitole budú uvedené základné fakty a informácie týkajúce sa kartézskeho genetického programovania.

3.1 Úvod do kartézskeho genetického programovania

Kartézske genetické programovanie (CGP) bolo prvý krát uvedené J. Millerom v roku 1999.[10] Tento typ evolučného programovania nepracuje na základe Darwinovej téze prirodzeného výberu, ale na jave genetického driftu [1]. Bolo pozorované, že tento jav hrá dôležitú úlohu v procese evolúcie. To znamená, že hybnou silou evolúcie nie je (len) prirodzený výber, ale náhodné zmeny v neaktívnych génoch (teória neutralizmu). CGP stavia na tomto základe a zavádza radu výhod oproti GP[13] .

3.2 Chromozóm

Chromozóm v CGP je reprezentovaný orientovaným acyklickým grafom. Kóduje sa do lineárneho reťazca celých čísel s pevnou dĺžkou, ktorý kóduje funkcie jednotlivých uzlov a ich vzájomné prepojenia. Hoci dĺžka tohto reťazca je nemenná, neznamená to, že dĺžka grafu je tiež pevná. Do orientovaného grafu (riešenia problému) nie sú totiž zapojené všetky uzly. Toto dovoľuje určitým častiam chromozómu zostať neaktívnymi, čo vedie neutrálnemu vplyvu na veľkosť fitness hodnoty jedinca, javu zvanému *neutralita*[12]. Tento typ explicitnej neutrality sa ukázala ako veľký prínos k evolučnému procesu.

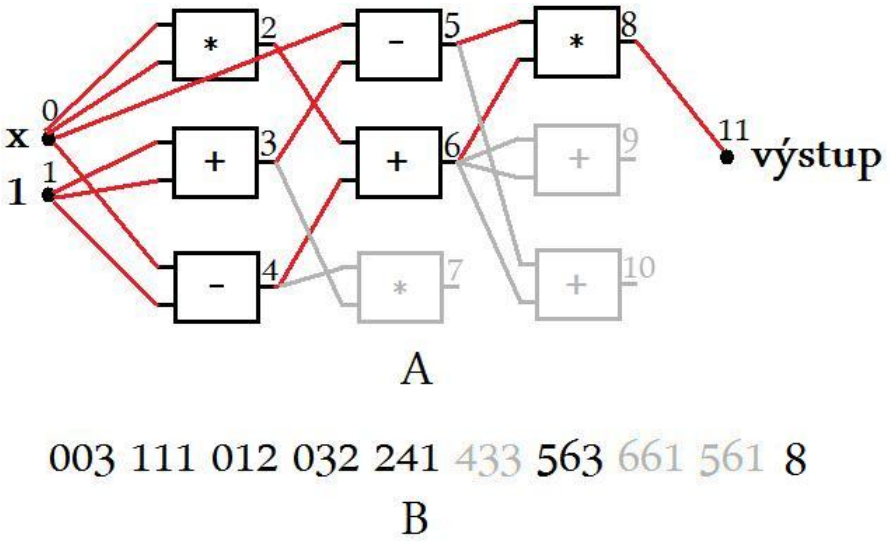
Každý program v CGP (chromozóm) je zložený z množiny $G, n_i, n_o, n_n, F, n_f, n_r, n_c, l$, kde G reprezentuje genotyp a sám je množinou celých čísel reprezentujúcich n_i vstupov programu, n_n spojení vstupov a uzlov a uzlov medzi sebou, n_o spojení uzlov a výstupov tohto programu (Narozdiel od GP kde je chromozóm reprezentovaný koreňovým stromom, chromozóm v CGP môže mať viac výstupov). Množina F reprezentuje n_f funkcií uzlov. Počet uzlov v riadku a v stĺpci je daný n_r,n_c. Možnosti vnútorného prepojenia v chromozóme určuje „level – back“ parameter (l – back parameter). Každý uzol má tiež určitý počet vstupov a výstupov. V podstate, chromozóm v CGP je mriežka navzájom prepojených uzlov[7]. V praxi sa však ukázalo, že CGP je efektívnejšie pokiaľ počet riadkov je rovný jednej.[13]

Takáto forma reprezentácie chromozómu umožňuje implicitné viac násobné použitie časti chromozómu, keďže výstup jedného uzlu môže byť napojený na vstupy viacerých

[1] Genetický drift je náhodná zmena vo frekvencii alel v populácii, ktoré nie sú späté s prirodzeným výberom[13]

uzlov. Je to výhoda oproti klasickému GP, kde totožné podstromy musia byť vytvorené samostatne.[11]

Pri prepojovaní platia určité pravidlá. Aby nemohla vzniknúť spätná väzba, vstupy uzlov sa môžu pripájať len na výstupy uzlov z predchádzajúceho stĺpca alebo na akýkoľvek vstup chromozómu[10].



Obrázek 3.1: Chromozóm v CGP

Na obrázku 3.1 je v diagrame A sú zobrazené vnútorné prepojenia chromozómu v matici 3x3 s dvoma vstupmi a jedným výstupom. Šedo označené bloky nie sú zaradené do riešenia, čiže sú redundantné. V diagrame B je tento chromozóm zapísaný formou lineárneho reťazca celých čísel, kde jedna trojica čísel reprezentuje jeden blok. Prvé dve čísla v trojici označujú výstupy blokov z predchádzajúcich stĺpcov pripojené na vstup daného bloku. Tretie číslo definuje funkciu nad týmito vstupmi (1 pre $+$, 2 pre $-$, 3 pre $*$). Šedo zapísané bloky sú nadbytočné. Posledné číslo označuje poradové číslo bloku, ktorý je napojený na výstup chromozómu.

3.2.1 Level-back parameter

Tento parameter určuje o koľko stĺpcov vpred je možné pripojiť výstup jedného uzlu na vstup iného. L – back parameter môže nadobúdať hodnoty z intervalu $< 1, n_c >$. Pokiaľ je l – back parameter rovný jednej, vstup každého uzlu sa môže pripojiť len na výstup uzlu z predchádzajúceho riadku. Ak je l – back parameter rovný n_c je dosiahnutá maximálna konektivita, čiže je povolené prepojiť vstup uzlu s ktorýmkoľvek výstupom uzlu z predchádzajúceho stĺpca alebo vstupu chromozómu[10].

3.3 Algortimus kartézskeho genetického programovania

Algortimus kartézskeho genetického programovania využíva evolučnú stratégiu $(1 + \lambda)$. Tento spôsob vytvára novú populáciu z jedného rodiča a λ jeho potomkov. Jediným spôsobom ako vytvoriť potomka v algoritme CGP je mutácia[10]. Operácia kríženia bola síce

predstavená v [3], avšak pre jej aplikáciu bolo nutné pozmeniť reprezentáciu chromozómu a jej prínos bol viditeľný len v počiatočných generáciách evolúcie[3]. Algoritmus CGP je nasledujúci:

1. Vytvor počiatočnú populáciu o veľkosti $(1 + \lambda)$
2. Ak je splnené kritérium pre ukončenie prejdi k bodu 6 inak pokračuj
3. Vyhodnoť fitness hodnotu pre každého jedinca v populácii
4. Vyber najlepšieho jedinca v populácii
5. Vytvor λ potomkov najlepšieho jedninca, vlož ich do novej populácie spolu s týmto jedincom a prejdi k 2
6. Stanov výsledok a skonči

3.3.1 Výber najlepšieho jedinca

Jedinec sa môže vybrať dvojakým spôsobom. Prvý, vždy sa vyberie jedinec z najväčšou fitness hodnotou v populácii. Druhý, pokiaľ niektorý potomok má rovnakú fitness hodnotu ako rodič, uprednostní sa tento jedinec ako rodič budúcej generácie. Pokiaľ by viacero potomkov bolo rovnako dobrých ako rodič, vyberie sa náhodne medzi nimi. Tento prístup sa volá neutrálnosť, a značne zlepšuje efektivitu CGP[13].

3.3.2 Mutácia

Táto operácia v CGP pracuje inak ako v GP. Je riadená parametrom udávajúcim percentuálny počet mutovaných génov. Mutácia jedného génu znamená, zmenu jedného čísla v chromozóme na iné. Samozrejme z prípustnej množiny. Ak mutuje gén reprezentujúci funkciu, jeho hodnota sa zmení len na hodnotu, ktorá reprezentuje funkciu. Ak sa mení gén označujúci pripojenie vstupu, zmutuje na hodnotu rešpektujúcu pravidlá CGP. (Pripojí sa len na uzol z predchádzajúcich stĺpcov, berúc do úvahy $l-$back parameter)[10].

Kapitola 4

Návrh a implementácia programu

V súčasnosti je vytvorených veľa aplikácií realizujúcich princípy evolučných algoritmov schopných riešiť zložité problémy v rôznych oblastiach vedy a hospodárstva. Existuje viac prístupov využívajúcich evolúciu ako spôsob hľadania riešenia skúmaného problému. Tiež sa hľadajú nové alebo vylepšujú už známe algoritmy. Všetko v snahe zefektívniť evolúciu. Avšak je veľmi zložité porovnať rôzne metódy využívajúce tento postup, pretože niektoré realizácie sú vhodnejšie pre určitú skupinu problémov viac ako iné.

Mojou úlohou je porovnať GP a novo predstavený spôsob CGP na úlohách symbolickej regresie. Oba typy genetického programovania som sa snažil maximálne zefektívniť a vylepšiť. V tejto kapitole bude popísaný spôsob implementácie a návrhu vytvorených aplikácií GP a CGP. Budú uvedené takiež praktiky použité k zefektívneniu a vylepšeniu týchto algoritmov.

4.1 Programovací jazyk

Pre implementáciu programov GP a CGP som zvažoval nasledujúce programovacie jazyky: C/C++, Java, C#. Pretože algoritmus GP,CGP a hlavne výpočet fitness hodnoty v týchto algoritmoch je veľmi náročný na výpočetnú silu procesoru[4], potreboval som aby výsledný kód bol čo najrýchlejší. Na tomto základe som sa rozhodol pre použitie jazyka C pre obe aplikácie. Medzi dôvody hovoriace v prospech tohto programovacieho jazyka je jeho jednoduchosť, dostupnosť, flexibilita, prenositelnosť a efektivita, umožňujúca dobrý výkon programov.[6]

4.2 Aplikácia GP

Zdrojový kód programu je rozdelený do dvoch samostatných súborov: `gp.c` a `gp.h`. V `gp.c` sú uvedené definície štruktúr, makrá, všetky globálne premenné, a hlavičky funkcií. V `gp.c` je kód programu.

V nasledujúcej časti je popísaný spôsobo implementácie aplikácie GP bod po bode od spôsobu implmentácie chromozómu až po celkový algoritmus.

4.2.1 Návrh implementácie chromozómu a populácie

Pre implementáciu chromozómu som zvolil stromovú štruktúru, binárny koreňový strom, podobne ako je popísané v sekcii 2.2, pretože práca s týmito štruktúrami je použitím re-

kurzie rýchla a jednoduchá. Návrh je Keďže som chcel maximalizovať výkon, snažil som sa vyhnúť neustálej alokácii pamäti pre novovzniknuté chromozómy potomkov, a dealokácii už nepotrebných chromozómov zo starej generácie. To bol dôvod pre rozhodnutie, že pamäť pre všetky chromozómy naalokujem dopredu na začiatku behu programu.

Znamená to, že pred začiatkom evolúcie, alokujem priestor pre dve generácie chromozómov (generácia rodičov a generácia potomkov). Každý chromozóm má alokovaný priestor pre všetky uzly až do maximálnej povolenej hĺbky. Hĺbka sa nastavuje zmenou hodnoty makra `DEPTH` v súbore `gp.h`. Avšak nie každý alokovaný uzol musí byť použitý. Tým je pri operáciách s týmito stromami znemožnený tradičný princíp práce, kedy strom končil ak `strom->lavyPodstrom == NULL and strom->pravyPodstrom == NULL`.

Riešenie tohto problému poskytol Readov lineárny kód. Využil som princíp Readovho kódovania(viď 2.2.2) a pridal do každého uzlu okrem ukazatela na ľavý,pravý podstrom a hodnoty uzlu aj celočíselnú hodnotu, ktorá udáva počet použitých podstromov v chromozóme. Pseudokód štruktúrí reprezentujúcej uzol stromu:

```
struct {
        data;
        pocetOperadnov;
        uzol *lavyPodstrom;
        uzol *pravyPodstrom;
       }uzol;
```

Tabulka 4.1: Uzol stromu

Ak premenná `pocetOperadnov` je rovná nule, reprezentuje tento uzol v chromozóme terminál a v premennej `data` je hodnota tohto terminálu. Pokiaľ je v `pocetOperadnov` hodnota väčšia ako nula, v premennej `data` je hodnota reprezentujúca funkciu, s počtom operandov uvedených v premennej `pocetOperadnov`. Ak má funkcia jeden operand (v `pocetOperadnov` je hodnota jedna) vyhodnocuje sa len ľavý podstrom. V prípade funkcie s dvoma operandami sa táto funkcia vykoná nad hodnotami vrátenými z oboch podstromov. Funkciám s viac ako dvoma operandami nebol tento program prispôsobený.

Populácia je tvorená globálnym poľom ukazatelov na koreňové uzly jednotlivých chromozómov. Počet jedincov v populácii sa upravuje hodnotou makra `POPULATION` v hlavičkovom súbore `gp.h`. V skutočnosti je veľkosť tohto poľa `POPULATION+1`. Zväčšenie o jednotku si vyžiadalo použitie genetických operácií. Niektoré operácie totiž vytvárajú len jedného a niektoré dvoch potomkov, preto môže nastať situácia, kedy v novej populácii zostane len jedno voľné miesto a bude použitá genetická operácia vytvárajúca dvoch potomkov. Miesto toho aby som sa v každej takejto operácii pýtal, či má miesto pre vloženie druhého potomka, tento potomok bude jednoducho do novej populácie vložený, hoci nikdy nebude použitý.

V programe sú vytvorené dve populácie. Jedna je populácia rodičov, pomocou ktorých sú genetickými operáciami vytváraný jedinci do populácie potomokov. Pri vytváraní ďalšej generácie sa populácie jednoducho vymenia a z potomkov sa stanú rodičia.

Zo 4.1 je vidieť, že v chromozóme nie je miesto pre uloženie jeho fitness hodnoty. Bolo by plýtvanie miestom ak by som do každého uzlu vložil premennú uchovávajúcu fitness hodnotu chromozómu. Preto je vytvorené globálne pole „hlavičiek“ chromozómov. Pseudokód :

V premennej `fitnessHodnota` je uložená fitness hodnota jedného chromozómu. O ostatných premenných sa bude hovoriť neskôr.

```
struct {
        fitnessHodnota;
        lavyOkraj;
        pravyOkraj;
        poradie;
       }hlavicka;
```

Tabulka 4.2: Hlavička chromozómu

Voľba funkcií a terminálov

Implementoval som 5 aritmetických operácií a to: sčítanie, odčítanie, násobenie, delenie a faktoriál. Princíp uzavretosti (viď 2.2.1) je pre prvé tri operácie automaticky zaručená.

Operáciu delenia som implementoval tak, že pokiaľ je deliteľ rôzny od nuly, vráti výsledok delenia. Ak by deliteľ bol rovný nule, výsledok tejto operácie bude len delenec.

Faktoriál je realizovaný nasledujúcim spôsobom. Argument tejto operácie sa najprv zaokrúhli smerom nadol, aby sa predišlo počítaniu faktoriálu desatinného čísla. Ak je argument z intervalu $< 0, 170 >$ výsledok je faktoriál daného čísla. Pre čísla z intervalu $< -170, 0)$ je vrátený faktoriál absolútnej hodnotý tohto čísla. Pre čísla mimo tohto intervalu je výsledok hodnota makra `VERYLARGEDOUBLE`. Dôvodom je fyzické obmedzenie premenných typu `double`, pretože nedokážu udržať čísla väčšie ako hodnota faktoriálu zo 170(overené v praxi) a nadobudnú hodnotu `+-INIFINITY` čo narúšalo ďalšie vyhodnocovanie.

Do množiny terminálov som zvolil celé čísla z intervalu $< -5, 5 >$, narozdiel od Kozu, ktorý volil terminály náhodnou generáciou čísel z určitého intervalu[4]. Dôvod je ten, že tento prístup (generovanie náhodných čísel) nie je možný v CGP a pri porovnávaní GP a CGP chcem aby mali oba spôsoby rovnaké podmienky pre evolúciu.

Metóda generovania chromozómu

Ako spôsob vytvárania chromozómov a tiež celej populácie som zvolil rastovú metódu. Od hĺbky väčšej ako 1 sa uzly začínajú vyberať z množiny terminálov z pravdepodobnosťou danou hodnotou makra `ENDBRANCH` a funkcií s pravdepodobnosťou $1 -$ `ENDBRANCH`. Čím menšia bude hodnota tohto makra, tým budú mať stromy väčšiu hĺbku. V hĺbke jedna sa volí len z množiny funkcií a v maximálnej povolenej hĺbke z len množiny terminálov.

4.2.2 Realizácia algoritmu GP

Algoritmus aplikácie GP prebieha podľa popisu v sekcii 2.5. Avšak miesto obyčajného kríženia som implementoval metódu rekombinácie potomstva a tiež bodovú mutáciu. Pravdepodobnosti jednotlivých genetických operácií sa nastavujú priamo jednoduchým zásahom do kódu programu. Aby som zabránil efektu „bloat“, operácie kríženia a mutácií sú implementované tak, že rešpektujú maximálne povolenú hĺbku a nevytvárajú hlbšie stromy. V nasledujúcej sekcii sú uvedené podrobnejšie informácie o implementácií jednotlivých zložiek evolučného algoritmu.

4.2.3 Výpočet hrubej fitness hodnoty chromozómu v aplikácii

Keďže aplikácia GP má slúžiť k symbolickej regresii, vhodnosť jedinca sa odvíja od rozdielu y – hodnôt trénovacej množiny a hodnôt chromozómu pre príslušné x – hodnoty z trénovacej množiny. Fitness hodnota chromozómu sa počíta:

$$f_h = \frac{1}{\sum_{i=1}^{N} |y_i - ch(x_i)|} \tag{4.1}$$

Kde f je fitness hodnota chromozómu ch, N je počet hodnôt v trénovacej množine, y_i i – tá y – hodnota v trénovacej množine, $ch(x_i)$ hodnota chromozómu ch pre i – tú x – hodnotu z trénovacej množiny.

Pokiaľ vo vyhodnocovaní chromozómu ch nebude možné použiť len štandardné matematické operácie, napríklad dôjde k deleniu nulou alebo k výpočtu faktoriálu záporného čísla, tak výsledná fitness hodnota tohto chromozómu bude zmenšená na polovicu. Táto úprava sa dá nastaviť zmenou hodnoty makra `LOWERFITNESS` v hlavičkovom súbore `gp.h`.

4.2.4 Výber chromozómov pre genetické operácie

V snahe zvoliť čo najvhodnejšiu metódu výberu chromozómov pre genetické operácie som implemtoval nasledujúce tri spôsoby.

- Ruleta — tento spôsob výberu je úmerný veľkosti fitness hodnoty chromozómu. Aby som predišiel prílišnému uprednostňovaniu jedincov v populácii , ktorí majú veľkú fitness hodnotou, každému chromozómu je pridelená regulovaná fitness hodnota, ktorá by mala tento problém vyriešiť, podľa :

  $$f = \frac{1}{1 + f_h} \tag{4.2}$$

  Kde f je fitness hodnota chromozómu ch a f_h je jeho hrubá fitness hodnota.

  Potom je každý jedinec ohodnotený normalizovanou fitness hodnotou a je mu pridelený úsek na jednotkovej priamke rovný tejto hodnote. Tento úsek je uložený v hlavičke chromozómu v premenných `lavyOkraj`,`pravyOkraj`.

- Pořadová selekcia — výber už nie je úmerný fitness hodnote, a preto nie je nutné používať regulovanú fitness hodnotu. Každému chromozómu je priradená hrubá fitness hodnota. Na základe poradia veľkosti tejto hodnoty medzi fitness hodnotami v celej populácii je mu pridelené poradie. Toto sa zaznamená do premennej `poradie` v hlavičke chromozómu. Jedinec s najmenšou fitness hodnotou dostane poradie rovné jednej, jedinec ktorého fitness hodnota je v populácii najväčšia dostane hodnotu rovnú počtu jedincov v populácii. Jedinci s rovnakou fitness hodnotou majú rovnaké poradie. Potom sa pokračuje ako pri výbere ruletou, ale miesto regulovanej fitness hodnoty sa použije poradie.

- Turnaj — v tomto spôsobe sa chromozómy tiež ohodnocujú hrubou fitness hodnotou. Z populácie sa náhodne vyberie osem jedincov, z ktorých sa vyberú vždy dvaja najlepší. V prípade ak operácia potrebuje len jedného rodiča (mutácia, reprodukcia) použije sa len najlepší.

Následne som ich testoval na troch príkladoch symbolickej regresie rôznej náročnosti. Snažil som sa nájsť predpisy nasledujúcich rovníc:

$$f_1(y) = x^2 + 2x - 17 \tag{4.3}$$

$$f_2(y) = \frac{2x^5 - 4x^3 + 2x}{2x^3 + x^2 - 14} \tag{4.4}$$

$$f_3(y) = 3x^4 - 8x^3 + 2x \tag{4.5}$$

Algoritmus GP som 50 – krát pustil postupne pre 1000, 2000 a 3000 generácií. Pre ka pre všetky spôsoby výberu a meral úspešnosť jednotlivých metód. Parametre GP boli nasledovné:

- veľkost populácie: 150
- pravdepodobnosť kríženia: 90 %
- pravdepodobnosť reprodukcie: 10 %
- maximálna hĺbka jedinca: 8
- použité matematické operácie: $+, -, /, *$
- použité terminály: celé čísla z množiny $< -5, 5 >$
- použitý elitizmus

Trénovacia množina obsahovala 20 hodnôt. Úspešnosť jednotlivých behov som meral na základe súčtu absolútnych hodnôt odchyliek nájdeného riešenia od trénovacej množiny. Pre funkciu 4.3 bol beh aplikácie úspešný ak tento súčet bol rovný alebo menší ako 1. Najlepšia dosiahnutá úspešnosť pre rovnaký počet generácií je v každej tabuľke vyznačené hrubým písmom.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Turnaj	74 %	**88 %**	**86 %**
Ruleta	**88 %**	66 %	**86 %**
Pořadová selekcia	66 %	82 %	**86 %**

Tabulka 4.3: Úspešnosť nájdenia riešenia aproximácie funkcie 4.3 v závislosti od výberu chromozómov pre vybraný počet generácií

Pre funkciu 4.4 bol beh aplikácie úspešný ak súčet bol rovný alebo menší ako 10.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Turnaj	56 %	64 %	60 %
Ruleta	72 %	64 %	68 %
Pořadová selekcia	**90 %**	**86 %**	**92**

Tabulka 4.4: Úspešnosť nájdenia riešenia aproximácie funkcie 4.4 v závislosti od výberu chromozómov pre vybraný počet generácií

Pre funkciu 4.5 bol beh aplikácie úspešný ak súčet bol rovný alebo menší ako 1000.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Turnaj	20 %	6 %	12 %
Ruleta	12 %	8 %	20 %
Pořadová selekcia	**24 %**	**46 %**	**60 %**

Tabulka 4.5: Úspešnosť nájdenia riešenia aproximácie funkcie 4.5 v závislosti od výberu chromozómov pre vybraný počet generácií

Predpokladám, že výber spôsobom turnaja zlyhával pretože nemohol využiť elitizmus, keďže jedinci sa vyberali z populácie náhodne. Toto mohlo viesť k slabším výsledkom oproti ostatným dvom spôsobom. Je taktiež možné, že som zvolil príliš malú veľkosť turnaja, avšak tento problém už naďalej skúmať nebudem. Pri rulete sú vidieť slabnúce výsledky s rastúcim počtom generácií. Zastávam názor, že pri tomto spôsobe výberu dochádzalo k poklesu genetickej diverzity, najmä v neskorších generáciách, kedy niekoľko málo jedincov mohlo dosianhuť značne vyššiu fitness hodnotu vzhľadom k ostatnej populácii. Toto viedlo k predčasnej konvergencii. Naproti tomu pořadová selekcia mala najvyššiu úspešnosť a s rastúcim počtom generácií mala táto úspešnosť tendenciu rásť. Toto a aj najlepšie výsledky vo všetkých testoch ma presvedčili k použitiu tejto metódy v mojej aplikácii.

4.2.5 Voľba a návrh genetických operácií

V tejto sekcii uvediem akým spôsobom sú jednotlivé operácie realizované. Ďalej budú tiež uvedené testy na ktorých základe, nájdem najlepšiu kombináciu týchto operácií.

- Kríženie — Operácia kríženia je implentovaná spôsobom uvedeným v sekcii 2.5.3. Podľa Kozovho vzoru[4] je 90 percentná šanca, že ako uzly k výmene budú vybraté uzly reprezentujúce funkciu. Vo zvyšných 10 – tich percentách sú zvolené terminálne uzly. Zmena tohto nastavenie je možná prepisom hodnoty makra `FUNC` hlavičkovom súbore `gp.h`.

  V snahe zabezpečiť čo najväčšiu diverzitu génov som však narozdiel od Kozu[4], nepovolil aby jeden chromozóm zastupoval oboch rodičov. To znamená, že prvý rodič je náhodne vybraný z populácie pomocou pořadové selekce a voľba druhého rodiča sa opakuje pokým tento druhý chromozóm – rodič bude rovnaký ako prvý.

- Rekombinácia potomstva — rekombinácia potomstva pracuje na základe postupu uvedeného v sekcii 2.5.3. Počet operácií kríženia v rámci tejto metódy je v programe

pevne nastavených na 4, vytvárajúc tým osem potomkov. Z týchto potomkov sú následne vybratí dvaja najlepší a vložení do novej populácie.

Počet trénovacích dát pre výpočet fitness hodnôt vytvorených potomkov som zvolil na jednu desatinu z celkového množstva hodnôt z trénovacej množiny. Tieto trénovacie hodnoty sú náhodne vybraté, avšak pre všetkých potomkov vytvorených rekombináciou sú rovnaké.

- Mutácia — V programe sú implementované dva druhy mutácií, uvedených v sekcii 2.5.5. Klasická, kde sa náhodne vyberie uzol chromozómu a celý podstrom počínajúc týmto uzlom bude nahradený náhodne vygenerovaným podstromom. Druhý typ je bodová mutácia. V tomto prípade sa vyberie uzol a je nahradená len hodnota tohto uzlu (funkcia funkciou, terminál terminálom).

  Pokiaľ sa mutuje uzol reprezentujúci funkciu s jedným operandom na funkciu s dvoma, je ešte náhodne vygenerovaný podstrom, ktorý sa stane druhým operadnom tejto funkcie. V opačnom prípade,pri zmene operácie s dvoma operandami na operáciu s jedným, je pre tento uzol jednoducho prepísaná hodnota v premenne udávajúcej počet operandov na 1 a pravý podstrom sa pri výpočte hodnoty chromozómu zanedbáva. Pravdepodobnosti výberu uzlu reprezentujúceho funkciu alebo terminálu je rovnaký ako v prípade kríženia.

Kríženie alebo rekombinácia potomstva?

Rozhodnutie či použiť kríženie alebo rekombináciu potomstva som založil na testoch rovnako ako v sekcii 4.2.4. Použitie rekombinácie bolo testované tiež na rovnakých problémoch. Keďže výsledky použitia kríženia spolu s pořadovou selekciou som už získal, nebolo nutné tieto testy zopakovať. Nasledujúce testy prebiehali rovnako ako v 4.2.4 pre 1000, 2000, 3000 generácií, taktiež s rovnakými parametrami evolúcie, až na to že miesto kríženia bola použitá rekombinácia potomstva.

Výsledky pre funkciu 4.3, beh je úspešný ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 1:

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Kríženie	76 %	78 %	**90 %**
Rekombinácia	**80 %**	**82 %**	**90 %**

Tabulka 4.6: Úspešnosť nájdenia riešenia aproximácie funkcie 4.3 v závislosti od druhu genetickej operácie pre vybraný počet generácií

Výsledky pre funkciu 4.4. Kritérium úspešnosti je sprísnené, beh je úspešný ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 5.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Kríženie	**56 %**	62 %	74 %
Rekombinácia	50 %	**76**	**76 %**

Tabulka 4.7: Úspešnosť nájdenia riešenia aproximácie funkcie 4.4 v závislosti od druhu genetickej operácie pre vybraný počet generácií

Výsledky pre funkciu 4.5. Kritérium úspešnosti je sprísnené, beh je úspešný ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 500.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Kríženie	16 %	**38 %**	56 %
Rekombinácia	**22 %**	36 %	**64**

Tabulka 4.8: Úspešnosť nájdenia riešenia aproximácie funkcie 4.5 v závislosti od druhu genetickej operácie pre vybraný počet generácií

Výsledky oboch prístupov sú porovnateľné. Prepokladám, že dôvodom môže byť malý počet trénovacích dát, ktoré obmedzuje čiastočné vyhodnotenie chromozómov v procese rekombinácie v našom prípade len na dve hodnoty. Avšak rekombinácia potomstva mala vo väčšine prípadov o niečo vyššiu úspešnosť. Časovú náročnosť pri porovnávaní oboch prístupov som sa rozhodol nebrať do úvahy, hoci spomalenie aplikácie pri použití rekombinácie bolo podľa mojich skúseností zanedbateľné. Zvolil som teda použitie rekombinácie potomstva a nie len použitie obyčajného kríženia. V ďalších testoch bude automaticky použitá rekombinácia potomstva a výber pořadovou selekciou bez explicitného oznámenia.

Výber druhu mutácie

Aký druh mutácie prináša najlepšie výsledky alebo či vôbec je mutácia prospešná pre evolúciu som tiež podrobil testom. Znova som testoval hľadanie predpisov rovnakých funkcií ako v predchádzajúcich sekciách. Výsledky aplikácie bez použitia mutácie som získal už pri testoch rekombinácie potomstva, takže nie je nutné ich opakovať. Znova som opakoval 50 behov pre 1000, 2000, 3000 generácií pre všetky tri funkcie, ale tento krát bola pravdepodobnosť reprodukcie znížená na 7 % a vo zvyšných 3 % sa uskutoční niektorá operácia mutácie. Ostatné nastavenia boli rovnaké. Testoval som päť konfigurácií mutácií:

- 1. kofigurácia: pravdepodobnosť uskutočnenia obyčajnej mutácie je 3 %, bodová mutácie nie je použitá
- 2. kofigurácia: pravdepodobnosť uskutočnenia bodovej mutácie je 3 %, obyčajná mutácie nie je použitá
- 3. kofigurácia: pravdepodobnosť uskutočnenia obyčajnej mutácie je 1.5 %, pravdepodobnosť uskutočnenia bodovej mutácie je 1.5 %
- 4. kofigurácia: pravdepodobnosť uskutočnenia obyčajnej mutácie je 1.5 %, bodová mutácie nie je použitá, pravdepodobnosť reprodukcie 8.5 %
- 5. kofigurácia: pravdepodobnosť uskutočnenia bodovej mutácie je 1.5 %, obyčajná mutácie nie je použitá, pravdepodobnosť reprodukcie 8.5 %

Výsledky pre funkciu 4.3, beh programu je úspešný ak ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 1:

Výsledky pre funkciu 4.4. Je stále použité sprísnené kritérium úspešnosti a beh je úspešný ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 5.

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Bez mutácií	80 %	82 %	90 %
1. kofigurácia	76 %	**98 %**	80 %
2. kofigurácia	66 %	92 %	90 %
3. kofigurácia	92 %	**98 %**	**98 %**
4. kofigurácia	88 %	80 %	90 %
5. kofigurácia	**100 %**	90 %	90 %

Tabulka 4.9: Úspešnosť nájdenia riešenia aproximácie funkcie 4.3 v závislosti od druhu a množstva operácií mutácie pre vybraný počet generácií

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Bez mutácií	50 %	76 %	76 %
1. kofigurácia	56 %	76 %	76 %
2. kofigurácia	56 %	70 %	78 %
3. kofigurácia	**64** %	80 %	**80 %**
4. kofigurácia	48 %	**82 %**	70 %
5. kofigurácia	50 %	74 %	**80 %**

Tabulka 4.10: Úspešnosť nájdenia riešenia aproximácie funkcie 4.4 v závislosti od druhu a množstva operácií mutácie pre vybraný počet generácií

Výsledky pre funkciu 4.5. Rovnako ako v predchádzajúcej situácii je použité sprísnené kritérium a beh je úspešný ak odchylka riešenia od trénovacích dát je menšia alebo rovná ako 500:

	Úspešnosť v %		
	1000 generácií	2000 generácií	3000 generácií
Bez mutácií	22 %	36 %	64 %
1. kofigurácia	22 %	40 %	60 %
2. kofigurácia	**26 %**	**42 %**	60 %
3. kofigurácia	**26 %**	40 %	**70 %**
4. kofigurácia	18 %	34 %	68 %
5. kofigurácia	22 %	40 %	64 %

Tabulka 4.11: Úspešnosť nájdenia riešenia aproximácie funkcie 4.5 v závislosti od druhu a množstva operácií mutácie pre vybraný počet generácií

Z vyššie uvedených výsledkov je vidieť, že mutácie v priebehu evolúcie vo väčšine prípadov pôsobia kladne. Najmä pri nižšom počte generácií. Väčším problémom je určiť, ktorú z konfigurácií by bolo najvýhodnejšie použiť pretože sa v rôznych prípadoch správali rôznym spôsobom. Testy neukázali žiadnu, ktorá by vo všetkých situáciách produkovala najlepšie výsledky.

Nakoniec som sa rozhodol pre tretiu konfiguráciu, čiže použitia bodovej mutácie v 1.5 % prípadoch a v ostatných 1.5 % situáciách použiť obyčajnú mutáciu. Nie že by jej výsledky vynikali nad všetkými ostatnými konfiguráciami, ale vo väčšine situácií dávala stabilné

výsledky, ktoré nepodliehali anomáliám ako nižšia úspešnosť pri väčšom počte generácií. Výsledná konfigurácia aplikácie GP:

- Pravdepodobnosť rekombinácie potomstva: 90 %
- Pravdepodobnosť reprodukcie: 7 %
- Pravdepodobnosť bodovej mutácie: 1.5 %
- Pravdepodobnosť obyčajnej mutácie: 1.5 %
- Výber pořadovou selekciou
- Použitý elitizmus
- Ostatné nastavenia ako veľkosť populácie, maximálna hĺbka stromu, počet generácií, použité funkcie a terminály budú uvedené pred každým testom.

4.2.6 Zobrazovanie výsledkov

Na konci behu aplikácia GP, vypíše na štandardný výstup generáciu v ktorej evolúcia skončila, fitness hodnotu najlepšieho jedinca v celom behu evolúcie a nájdenú funkciu v infixovom tvare. Do súboru `gp_out.txt` budú zapísané hodnoty nájdenej funkcie pre všetky vstupy z trénovacej množiny. Prvé číslo v riadku je x, druhé hodnota funkcie pre x. Pokiaľ by v nájdenej funkcie malo dôjsť k matematicky nepovolenej operácii (delenie nulou a pod.) bude na štandardný výstup vypísaná chybová hláška.

4.3 Aplikácia CGP

Zdrojový kód programu je tak ako pri aplikácii GP rozdelený do dvoch samostatných súborov: `cgp.c` a `cgp.h`. V `cgp.c` sú uvedené definície štruktúr, makrá, všetky globálne premenné, a hlavičky funkcií. V `cgp.c` je kód programu.

V nasledujúcej časti je popísaný spôsob implementácie aplikácie CGP bod po bode od spôsobu implementácie chromozómu až po celkový algoritmus podobným spôsobom ako v predchádzajúcej časti opisujúcej aplikáciu GP.

4.3.1 Návrh implementácie chromozómu a populácie

Chromozóm je v aplikácii reprezentovaný lineárnym poľom celých čísel, ktoré reprezentujú napojenia vstupov každého bloku a jeho funkciu. Keďže aplikácia bude riešiť problémy symbolickej regresie, rozhodol som, že každý blok bude mať dva vstupy a jeden výstup. Takéto nastavenie by malo postačovať pre hocijakú matematickú operáciu. Pokiaľ by implementovaná operácia potrebovala len jeden operand, jednoducho by jeden vstup, aj keby bol pripojený na nejaký blok, nebol braný do úvahy. Toto podporuje tiež explicitnú redundanciu, čo by malo podporiť evolúciu.

Každý chromozóm je teda zostavený z trojíc čísel. Prvé dve sú čísla blokov, na ktoré sú napojené vstupy daného bloku. Tretie číslo reprezentuje operáciu ktorá sa vykoná nad týmito vstupmi. Chromozóm má len jeden výstup, pretože ako celok reprezentuje hľadanú matematickú funkciu. Číslo bloku ktorý predstavuje výstup chromozómu, je posledné číslo v poli.

Množstvo blokov v chromozóme je dané maticou, ktorá je jeho abstrakciou. Tým je myslené to, že chromozóm má tvar matice, hoci v skutočnosti je implementovaný lineárnym poľom a príslušnosť jednotlivých blokov do riadkov prípadne stĺpcov je prepočítavaná. Veľkosť tejto matice je nastaviteľná v hlavičkovom súbore `cgp.h`, zmenou makier hodnôt `ROWS`(počet riadkov matice) a `COLUMNS`(počet stĺpcov matice). Počet blokov chromozómu je potom určený `ROWS*COLUMNS`. Počet čísel v poli je `ROWS*COLUMNS*3 + 1`.

Podobne ako v aplikácii GP, ani v aplikácii CGP nie je informácia o fitness hodnote uložená priamo v chromozóme. Bolo by síce možné túto hodnotu vložiť do poľa implementujúceho chromozóm, avšak keďže fitness hodnota je reálne číslo(typ `double`) a čísla označujúce vstupy a funkcie v chromozóme sú celé (typ `integer`), bolo by plýtvanie pamäťou, ak by bol chromozóm implementovaný poľom reálnych čísel. Z tohoto dôvodu som vytvoril globálne pole reálnych čísel, v ktorom sú uložené fitness hodnoty jednotlivých chromozómov.

Kvôly tomu aby som pri prepájaní blokov v chromozóme nemusel pri každom bloku prepočítavať, na ktoré elementy môže byť vstup daného bloku pripojený, vytvoril som globálne pole, ktoré obsahuje všetky možnosti pripojenia pre ten – ktorý blok. Toto pole je vytvorené ešte pred začiatkom evolučného algoritmu a je používané tak ako pri vytváraní prvej generácie, tak aj pri operáciách mutácie chromozómov.

Populáciu tvorí pole ukazatelov na polia predstavujúce jednotlivé chromozómy. Veľkosť populácie sa upravuje hodnotou makra `POPULATION` v hlavičkovom súbore `cgp.h`. Veľkosť populácie zahŕňa nielen potomkov ale aj rodiča. Takýmto spôsob som sa vyhol nutnosti vytvoriť miesto pre rodiča mimo poľa s populáciou, čo šetrí pamäť a tiež uľahčuje manipuláciu s chromozómami. Aby bolo jasné, ktorý jedinec v populácii je rodič, rozhodol som, že vždy bude na prvej pozícii v poli. Hoci to znamená, že v prípade zmeny rodiča je nutné tento chromozóm premiestniť na prvú pozíciu. Na druhej strane chromozóm na prvej pozícii netreba vyhodnocovať keďže bol už vyhodnotený v predchádzajúcej generácii.

Voľba funkcií a terminálov

V aplikácii CGP som implementoval rovnaké funkcie a terminály ako v prípade aplikácie GP. Rozdiel však nastal v realizácii terminálov. Keďže v prípade CGP nemá chromozóm terminálne uzly, nemohol byť použitý rovnaký postup ako pri GP. Existujú dve možnosti ako v CGP implementovať použitie terminálov. Prvý je jednoducho vložiť terminál do chromozómu ako funkciu, ktorá nevyužíva vstupy bloku a vždy vracia konštantnú hodnotu. Tento prístup však znamená, že niektoré operácie mutácie by boli zbytočné, pretože zmena prepojenia vstupov takéhoto bloku by nemala žiadny efekt. Taktiež takýto spôsob implementácie terminálov by obmedzoval množstvo možného použitia blokov s tradičnými funkciami, ktoré pracujú nad svojimi vstupmi a mohol by viesť k nutnosti zväčšiť chromozóm a tým k väčšej náročnosti na pamäť.

Tieto dôvody ma viedli k tomu, aby som použitie terminálov realizoval tak, že budú pridané ako vstup chromozómu a bloky sa potom môžu pripojiť na tento vstup. Týmto spôsobom je zabezpečené maximálne využitie chromozómu pre bloky realizujúce implementované funkcie nad svojimi vstupmi.

Avšak aby bolo možné pripojiť terminál na akýkoľvek vstup bloku v chromozóme, je nutné nastaviť level – back parameter na hodnotu rovnú počtu stĺpcov matice, čo by niekedy mohlo byť na škodu. V mojom prípade to nie je problém, preto som zvolil takýto prístup. Toto je dôvod prečo nie je možné náhodne generovať terminály z určitého intervalu ako používal Koza[4]. Implementované terminály sú celé čísla z intervalu $< -5, 5 >$

Výpočet fitness hodnoty chromozómu

V algoritme CGP sa vyberá vždy jeden jedinec z populácie, ktorý má najvyššiu fitness hodnotu a tým pádom nie je treba použiť inú typ ako hrubá fitness hodnota. Chromozóm sa v aplikácii CGP ohodnocuje rovnakým spôsobom ako v GP a platia rovnaké pravidlá. V CGP je však možné hodnotu chromozómu počítať dvojakým spôsobom.

Jednoduchší, avšak na čas náročnejší spôsob je vypočítať postupne hodnoty všetkých blokov chromozómu a ako výsledok vziať hodnotu bloku predstavujúceho výstup. Takto by boli počítané aj bloky, ktoré nemusia byť zahrnuté vo výslednej realizácii riešenia, čo môže niekedy značne spomaliť výpočet. Najmä ak chromozóm pozostáva z veľkého počtu blokov.

Zvolil som si druhý spôsob, a to riešenie pomocou rekurzie. Začnem pri bloku predstavujúci výstup chromozómu a postupne rekurzívnymi operáciami prechádzam a počítam len tie bloky, ktoré patria do výsledného riešenia. Tento spôsob dokázal ohodnotenie chromozómu značne zrýchliť.

4.3.2 Realizácia algoritmu CGP

Algoritmus v aplikácii je implementovaný rovnakým postupom ako je uvedený ako v sekcii 3.3. Pred začatím evolúcie sa vygeneruje počiatočná populácia. Vyhodnotí sa fitness hodnota všetkých jedincov a začne sa evolúcia tým, že sa vyberie rodič pre prvú populáciu potomkov. Tento rodič – chromozóm sa skopíruje na prvú pozíciu v populácii. Potom sa mutáciou vytvoria nový potomkovia, ktorý spolu s rodičom vytvoria novú generáciu. Fitness hodnota sa počíta pre všetky chromozómy len v počiatočnej populácii, pretože na prvom mieste je uložený rodič, ktorého fitness hodnota je už známa. Evolúcia je ukončená v momente kedy sú splnené ukončovacie kritériá.

Výber rodiča

Pri výbere rodiča v CGP je veľmi dôležité zachovať neutrálnosť výberu. To znamená, že pokiaľ je fitness hodnota rodiča a niektorého z jeho potomkov rovnaká , je treba uprednostniť tohto potomka ako rodiča ďalšej generácie. Táto situácia sa nazýva tzv. „neutrálna prechádzka“ v priestore riešení problému a významne zefektívňuje evolúciu v CGP.

Problémom však je, že fitness hodnota v aplikácii je prirodzene typu `double` a je veľmi málo pravdepodobné že dve fitness hodnoty tohto typu by boli úplne presne rovnaké. To znamená, že ak by sa fitness hodnota rodiča a potomka líšila na desiatom desatinnom mieste, hoci tento rozdiel je zanedbateľný, stále by nebola fitness hodnota potomka rovná fitness hodnote rodiča a nemohol by byť tento potomok zvolený za rodiča nasledujúcej generácie.

Tento problém som vyriešil zavedením tzv. „limitu“ pri porovnávaní fitness hodnôt rodiča a jeho potomkov. Tým je myslené to, že fitness hodnota potomka je považovaná za rovnú s fitness hodnotou rodiča pokiaľ je rovná alebo nižšia do určitej úrovne od veľkosti fitness hodnoty rodiča. Tento limit je rovný jednému percentu hodnoty fitness. Veľkosť sa dá zmeniť v makre `LIM` v hlavičkovom súbore `cgp.h`. V skratke to znamená, že potomka je možné vybrať za rodiča ak `fitnessPotomka >= fitnessRodica - fitnessRodica*LIM` (LIM je implicitne rovné 0.01).

Výber rodiča v aplikácii CGP prebieha tak, že sa najprv zistí najlepšia fitness hodnota v populácii potomkov. Ak je táto hodnota dostatočne veľká(viď odsek vyššie), sú do množiny kandidátov na rodiča vybraný všetci potomkovia, ktorý sú ohodnotený najlepšou fitness hodnotou alebo fitness hodnotou nižšou od najlepšej do určitého limitu. V tomto prípade je

limit nastavený na jednu tisícinu hodnoty makra `LIM`. Nakoniec je ako rodič ďalšej generácie zvolený náhodne vybratý jedinec z množiny kandidátov.

Realizácia mutácie chromozómu

Mutácia v CGP je jediná operácia, ktorá mení gény v chromozómoch. Vytvára sa ňou z jedného vybraného jedinca nová populácia, ktorá predstavuje nové riešenia skúmaného problému. V aplikácii CGP je realizovaná rovnako ako je uvedené v kapitole 3.3.2. Náhodne vyberie a zmení hodnotu v určitom počte génov v chromozóme. Tento počet je náhodne vybraný z intervalu od 1 do hodnoty premennej `maxMutGens`, ktorá udáva maximálne koľko génov je pri je pri jednej operácii mutácie možné zmeniť v rodičovi aby sa vytvoril jeden potomok. Hodnota tejto premennej je rovná nejakému percentu z počtu všetkých génov v chromozóme (dĺžky poľa implementujúceho chromozóm).

Avšak aké veľké by toto percento malo byť. Aby som zistil koľko percent génov je vhodné mutovať v mojej aplikácii CGP, rozhodol som sa otestovať ju pre niekoľko rôznych počtov mutovaných génov. Spoločné nastavenia aplikácie:

- počet stĺpcov matice chromozómu: 255
- počet riadkov matice chromozómu: 1
- veľkosť populácie: 5
- použité matematické operácie: $+, -, *, /$
- použité terminály: celé čisla z intervalu $< -5, 5 >$

Postupne som potom 50 – krát pustil aplikáciu s maximálnym povoleným počtom mutovaných génov rovnajúcemu sa 10 %, 15 %, 20 % a 30 % z celkového počtu génov pre 20000, 40000, 60000 generácií. Znova ako v pri testoch GP som porovnával počet úspešných behov. Úspešnosť sa znova odvýjala od súčtu absolútnych hodnôt odchyliek nájdeného riešenia od trénovacej množiny. V trénovacej množine bolo 20 dvojíc x,y. pre každú z nasledujúcich funkcií, ktorých predpisy som sa snažil nájsť.

$$f_4(y) = x^3 + 4x^2 - 7x + 19 \tag{4.6}$$

$$f_5(y) = \frac{5x^4 - x^2 + 2x}{x^2 + 7x + 5} \tag{4.7}$$

$$f_6(y) = 2x^4 - 10x^2 + 2x \tag{4.8}$$

Výsledky pre funkciu 4.6. Beh bol úspešný ak odchylka bola menšia ako 50.

	Úspešnosť v %		
	20000 generácií	40000 generácií	60000 generácií
10 %	26 %	**58 %**	**58 %**
15 %	30 %	48 %	30 %
20 %	**32 %**	48 %	48 %
30 %	26 %	46 %	54 %

Tabulka 4.12: Úspešnosť nájdenia riešenia aproximácie funkcie 4.6 v závislosti od množstva mutovaných génov pre vybraný počet generácií

Výsledky pre funkciu 4.7. Beh bol úspešný ak odchylka bola menšia ako 100.

	Úspešnosť v %		
	20000 generácií	40000 generácií	60000 generácií
10 %	**64 %**	**82 %**	**96 %**
15 %	56 %	74 %	82 %
20 %	46 %	76 %	82 %
30 %	44 %	52 %	84 %

Tabulka 4.13: Úspešnosť nájdenia riešenia aproximácie funkcie 4.7 v závislosti od množstva mutovaných génov pre vybraný počet generácií

Výsledky pre funkciu 4.8. Beh bol úspešný ak odchylka bola menšia ako 50.

	Úspešnosť v %		
	20000 generácií	40000 generácií	60000 generácií
10 %	18 %	**40 %**	**40 %**
15 %	20 %	30 %	**40 %**
20 %	**32 %**	34 %	36 %
30 %	24 %	**40 %**	34 %

Tabulka 4.14: Úspešnosť nájdenia riešenia aproximácie funkcie 4.8 v závislosti od množstva mutovaných génov pre vybraný počet generácií

Z výsledkov testov je zrejmé, že skôr nižšie percento mutovaných génov dáva lepšie výsledky. Preto som sa rozhodol ako maximálne povolený počet zmutovaných génov určiť 10 % z celkového počtu génov v chromozóme.

Kapitola 5

Porovnanie GP a CGP

V tejto časti budú porovnané oba prístupy GP a CGP na problémoch symbolickej regresie rôznej náročnosti. Keďže genetické programovanie a kartézske genetické programovanie pracujú so značne rôznymi počtami jedincov v populácii a chceme aby mali oba prístupy rovnaké podmienky, je nutné upraviť počet generácií ako ukončovacie kritérium. Z tohto dôvodu sa miesto počtu generácií bude počítať maximálny počet vyhodnotených chromozómov v evolúcii.

To znamená, že ak napríklad použijem v GP s populáciou 150 jedincov ukončovacie kritérium počet generácií 1000, potom v CGP s počtom jedincov v populácii rovnajúcemu sa 5 bude maximálny počet generácií 30000. Pretože 150*1000 = 5*30000. Všetky nižšie uvedené grafy sú vytvorené aplikáciou gnuplot na základe dát z výstupov aplikácií GP, CGP. Na zjednodušenie vzorcov z výstupov aplikácií GP a CGP je použitá wxMaxima.

Pokiaľ nebude nebude uvedené inak aplikácia GP bude mať populáciu o veľkosti 150 jedincov a CGP populáciu 5 jedincov. Maximálna povolená hĺbka stromu v GP bude 8, počet riadkov v matici CGP bude 1, počet stĺpcov 255. Toto dáva v aplikáciách GP a CGP chromozómom približne rovnaký počet stavebných blokov. Použité funkcie $+, -, *, /$, terminály celé čísla z intervalu $< -5, 5 >$.

Nasledujú dve sekcie experimentov. V prvej sa obe aplikácie snažia nájsť predpisy funkcií klasickej symbolickej regresie, v druhej zas predpisy iteračných postupov.

Aplikácie boli postupne spustené pre maximálny počet 150000, 300000 a 450000 ohodnotení chromozómov, čo je v prípade GP 1000, 2000 a 3000 generácií, v prípade cgp 30000, 60000, 90000 generácií.

Pre každý stanovený počet ohodnotených chromozómov bolo pre aplikáciu GP a CGP vykonaných vždy 50 behov, z ktorých výsledky sa vyhodnocovali podľa kritérií uvedených pri každom experimente. V tabuľkách stĺpec „úspešné“ znamená koľko percent z celkového počtu 50-tich behov programov bolo úspešných, čiže splnilo kritérium úspešnosti. Stĺpec „presné“ udáva koľko percent behov, skončilo nie len úspešne, ale aj nájdením presného predpisu hľadanej funkcie.

Všetky behy, ktorých výsledkom bola funkcia s matematicky neprijateľnou operáciou (delenie nulou) boli automaticky považované za neúspešné.

5.1 Klasická symbolická regresia

5.1.1 Experiment 1

Aplikácie GP a CGP hľadali predpis funkcie:

$$f_1 = x^3 + 2x^2 + 3x + 1 \tag{5.1}$$

Počet hodnôt v trénovacej množine bol 50 dvojíc x,y. Beh bol úspešný ak absolútna hodnota odchyliek riešenia od trénovacej množiny bola menšia ako 1.

	maximálny počet ohodnotení chromozómu					
	150000		300000		450000	
	úspešné	presné	úspešné	presné	úspešné	presné
GP	36 %	34 %	34 %	34 %	40 %	30 %
CGP	40 %	38 %	50 %	46 %	80 %	80 %

Tabulka 5.1: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad nájdenej funkcie pomocou GP, ktorá spĺňa podmienku úspešnosti, avšak nie je presne hľadaná funkcia:
(((-1)+((-5)-(((x+(x+(-3)))+(1-(1-1)))/((((-4)-x)/((-3)-x))*((2/(-1))/((-4)-x))))))*(((((-5)/(5*(-4)))*((2/(-1))/((-3)-x)))*((((-5)-x)/((-1)-x))*((2/(-1))/((-3)-x))))+x)*(-1)))
Po zjednodušení:

$$f(y) = x^3 + 2x^2 + 3x + 1 + \frac{6 - 2x}{x^3 + 7x^2 + 15x + 9} \tag{5.2}$$

5.1.2 Experiment 2

Aplikácie GP a CGP hľadali predpis funkcie:

$$f_2 = 4x^5 - 3x^4 - 20x + 197 \tag{5.3}$$

Počet hodnôt v trénovacej množine bol 50 dvojíc x,y. V tomto prípade bol beh úspešný ak súčet absolútnych odchyliek nájdeného riešenia od hľadanej funkcie bol menší ako 1000. Môže sa to zdať veľa avšak po prepočítaní na jednu hodnotu to robí odchylku veľkosti 20. Čo je rozdiel o 32 % od najmenšej hodnoty v trénovacej množine, ale už len 0.0000497 % od najväčšej.

	maximálny počet ohodnotení chromozómu					
	150000		300000		450000	
	úspešné	presné	úspešné	presné	úspešné	presné
GP	8 %	2 %	10 %	2 %	14 %	6 %
CGP	8 %	4 %	26 %	14 %	28 %	14 %

Tabulka 5.2: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad nájdenej funkcie pomocou CGP, ktorá spĺňa podmienku úspešnosti:
(((((((-5)+((x*(-1))*(x*2)))/(-4))+((2*3)+((-1)+(((-5)-((-5)+((x*(-1))*(x*2))))+4))))-(x-
(((((-5)+(x*2))/(((2+(-2))-2)-1))+((((2+(-2))-2)*(2*3))*(((0/2)-1)+(-4))))-(((-5)-((-5)+(
(x*(-1))*(x*2))))*(((-1)+(((-5)-((-5)+((x*(-1))*(x*2))))+4))/(-4))))))-((((-5)+((2*3)*x))
-x)+(((5+(((-5)-((-5)+((x*(-1))*(x*2))))+(((-5)-((-5)+((x*(-1))*(x*2))))*(x*(-1)))))+(x/(
-3)))*((-5)-((-5)+((x*(-1))*(x*2)))))))-((((((2+(-2))-2)-1)-(2+(-2)))-(x/(-3)))*(((((-5)-
((-5)+((x*(-1))*(x*2))))+4)-(((2+(-2))-2)*(2*3)))-((((-4)/((-5)*4))+(-5))/((-4)/
((-5)*4))))))
Po zjednodušení:

$$f(y) = \frac{48x^5 - 36x^4 - 240 + 2363}{12} \tag{5.4}$$

5.1.3 Experiment 3

Aplikácie GP a CGP hľadali predpis funkcie:

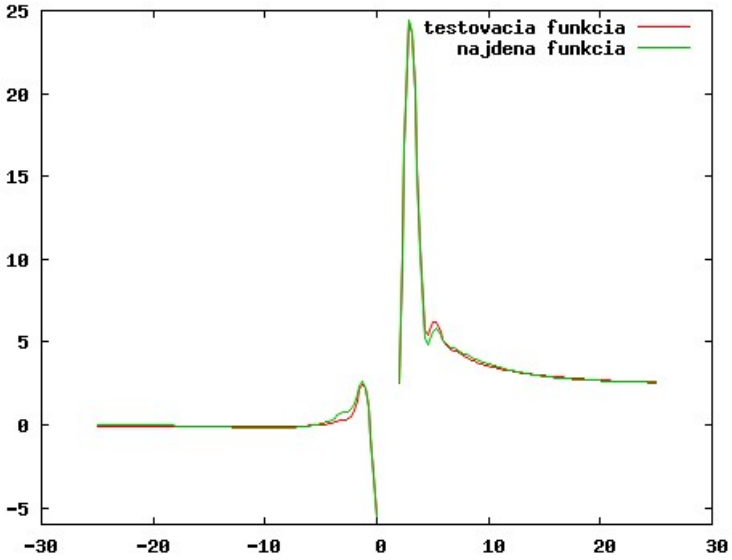
$$f_3 = \frac{2x^3 - 7x^2 - 19x - 11}{x^4 - 3x^3 + 2} \tag{5.5}$$

Počet hodnôt v trénovacej množine bol 50 dvojíc x,y. Beh bol úspešný ak súčet absolútnych odchyliek nájdeného riešenia od hľadanej funkcie bol menší ako 25. Keďže nebolo nájdené ani jedno presné riešenie stĺpec „presné“ je z tabuľky vynechaný.

	maximálny počet ohodnotení chromozómu		
	150000	300000	450000
	úspešné	úspešné	úspešné
GP	66 %	78 %	78 %
CGP	22 %	38 %	66 %

Tabulka 5.3: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad najlepšieho nájdeného riešenia pomocou GP, s odchylkou 4.73 od trénovacích dát:
(((((((-4)*((-3)+(-2)))*(3+(2+(-4))))-(((-4)+(-4))/((2*x)+((-3)+(-2)))))/((((2*3)/(3+(-2)
))/((2*x)+((-3)+(-2))))+(((-3)+(-2))+(-3))))/(((((3+(-2))-((-3)+x))/((-3)+(-2)))*((((-5)/
(-1))/(-4))/2))-((4/((2*x)+((-5)/(-1))))*2)))+(((((2*(2*x))/((2*x)+(-2)))/(((-3)+x)+(((-5
)/(-1))/(-4))))+(((((-3)+(-2))-((-3)+x))/((2*x)+(-3)))+((-4)+(((-3)+(-2))+(2+(-4))))))/
((((((-3)+(-2))+((-3)+(-4)))+(((-3)+(-4))+(2*3)))/((2*(2*x))+(((-3)+x)-(2+(-4)))))+((((
*3+(-3))-((-3)+x))/(2+((-3)+(-4))))+((-4)+(((-3)+(-2))+(2+(-4))))))))
Po zjednodušení:
$f(y) = -(24000x^7 + 411240x^6 - 2358898x^5 - 8278157x^4 + 5.7065931 * 10^{+7}x^3 - 9.0829203 * 10^{+7}x^2 + 4.5871627 * 10^{+7}x - 2009740)/(640x^8 - 44368x^7 + 456988x^6 - 2713912x^5 + 1.1619251 * 10^{+7}x^4 - 2.9154128 * 10^{+7}x^3 + 3.437846 * 10^{+7}x^2 - 1.4181647 * 10^{+7}x - 361284)$



Obrázek 5.1: Priebehy hľadanej funkcie a najlepšieho nájdeného riešenia

5.1.4 Experiment 4

Aplikácie GP a CGP hľadali predpis funkcie:

$$f_4 = \sin(x) \tag{5.6}$$

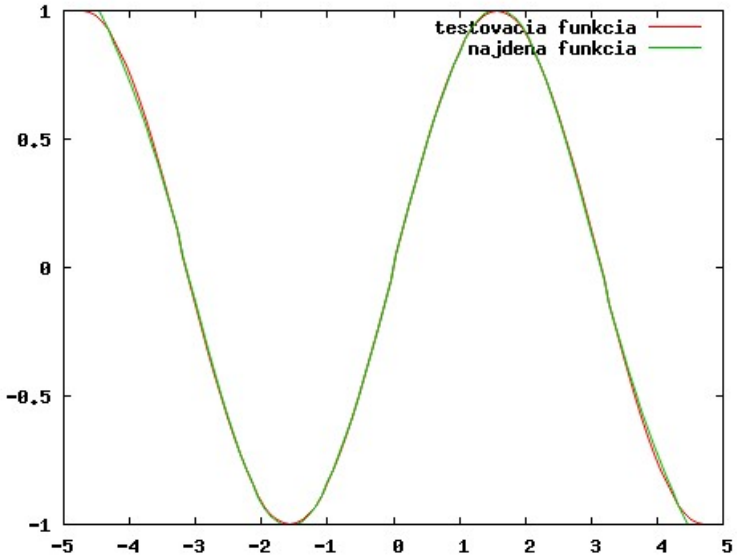
Beh bol úspešný ak súčet absolútnych odchyliek nájdeného riešenia od hľadanej funkcie bol menší ako 10. Množstvo dvojíc x,y v trénovacej množine bolo zvýšených na 101. Keďže nebolo nájdené ani jedno presné riešenie stĺpec „presné“ je z tabuľky znova vynechaný.

	maximálny počet ohodnotení chromozómu		
	150000	300000	450000
	úspešné	úspešné	úspešné
GP	70 %	86 %	80 %
CGP	26 %	44 %	74 %

Tabulka 5.4: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad nájdenej najlepšej nájdenej funkcie pomocou GP, s odchylkou veľkosti 1.12 od trénovacích dát:
(((((x/((-5)/(-4)))/(((-4)-((-5)/(-4)))/(-4)))+((x/(((-1)*2)/(-4)))-(((x/(-1))/(-3))*
((x*x)-(2+0)))))/((((1-((-1)*(2+0)))*(-1))+(((((-1)*2)/(-4))*(((-1)*2)/(-4)))-((-1)*2))))
/((((((-3)-(x*x))*((x*x)/(-4)))/(((-1)*((-5)/(-4)))*(2-(-4))))-(((((-1)*2)/(-3))-((2+0)*
((-1)*(-5))))/((-5)*((-3)- (x*x)))))+(-5)))
Po zjednodušení:

$$f(y) = -\frac{280x^5 - 1912x^3 - 8256x}{21x^6 + 126x^4 + 3339x^2 + 8274} \tag{5.7}$$



Obrázek 5.2: Priebehy hľadanej funkcie a najlepšieho nájdeného riešenia 5.7

5.2 Hľadanie iteračných predpisov

5.2.1 Experiment 1

Aplikácie GP a CGP hľadali postup newtonovej iteračnej metódy pre riešenie delenia použitím operácií $+, -, *$. Vzorec tohto algoritmu:

$$x_{i+1} = x_i(2 - bx_i) \tag{5.8}$$

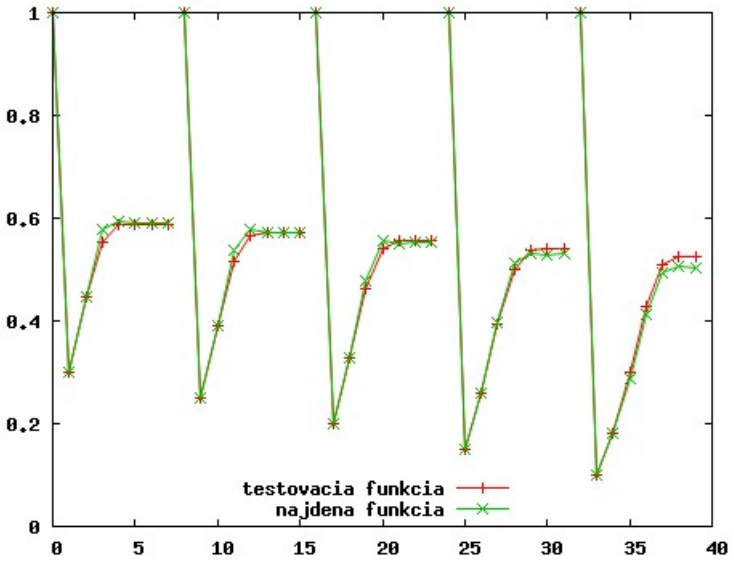
Použité operácie v tomto teste boli len $+, -, *$. Trénovaciu množinu tvorilo 136 hodnôt x_i, vrátane počiatočne zvolenej hodnoty x_0, ktorej hodnota bola vždy 1. Pre každú hodnotu premennej b sa vypočítalo 7 iterácií. Veľkosť b sa postupne pohybovala od 1.1 s prírastkom 0.05 až po hodnotu 1.9. Beh bol úspešný ak odchylka nájdeného riešenia bola menšia ako 5.

	maximálny počet ohodnotení chromozómu					
	150000		300000		450000	
	úspešné	presné	úspešné	presné	úspešné	presné
GP	42 %	18 %	40 %	18 %	60 %	34 %
CGP	28 %	14 %	68 %	48 %	66 %	48 %

Tabulka 5.5: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad riešenia získaného pomocou CGP, spĺňajúceho kritérium úspešnosti:
((((1-(1*(-1)))-b)-(((1-b)*((-3)+((3-x)-(1*(-1)))))*((1-((1-b)-x))-b)))-(((((1-(1*(-1)))-((3-x)-(1*(-1))))*((-3)+((3-x)-(1*(-1)))))*((1-((3-x)-(1-b)))-(-3)))*((((-3)+((3-x)-(1*(-1))))*1)*((1-((3-x)-(1*(-1))))-(((1-b)-x)-b)))))
Po zjednodušení:

$$x_{i+1} = -2x_i^5 + 8x_i^4 + (2b^2 - 8b - 2)x_i^3 + (-8b^2 + 31b - 27)x_i^2 + (10b^2 - 39b + 39)x_i - 4b^2 + 15b - 14 \tag{5.9}$$



Obrázek 5.3: Časť priebehov hľadaného iteračného postupu a riešenia 5.9

5.2.2 Experiment 2

Aplikácie GP a CGP hľadali iteračný algoritmus na výpočet odmocniny tzv. babylonský algrotimus:

$$x_{i+1} = \frac{1}{2}(x_i + \frac{S}{x_i}) \tag{5.10}$$

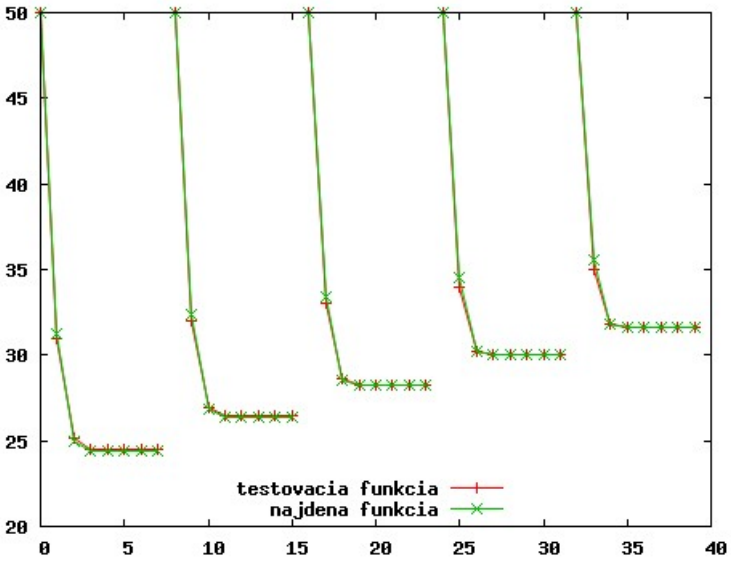
S je číslo. ktorého odmocnina sa počíta.
V trénovacej množine bolo 80 hodnôt x_i, vrátane počiatočnej hodnoty, vždy nastavenej na 50. Potom bolo vykonaných 7 iterácií pre S rovné postupne 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000. Beh bol považovaný za úspešný ak odchylka nájdeného riešenia bola menšia ako 10.

	maximálny počet ohodnotení chromozómu					
	150000		300000		450000	
	úspešné	presné	úspešné	presné	úspešné	presné
GP	20 %	2 %	18v	8 %	34 %	20 %
CGP	36 %	20 %	40 %	32 %	54 %	40 %

Tabulka 5.6: Výsledky porovnania GP a CGP pre zadaný maximálny počet ohodnotených chromozómov

Príklad riešenia získaného pomocou CGP spĺňajúceho kritérium úspešnosti:
(3+(((-2)+((((x+x)-(5*3))+((b*5)/3))/((2*3)+((x+x)+x))))+(((x/(1/((-3)-(-5))))+((2*3)/1))/(((((-1)-(0+x))/((5*4)+(5*3)))-(1+((-1)-(2*3)))))))
Po zjednodušení:

$$x_{i+1} = -\frac{615x_i^2 + (6312 - 5S)x_i + 1045S - 1863}{9x_i^2 - 1863x_i - 3762} \tag{5.11}$$



Obrázek 5.4: Časť priebehov hľadaného iteračného postupu a riešenia 5.11

5.3 Vyhodnotenie experimentov

Najprv k všeobecnej úspešnosti oboch prístupov GP aj CGP. Po vyhodnotení vyššie uvedených experimentálnych úloh som prišiel na na to, že obe mnou vytvorené aplikácie majú problém s aproximáciou dát s vysokými číselnými hodnotami. Toto bol prípad najmä implementácie GP. Mojím názorom je, že za to môžu nepresnosti ako zaokrúhlenie a orezanie pri operáciách s premennými typu `double`, do ktorých sa ukladali výsledky matematických operácií. Veľké hodnoty v trénovacích dátach implikujú rýchle stúpanie hľadanej funkcie, čo podporuje kopenie operácií ako násobenie, prípadne sčítanie v niektorých vetvách chromozómu. Nesprávne číselné vyhodnotenie týchto častí chromozómu potom môže viesť k celkovej degradácii tohto jedinca.

Druhý poznatok bol, že pokiaľ hľadaná funkcia obsahovala maximum, prípadne lokálne maximum, ktorého číselná hodnota bola o poznanie vyššia ako ostatné hodnoty funkcie, obe aplikácie mali tendenciu vytvárať riešenia, ktoré aproximovali tieto miesta a odchylka od ostatných hodnôt bola vyššia. To isté platí aj v prípade, kedy hľadaná funkcia v určitých miestach nadobúdala malé číselné hodnoty a v iných hodnoty ďaleko väčšie. V týchto situáciach sa stávalo, že odchylky od najväčších hodnôt sa pohybovali v stotinách a od odchylky od najmenších hodnôt v desiatkach alebo stovkách.

Domnievam sa, dôvod bol taký, že vo väčšine prípadov náhodne vytvorené funkcie nerástli tak prudko ako hľadaná. Čo znamená, že sa tvorili 2 druhy funkcií. Buď aproximovali nižšie hodnoty, čo vedie k odchylkám v priebehu funkcie vo väčších číselných hodnotách. Druhý typ zas aproximoval väčšie číselné hodnoty avšak s veľkými odchylkami s hľadanou funkciou v jej časti priebehu s nižšími číselnými hodnotami. Pokiaľ rozdiel medzi číselnými hodnotami hľadanej funkcie bol dostatočne veľký, tak sa fenotyp jedincov celej populácie obmedzil na aproximáciu priebehu funkcie v týchto veľkých číselných hodnotách a až keď väčšina jedincov v populácii riešení mala podobnú fitness hodnotu, začali nový jedinci viac a viac približovať zbytok priebehu funckie. Napríklad ak je v trénovacej množine 20 hodnôt z čoho prvá polovica sa pohybuje v rozmedzí 1 až 20 a zbytok v rozmedzí 100 až napríklad niekoľko miliónov, nájdené riešenia budú s najväčšou pravdepodobnosťou mať nízke

odchylky od druhej polovice a naopak vysoké od prvej polovice trénovacích dát.

S týmito problémami ako je vidieť z výsledkov testov si lepšie vedela poradiť aplikácie CGP. Na druhej strane výsledky aplikácie GP sú lepšie v aproximácii funkcií s nižšími hodnotami.

Podľa môjho názoru, implementácia GP má tiež nevýhodu pri aproximovaní funkcií s kratšími predpismi (testy 1,2,5,6), kvôly spôsobu reprezentácie chromozómu koreňovým stromom. V priebehu evolúcie majú totiž stromy tendenciu rásť, zvyčajne na konci dosahuje väčšina maximálne povolenú hĺbku a ku skracovaniu stromov dochádza relatívne málo. Tým pádom je nutné aby sa vytvorila funkčná neutralita v niektorých vetvách stromu, aby neboli ovplyvnené vetvy s vhodným riešením. V CGP je dĺžka chromozómu variabilnejšia, čo je podľa môjho názoru veľkou výhodou. Ako vidieť v testoch 3 a 4, kde je náročné nájsť presné riešenie, sa táto výhoda stráca a GP dosahuje väčšiu úspešnosť.

Kapitola 6

Závěr

Písaním tejto práce som nadobudol mnoho teoretických vedomostí a praktických skúseností z oblasti evolučných algoritmov, najmä z genetického a kartézskeho genetického programovania. Zoznámil som sa s problémami, ktoré musí riešiť každá aplikácia implementujúca tieto postupy. Pokúsil som sa tiež vytvoriť čo najefektívnejšie aplikácie riešiace problémy symbolickej regresie použitím genetického a kartézskeho genetického programovania a následne ich podrobil testom.

Mojou úlohou bolo vytvoriť dve čo najefektívnejšie aplikácie GP a CGP slúžiace k riešeniu úloh symbolickej regresie. Ďalej som mal podrobiť oba programy testom rôznej náročnosti, porovnať oba spôsoby a ukázať ich výhody poprípade nevýhody.

Toto zadanie som podľa môjho názoru splnil. Podarilo sa mi vytvoriť obe aplikácie a na základe testov zvoliť také nastavenia a postupy, aby oba programy pracovali čo najefektívnejšie. Otestoval som ich na rôznych úlohách a poukázal na výhody a nevýhody GP a CGP. Mrzí ma však, že som nenašiel výsledky iných aplikácií GP a CGP riešiacich podobné problémy, aby som ich mohol porovnať s výsledkami mnou vytvorených programov.

V tejto časti informatiky je ešte veľa nepreskúmaných možností a vývoj bude viesť k objavovaniu nových a lepších genetických algoritmov, ktoré budú schopné riešiť o mnoho zložitejšie problémy, v kratšom čase. Ďalším krokom v pokračovaní tejto práce by napríklad mohlo byť nájdenie riešení problémov, na ktoré som v nej poukázal, a tak vylepšiť efektivitu GP a CGP.

Literatura

[1] Balčák, T.; Butka, P.: Prednášky z predmetu EVOLUČNÉ ALGORITMY. `http://neuron.tuke.sk/~machm/studenti/ea/lectures1/html-nochunks/main.html`, 2002 [cit. 2009-4-27].

[2] Banzhaf, W.: *Genetic programming: an introduction on the automatic evolution of computer programs and its applications*. Morgan Kaufmann, 1998, iSBN 155860510X.

[3] Clegg, J.; Walker, J. A.; Miller, J. F.: A new crossover technique for Cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, 2007, ISBN 978-1-59593-697-4, s. 1580–1587.

[4] Koza, J. R.: *Genetic Programming: on the programming of computers by means of natural selection*. The MIT Press, 1992, iSBN 0-262-11170-5.

[5] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. Vydavateľstvo STU, 2000, iSBN 80-227-1377-5.

[6] Lessman, R. E.: Changes and extensions in the C family of languages. *SIGCSE Bull.*, ročník 21, 1989: s. 34–39, ISSN 0097-8418.

[7] Miller, J. F.; Thomson, P.: Cartesian Genetic Programming. In *Genetic Programming, Proceedings of EuroGP'2000*, Springer-Verlag, 2000, ISBN 3-540-67339-3, s. 121–132.

[8] Pohlheim, H.: Evolutionary Algorithms 3 Selection. `http://www.geatbx.com/docu/algindex-02.html`, 2006 [cit. 2009-5-5].

[9] Poli, R.; Langdon, W. B.: Genetic Programming with One-Point Crossover and Point Mutation. In *Soft Computing in Engineering Design and Manufacturing*, Springer-Verlag, 1997, s. 180–189.

[10] Vašíček, Z.; Sekanina, L.: Evoluční návrh kombinačních obvodů. *Elektrorevue*, ročník 43, 2004.

[11] Walker, J.; Miller, J.: The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming. *Evolutionary Computation, IEEE Transactions on*, ročník 12, č. 4, Aug. 2008: s. 397–417, ISSN 1089-778X.

[12] Walker, J. A.; Miller, J. F.: Solving real-valued optimisation problems using cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, 2007, s. 1724 – 1730.

[13] Yu, T.; Miller, J.: Neutrality and the Evolvability of Boolean Function Landscape. In *Genetic Programming, Proceedings of EuroGP 2001, volume 2038 of LNCS*, Springer-Verlag, 2001, s. 204–217.

[14] Zelinka, I.; Oplatková, Z.; Šeda, M.; aj.: *Evoluční výpočetní techniky – principy a aplikace*. BEN – technická literatura, 2009, iSBN 970-80-7300-218-3.

Dodatek A

Manual

Na prilozenom CD su v zlozke `bakalarska_praca` ulozene subory: `gp.c`, `gp.h`, `cgp.c`, `cgp.h`, `Makefile`. Po spusteni prikazu `make` v prikazovom riadku budu vytvorene programy: `gp,gp_iter,cgp,cgp_iter`. Programy `gp a gp_iter` implementuju geneticke programovanie. Programy `cgp a cgp_iter` implementuju kartezske geneticke programovanie. Programy sa v prikazovom riadku spustaju `./gp trenovacie_data`. Pokial bude program spusteny s inym poctom parametrov skonci s chybovou hlaskou, pokial nastane chyba pri citani suboru znova skonci s chybovou hlaskou.

Pre programy `gp cgp` obsahom suboru `trenovacie_data` musia byt dvojice cisel oddelene tabulatorom, pripadne medzerou. Prve cislo reprezentuje x druhe prislusnu hodnotu y pre hladanu funkciu. Na kazdom riadku moze byt len jedna dvojica.

Pre programy `gp_iter` a `cgp_iter` obsahuje subor `trenovacie_data` len jedno cislo na jednom riadku. Tieto cisla musia tvorit nasledujucu sekvenciu: b,x_1,x_2,x_3,x_4, ...,x_n. x_0 je pociatocna zvolena hodnota, b je napriklad pre hladanie newtonovho iteracneho algoritmu vo vzorci $x_{i+1} = x_i(2 - bx_i)$, a tiez vo vzorci babylonskeho algoritmu $x_{i+1} = \frac{1}{2}(x_i + \frac{S}{/}x_i)$ sa bckom mysli hodnota S. Taketo sekvencie sa mozu potom v tomto subore opakovat s inou hodnotou b,alebo x_0.

Je dolezite v pripade `gp cgp` hodnota v ich hlavickovych suborov vo vyznacenych miestach v makre NUMOFVAL zodpovedala poctu dvojic, ktore obsahuje subor s testovacimi datami (to znamena pocet riadkov v subore s testovacimi datami). A az potom pouzit prikaz make.

V pripade `gp_iter` a `cgp_iter` je naviac treba nastavit makro `REPSEQUENCE` na hodnotu rovnajucu sa poctu cisel v sekvencii v trenovacich datach. Napriklad ak v je subore sekvencia: b,x_1,x_2,x_3,x_4 hodnota v `REPSEQUENCE` bude 5. V trenovacich datach je potom mozne menit len hodnoty v sekvencii a nie pocet cisel.

Priklady trenovacich dat aj s popisom nastaveni su na CD v zlozke `trenovacie_data`.

Potom je mozne spustenie programov. Po skonceni behu program vypise na standardny vystup fitness hodnotu najdeneho riesenia a jeho infixovu formu. V pripade ak riesenie ma nulovu odchylku od trenovacich dat, bude miesto fitness hodnoty napisana hlaska: `Bolo najdene presne riesenie`. Pre programy `gp` a `cgp` budu v suboroch `gp_out.txt` a `cgp_out.txt` vypisane hodnoty najdeneho riesenia pre hodnoty x z trenovacej mnoziny v rovnakom formate ako pre trenovacie data. Pre programy `gp_iter` a `cgp_iter` budu v suboroch `gp_iter_out.txt` a `cgp_iter_out.txt` vypisane hodnoty pre hodnoty x z trenovacej mnoziny, avsak uz nebudu vypisane hodnoty b ale len $x_1, x_2, x_3, \ldots$. Pokial sa na standardnom vstupe objavy hlaska: `Matematicka funkcia osahuje nepovolenu operaciu`, znamena to, ze v rieseni sa napriklad nachadza operacia ako delenie nulou a pod.

Pokial je treba vypisovat na standardny vystup cislo aktualnej generacie je nutne odkomentovat urcene miesto v kode programu vo funkcii `life()`.

Vseobecne nastavenia:

- Typy pouzitych funkcii je nutne nastavit priamo v programe vo funkcii `selectOp()`.
- Druhy pouzitych terminalov je nutne nastavit priamo v programe vo funkcii `selectVal()`.
- Nastavenie mnozstva populacie je v hodnote makra `POPULATION`.
- Nastavenie poctu generacii je v hodnote makra `GENERATIONS`.
- Nastavenie poctu generacii trenovacich dat je v prislusom hlavickovom subore v hodnote makra `NUMOFVAL`.
- Nastavenie postacujucej fitness hodnoty je v prislusom hlavickovom subore v hodnote makra `FITNESS`. Ak dosiahne jedinec tejto fitness hodnoty evolucia sa konci. Zrusit je to mozne zakomentovanim prislusnej casti v kode vo funkcii `life()`
- Nastavenie o kolko sa ma znizit fitness hodnota jedinca obsahujuceho matematicky neprijatelnu operaciu (delenie nulou) je v prislusom hlavickovom subore v hodnote makra `LOWERFITNESS`.
- Nastavenie poctu cisel v sekvencii pre `cgp_iter` a `gp_iter` je v hodnote makra `REPSEQUENCE`

Nastavenia len pre GP:

- Hlbka stromu = hodnota makra `DEPTH`
- pravedepodobnost vlozenia terminalu namiesto funkcie pri vytvarani jedincov = hodnota makra `ENDBRANCH`
- pravedepodobnost vybratia uzlu reprezentujuceho funkciu pri genetickych operaciach = hodnota makra `FUNC`

Nastavenia len pre CGP:

- Pocet riadkov matice chromozomu = hodnota makra `ROWS`
- Pocet stlpcov matice chromozomu = hodnota makra `COLUMNS`
- Pocet vstupov chromozomu = hodnota makra `IN`
- Pocet vystupov chromozomu = hodnota makra `OUT` !nemenit!
- Tolerancia odchylky fitness hodnot = hodnota makra `LIM`