Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Genetické programování a jeho praktické využití

Bakalářská práce

Jaroslav Dibitanzl

Vedoucí práce: Ing. CSc. Jiří Jelínek

České Budějovice 2019

Dibitanzl, J., 2019: Genetické programování a jeho praktické využití. [Genetic programming and its practical use. Bc. Thesis, in Czech.] – 48 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Tato bakalářská práce se zabývá symbolickou regresí a jejím řešením pomocí genetického programování. Práce obsahuje teoretickou a praktickou část. V teoretické části je popsán princip genetického programování, praktická část obsahuje ukázky zpracování symbolické regrese pomocí knihoven Clojush, EllenGP, DEAP, FlexGP, KarooGP a popis vlastní implementace symbolické regrese.

Cílem práce je ukázat možnosti genetického programování a jeho využití při řešení symbolické regrese. Výsledkem práce jsou i ukázky jednotlivých výše zmíněných knihoven a analýza vlastní implementace symbolické regrese.

Abstract:

This bachelor's thesis deals with symbolic regression and its solution using genetic programming.

The thesis consists of theoretical a practical part. Theoretical part focuses on principle of genetic programming, practical part contains example solution of symbolic regression using libraries Clojush, EllenGP, DEAP, FlexGP, KarooGP and describes own solution. Goal of this thesis is to display possibilities of genetic programming and how it can be used for solving symbolic regression. Outcomes of thesis are examples of individual above-mentioned libraries and analyse own solution of symbolic regression.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Ve Českých Budějovicích dne 17. 4. 2019

Jaroslav Dibitanzl

Poděkování

Děkuji svému vedoucímu práce, Ing. CSc. Jiřímu Jelínkovi, za cenná doporučení, trpělivost a poskytnutí odborné literatury k tématu.

Obsah

1 Úvod

Oblast strojového učení může být inspirována přírodními principy evoluce, přestože využití těchto principů není nutné – oblast by mohla existovat na základě svých definovaných algoritmů, datových struktur a teorií učení i bez odkazování na organismy, kognitivní nebo genetické struktury a psychologické nebo evoluční teorie [1].

Evoluční algoritmy se snaží využít modelů evolučních procesů, aby nalezly řešení náročných nebo rozsáhlých úloh. Do oblasti evolučních algoritmů je zařazeno i genetické programování [2]. Za pomoci genetického programování je možné řešit zadané problémy vytvořením nejvhodnějšího programu. Programátor nemusí zadávat konkrétní postup, jen uvést problém a možné proměnné a funkce pro složení výsledného programu. Na základě analogie s evoluční biologií se populace programů vytvořená genetickým algoritmem postupně vylepšuje, dokud nedojde ke splnění ukončovací podmínky – buď nalezení optimálního řešení nebo dovršení maximálního počtu generací. Genetické programování vychází z Darwinovy teorie [3].

Tato práce si dává za cíl popsat hlavní principy genetického programování a aplikovat je při řešení symbolické regrese. V první části jsou vysvětlena základní pravidla genetického programování, jeho algoritmus a proces vzniku nové generace či více generací. Dále je uveden princip symbolické regrese a její potřebné vstupní parametry. Následuje výčet ukázek implementací symbolické regrese pomocí knihoven Clojush, EllenGP, DEAP, FlexGP, KarooGP a vlastní implementace symbolické regrese v jazyce JavaFX.

V závěru je vypracována analýza vlastní implementace symbolické regrese a shrnutí získaných poznatků.

2 Genetické programování

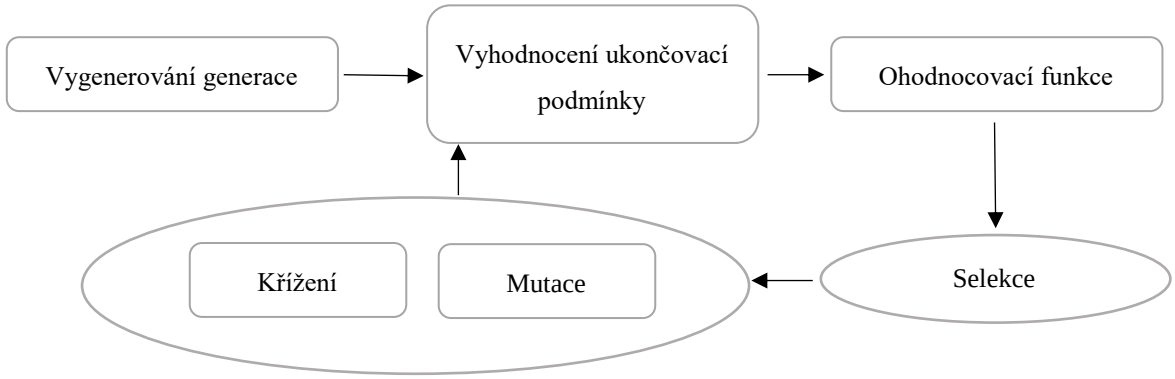
2.1 Vznik z evolučních algoritmů

Evoluční algoritmy začaly vznikat v 70. letech. Evoluční algoritmy pracují s množinou modelů evolučních procesů, pomocí kterých se snaží nalézt řešení různých úloh. Vzniklo tak několik různých přístupů; jedním z nich je právě genetický algoritmus, který získal svoji oblibu pro řešení problémů učení a adaptace 4 [3]. V knize „Genetic Programming: On the Programming of Computers by Means of Natural Selection“ J. Koza uvádí metodologii odvozenou z evolučního algoritmu nazvanou *genetické programování* [12].

2.2 Popis algoritmu

Genetické programování je technika, která automaticky řeší problém, aniž by uživatel musel znát a zadávat strukturu hledaného řešení [3].

Algoritmus probíhá tak, že se na začátku výpočtu vytvoří populace náhodných chromozomů. Tyto chromozomy reprezentují potenciální výsledky a skládají se z terminálů a funkcí. Chromozom je vždy ohodnocen fitness funkcí [3]. Nejlépe ohodnocené chromozomy jsou pak vybrány pro křížení a mutaci pro vytvoření další generace. Tento postup se opakuje, dokud není splněna ukončovací podmínka, která může být buď dovršení určené hodnoty fitness nebo vytvoření zadaného počtu generací. Výsledkem celého algoritmu je nejlépe ohodnocený chromozom z poslední vytvořené generace.



Obrázek 1: Znázornění genetického frameworku

2.3 Terminály a funkce

V genetickém programování není určena struktura programů (ty jsou vytvořeny náhodně nebo pomocí procesů genetického algoritmu), ale definují se prvky, ze kterých je výsledný program složen. Tyto prvky se rozdělují na *terminály* a *funkce*.

Do množiny terminálů patří proměnné či konstanty [5].

Do množiny funkcí patří operátory a funkce s argumenty. Množina funkcí pro symbolickou regresi obvykle obsahuje:

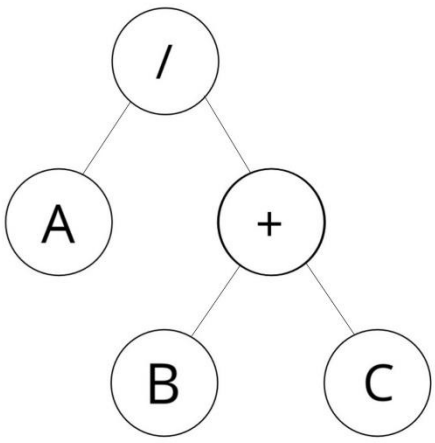
- aritmetické funkce (+, -, *, /),
- matematické funkce (*sin, cos, exp, abs, ...*).

Dále v některých jiných pracích se uvádí v případě GP další funkce [5]:

- booleovské logické funkce (*and, or, not*)
- podmíněné vyhodnocení (*if*).

2.4 Reprezentace programů

V genetickém programování se vygenerované programy častěji prezentují ve formě syntaktických stromů než řádků kódu. Proměnné, konstanty a funkce s nulovým argumentem (terminály) se stávají listy stromu, zatímco operátory a funkce s nenulovým argumentem se stávají vnitřními vrcholy grafu [6].



Obrázek 2: Ukázka programu (B + C) / A

2.5 Fitness funkce

Jelikož vytváření chromozomů v populaci probíhá náhodně, existuje pravděpodobnost vytvoření méně vhodného řešení. Proto je každému chromozomu v procesu genetického algoritmu určována fitness funkce, která charakterizuje úspěšnost chromozomu řešit zadaný problém. Existuje mnoho způsobů pro vyjádření fitness, řadí se mezi ně anglicky nazvané *raw fitness, standardized fitness, adjusted fitness, normalized fitness [3]*.

2.5.1. Raw fitness

Představuje účelovou funkci, úzce spjatou s daným problémem. Například u symbolické regrese by mohla prezentovat součet absolutních odchylek a aktuálního vypočteného řešení. Samotatná *raw fitness* se však běžně v genetickém programování nevyužívá. Mnohem běžnější je převádět *raw fitness* na *standardized fitness* [8].

2.5.2. Standardized fitness

Standardized fitness se používá jako alternativa za raw fitness. V práci je tento typ fitness užit, protože se hledá minimální chyba při porovnání očekáváné a výsledné hodnoty. Menší hodnota znamená lepší výsledek, je tedy ideální, aby hodnota fitness chromozomu představujícího optimální řešení byla rovna nule. Standardized fitness se vypočítává na základě toho, zda u konkrétního problému vyšší hodnota raw fitness představuje lepší řešení. V případě, že menší hodnota raw fitness je lepší řešení, pak se standardized fitness rovná raw fitness. Pokud je u konkrétního problému nižší hodnota raw fitness lepší řešení, počítá se pak standardized fitness jako rozdíl maximální hodnoty raw fitness a vypočtením aktuální raw fitness chromozomu [8].

V práci probíhá výpočet tak, že se vypočte pro chromozom jeho funkce dosazením hodnot z dat a porovnáním s očekávaným výsledkem dané dosazené řádky dat. Dosazení hodnot probíhá po vytvoření chromozomu. Výpočet funkce probíhá rekurzivně, počínaje kořenem stromu, který je vždy operátor, který má vždy aritu alespoň o hodnotě jedna. Výsledná fitness chromozomu se pak rovná průměru porovnání všech vypočtených a očekávaných výsledků z dat.

Výpočet fitness chromozomu probíhá dle:

$$F_{ch} = \frac{\sum_{j=1}^{n} |V_j - E_j|}{n},$$

kde F_{ch} je vypočítavaná fitness chromozomu, n je počet očekávaných výsledků, V je vypočtená funkce chromozomu po dosazení hodnot z řádku j a E je očekávaný výsledek řádku j.

2.5.3. Adjusted fitness

Adjusted fitness vychází ze standardized fitness a z jejího nedostatku neomezenosti maximální hodnoty. Hodnoty fitness se pohybují v intervalu od 0 do 1, kde větší hodnota značí lepší chromozom [8]. Adjusted fitness má výhodu většího zvýraznění menších rozdílů ve fitness než standardized fitness. V případě velmi vysokého rozmezí hodnot, které mohou nastat v případě standardized fitness, například $F = 0$ (nejlepší řešení) a $F = 260$ (nejhorší řešení), se adjusted fitness projeví méně výrazně, než když je rozmezí menší. Adjusted fitness je definována vzorcem:

$$F_{adjusted} = \frac{1}{1 + \frac{\sum_{j=1}^{n} |V_j - E_j|}{n}}$$

kde n je počet očekávaných výsledků, V je vypočtená funkce chromozomu po dosazení hodnot z řádku j a E je očekávaný výsledek řádku j.

2.5.4. Normalized fitness

Normalized fitness je další z možností určení fitness chromozomu. Na rozdíl od adjusted fitness platí, že součet všech normalized fitness chromozomů v jedné populaci je roven 1. Normalized fitness je definována vzorcem:

$$F_{normalized} = \frac{F_{standardized}}{\sum_{i=1}^{N} F_{standardized}}$$

2.6 Tvorba generace

Genetické programování opakovaně vytváří nové generace chromozomů na základě zavedených genetických procesů (nejčastěji jsou používány křížení a mutace). Tyto operace jsou aplikovány na chromozomy vybrané selekčním mechanismem. Tvorba nových generací je zásadní prvek evoluce, a tedy vývoje celého procesu vylepšování populace samotné [12].

2.7 Selekce chromozomů pro vytvoření další generace

Pro vytvoření dalších generací je potřeba v genetickém algoritmu vybrat chromozomy pro aplikaci genetických procesů. Selekční mechanismus vybere jeden chromozom (pro mutaci a reprodukci) a dva chromozomy (pro křížení). Mezi dva nejběžněji používané selekční mechanismy patří turnajová selekce a ruletová selekce.

2.7.1. Turnajová selekce

Turnajová selekce nabízí selekci pro tvorbu nové generace na základě náhodného výběru dvou i více chromozomů. Tato selekce se využívá pro svoji schopnost zajistit náhodnější výběr chromozomů, což zabraňuje celému procesu v předčasné konvergenci, popřípadě uváznutí v lokální extrému [9].

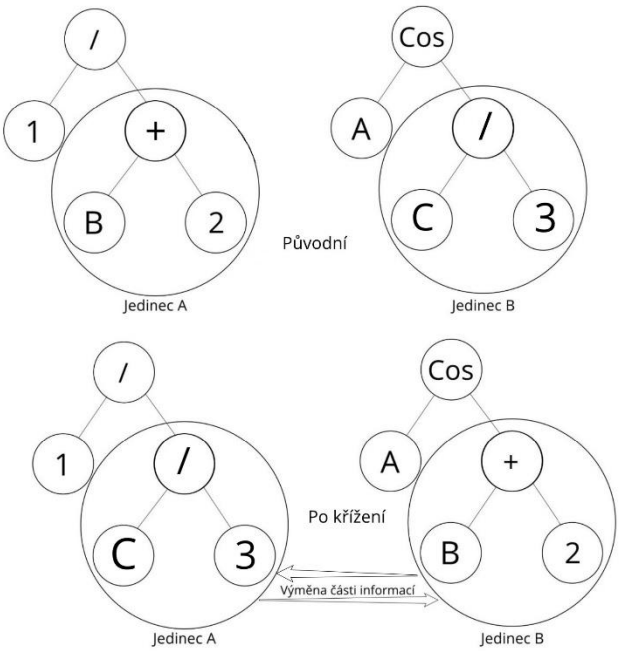
2.7.2. Ruletová selekce

Ruletová selekce rozvrhne celkovou sumu fitness funkcí chromozomů v prostoru, tedy každý chromozom získá na pomyslné ruletě tak velký úsek, který odpovídá velikosti hodnoty fitness. Tato selekce preferuje výběr silnějších chromozomů, a tedy konvertuje rychleji k řešení, ale algoritmus během výpočtu může uváznout v lokálním minimu [9].

2.8 Genetické operátory

2.8.1 Křížení

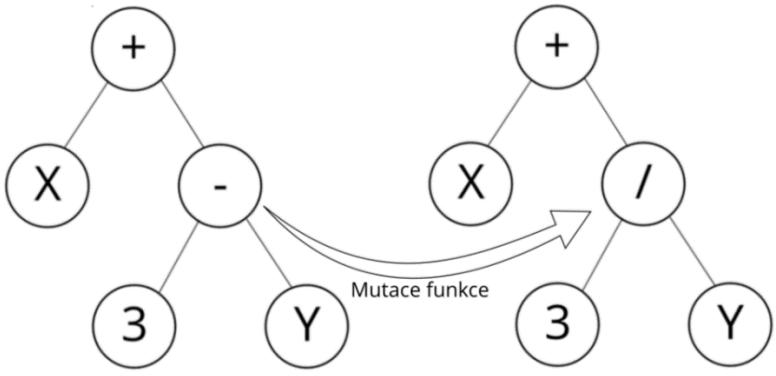
Křížení je genetický proces, při kterém jsou dva chromozomy, určené selekčním mechanismem, zkombinovány, tedy zanášejí do populace novou variantu chromozomu. V genetickém programování jsou chromozomy často prezentovány jako syntaktické stromy, proto dochází k náhodné výměně částí stromu [13].



Obrázek 3: Ukázka křížení mezi chromozomy A a B

2.8.2 Mutace

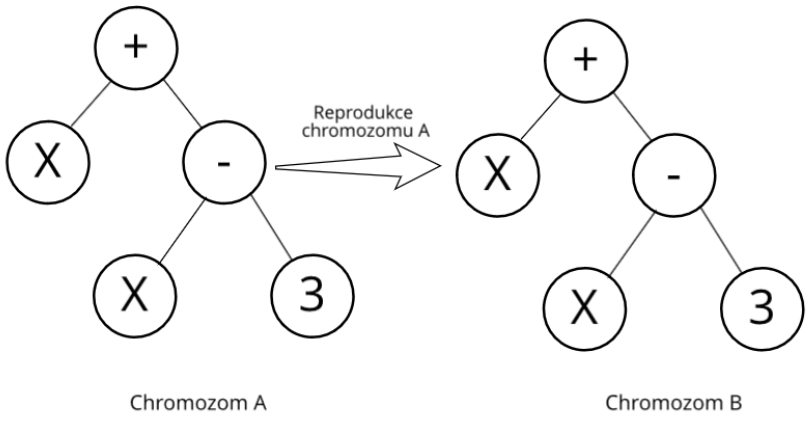
Mutace je genetická operace právě s jedním chromozomem. Mutace může způsobit vytvoření nového chromozomu, který lépe vyhovuje řešení, nebo také zhoršit stávající nejlepší chromozom. Může mutovat terminály či funkce [14]. V případě mutace funkce zůstávají zachovány terminály, to znamená, že nemůže být funkce nahrazena za jinou, která nemá stejnou aritu.



Obrázek 4: Příklad mutace funkce

2.8.3 Reprodukce

Reprodukce je genetický operátor, který vezme chromozom a zkopíruje celý obsah chromozomu do nového chromozomu, který je předán do nové populace. Reprodukce má obvykle nejmenší dopad na nalezení nového lepšího řešení, protože nepřináší žádnou novou změnu do následující generace.



Obrázek 5: Příklad reprodukce chromozomu

3 Symbolická regrese

Symbolická regrese je proces popsaní matematické závislosti v empirických datech. Dá se využít v případech, kdy není popisná funkce mezi daty zřejmá nebo dokonce vůbec známá [10]. Na začátku procesu není známá výsledná matematická závislost, pouze vybrané *funkce* a *terminály* a data vzniklá právě hledanou funkcí. Pro nalezení potřebné matematické funkce se využívá genetického programování, při kterém dochází k tvorbě generací na základě vybraných operátorů a terminálů. Schopnost symbolické regrese odhalit závislosti je hojně využívána ve všech oborech, kde je potřeba porozumět datům a jejich strukturám [15].

3.1 Řešení pomocí genetického programování

Pro vyřešení symbolické regrese bude nasazeno genetické programování, kde chromozom bude prezentovat potenciální hledanou matematickou funkci. Inicializační populace bude vytvořena na základě zadaných terminálů, povolených funkcí a empirických dat.

Výsledkem procesu bude chromozom s nejvyšší fitness, a tedy chromozom nejpravděpodobněji odpovídající skrytému vztahu mezi daty.

3.2 Množina terminálů

Množina terminálů představuje soubor čísel aplikovaných ve výpočtu programů. Může se jednat o konstanty nebo proměnné.

Terminály mohou být zahrnuty do výpočtu, a tedy se mohou objevit i ve výsledné funkci. Nicméně jejich podoba se může změnit s ohledem na postprocesing.

V práci jsou jako proměnné uvažovány všechny sloupce tabulky, kromě posledního sloupce očekávaných výsledků. Další terminály mohou vzniknout genetickými operacemi.

3.3 Množina funkcí

Množina funkcí představuje soubor funkcí s argumenty, může se jednat o známé operátory či vlastní funkce. Funkce má danou aritu, tedy počet genů, který jí v dané funkci náleží. Tyto geny mohou být tvořeny terminály, nebo opět dalšími funkcemi. Dále je možné funkci přiřadit tzv. kategorii, tedy skupinu, do které náleží a která bude využita při zakládání populace. V práci jsou užity aritmetické a matematické funkce:

- „+“: sčítání dvou sčítanců. Tato funkce má aritu 2, tedy přijímá vždy dva parametry. Funkce vrátí součet obou hodnot.
- „-“: rozdíl menšence a menšitele. Tato funkce má aritu 2, kde první parametr je menšenec a druhý menšitel. Funkce vrátí rozdíl hodnot.
- „*“: součin dvou činitelů. Tato funkce má aritu 2. Funkce vrátí součin hodnot.
- „/“: podíl dělence a dělitele. Tato funkce má aritu 2, kde první parametr je dělenec a druhý dělitel. Pokud se dělenec rovná nule, pak funkce vrátí hodnotu 0, pokud dělitel se rovná nule, vrátí funkce kladné nekonečno, jinak vrátí podíl hodnot.
- „^“: mocnina čísla. Tato funkce má aritu 2, kde první parametr je základ a druhý parametr je exponent.
- „sqrt“: druhá odmocnina čísla. Tato funkce má aritu 1, kde parametr je argument odmocniny. Druhá odmocnina není pro záporná čísla definována,

v případě záporných čísel funkce vrátí kladné nekonečno, jinak vrátí druhou odmocninu argumentu.

- „cos“: kosinus čísla. Tato funkce má aritu 1. Vrátí hodnotu kosinu parametru.
- „sin“: sinus čísla. Tato funkce má aritu 1. Vrátí hodnotu sinu parametru.
- „tan“: tangens čísla. Tato funkce má aritu 1. Vrátí hodnotu tangensu parametru.
- „log10“: logaritmus o základu deseti. Tato funkce má aritu 1. Pokud je parametr záporný nebo roven 0, funkce vrátí kladné nekonečno, jinak vrátí exponent o základu deseti.
- „ln“: logaritmus o základu Eulerova čísla. Tato funkce má aritu 1. Pokud je parametr záporný nebo roven 0, funkce vrátí kladné nekonečno, jinak vrátí exponent o základu Eulerova čísla.

Funkce jsou rozdělený do kategorií, které jsou postupně do průběhu výpočtu zahrnovány. Účelem postupného přidávání je optimalizace nalezení jednoduších řešení. Jednolivé funkce jsou přidávány dle seznamu odshora dolů:

- +, -
- *, /
- ^
- sqrt
- log, ln
- sin, cos, tan

4 Přehled vybraných implementací symbolické regrese

Následující podkapitoly představují vybrané implementace symbolické regrese a jejich možnosti využití, nasazení a práci s nimi.

4.1 Clojush

Clojush je verze programovacího jazyka Push pro evoluční výpočty a genetického programovacího systému PushGP implementovaného v Clojure.

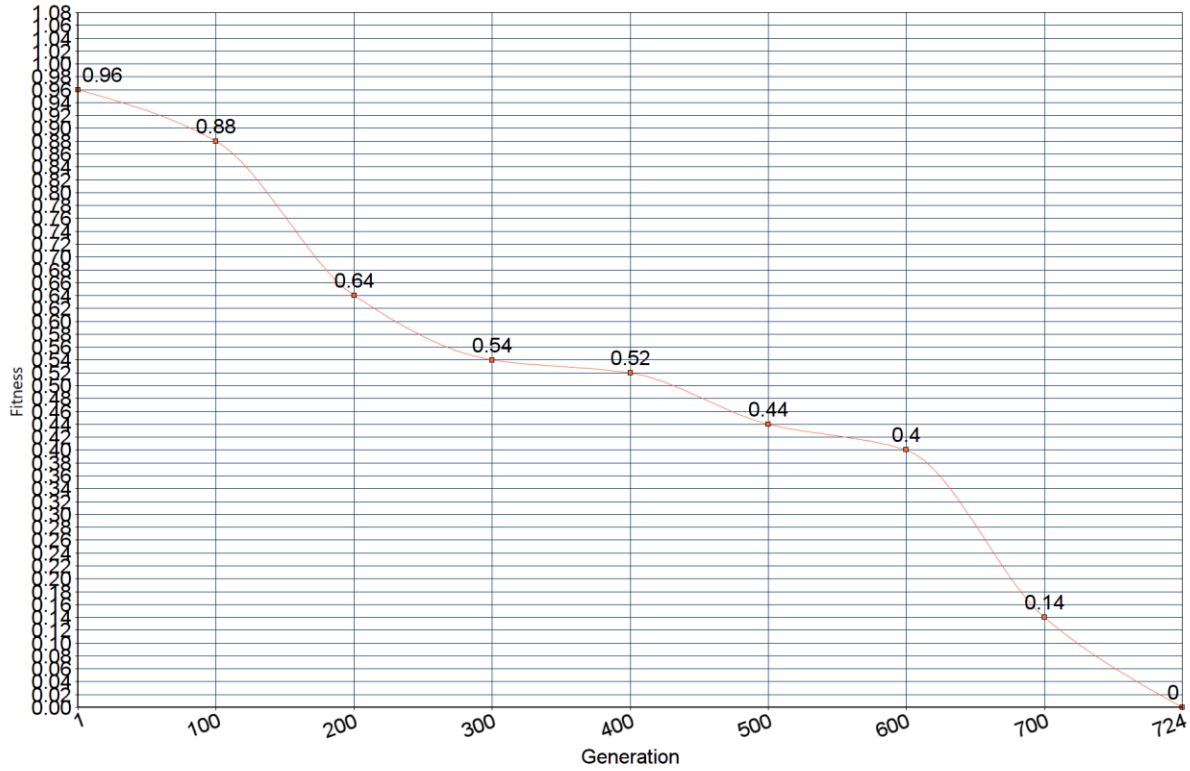
Clojush lze pustit dvěma způsoby. První způsob je přímé nainstalování potřebného prostředí, ideálně `Clojure 1.7.0`. Druhý způsob je využití Dockeru, který vytváří kontejner kolem puštěného programu a zajišťuje všechny potřebné balíčky a služby.

Byla testována symbolická regrese pro vzorec $F = X^3 + X^2 + X$. Byl zadán příkaz `lein run clojush.problems.demos.simple-regression :population-size 50`.

První pokus dopadl selháním programu; jak je psáno v dokumentaci, tento případ se může vyskytnout. V případě selhání program nevypíše nejlepší zvolený program a je nutné procházet ručně všechny generace a hledat optimální vlastnosti generace.

Vypočtené generace lze najít také ve logovacím souboru, pokud je dán příkaz na jeho vytvoření.

Druhý pokus, zadaný příkazem `lein run clojush.problems.demos.simple-regression :population-size 50 :error-threshold 0 :print-errors true :print-error-frequencies-by-case true` vyšel úspěšně a program vybral následující poslední generaci jako nejlepší chromozom. Chromozom popisoval funkci $F = X^3 + X^2 + X$ v postfixové podobě. Následující obrázek popisuje vývoj fitness hodnot nejlepších chromozomů v generacích:



Obrázek 6: Ukázka vývoje testu Clojush

4.1 EllenGP

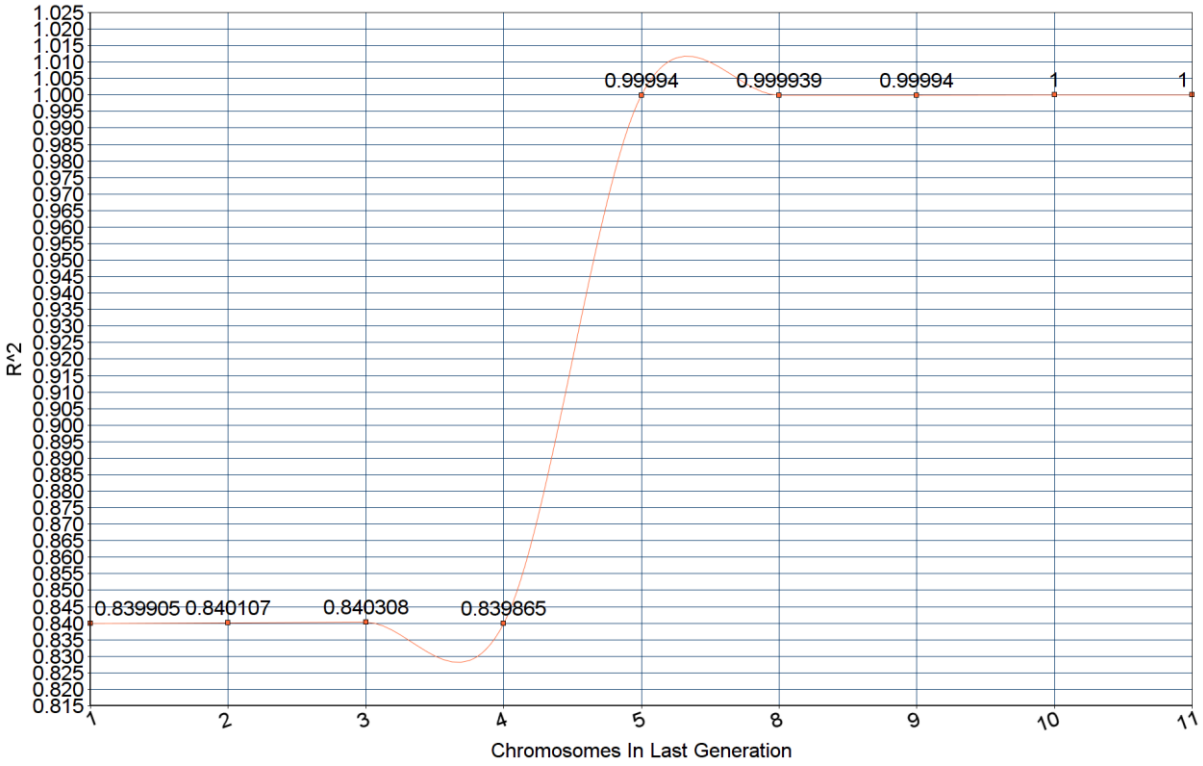
EllenGP nabízí sestavené projekty, `EllenGP.exe` a `RunTrials.exe`. Data a parametry lze upravit v souborech `sampledata.txt` a `sampleparams.txt`. Soubor `sampleparams.txt` obsahuje potřebné parametry pro řízení výpočtu. V případě chyby v `sampledata.txt` či `sampleparams.txt` celý program spadne na výjimku. Program jako výstup vrací výpisy na konzoli a současně generuje i logy do cesty uvedené v parametrech.

Při spuštění je na výpisu zobrazen průběh výpočtu a jednotlivé informace o generacích. V logovacích souborech lze najít detailnější informace o jednotlivých jedincích i poslední generaci.

Jako C++ projekt je spustitelný v prostředí Windows OS. Program se spouští příkazem `ellenGP sampleparams.txt sampledata.txt`.

V roce 2017 bylo podpora pro EllenGP ukončena a vývoj se přemístil na Python verzi Ellen, která funguje jako knihovna rozšiřující `sklearn`, tedy je možné knihovnu použít pro vlastní nasazení optimalizačních úloh.

V testu byla hledaná funkce $F = X^3 + X^2 + X$. Jako nejlepší chromozom se vrátil s hodnotou $F = (((X * (X + 1)) * X) + X)$, což po úpravě naprosto odpovídá hledané funkci (proto také R^2 bylo rovno jedné).



Obrázek 7: Ukázka poslední generace EllenGP

4.1 DEAP

DEAP je framework pro jazyk Python a prostředí Linux. Vyžaduje ke svému běhu minimálně Python 2.6, k pokročilejším funkcím Python 2.7. Je doporučeno při instalaci využít příkazu `pip`, `apt-get` instaluje zastaralé verze.

Umožňuje užívání struktur jako jsou například List, Array, Set, Dictionary, Tree, umožňuje silné i slabé typování, automaticky definované funkce, optimalizace evolučních strategií, paralelismus výpočtu. Také nabízí možnost porovnání s jinými metodami, například optimalizací hejnem částic.

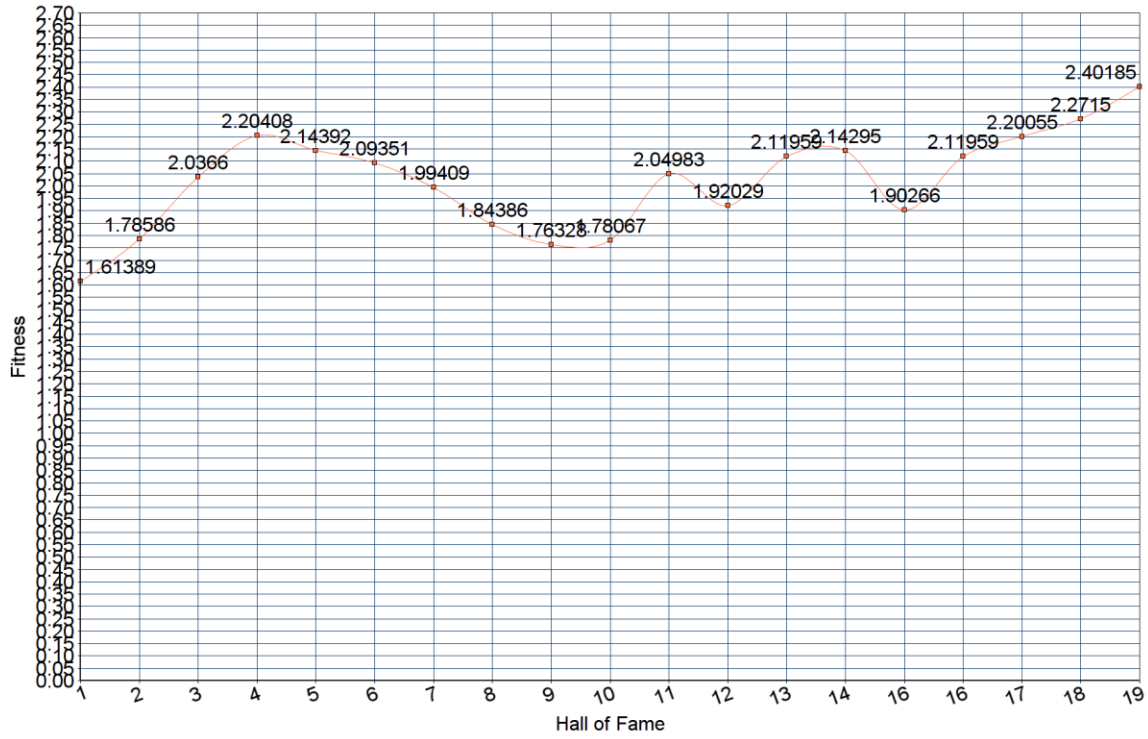
Protože se jedná o framework pro genetické programování, je třeba potřebná pravidla napsat. To může být pro uživatele neseznámené s jazykem Python a se strukturami používanými v DEAP zásádní nevýhoda. Nicméně modulárnost a univerzálost DEAPu

umožňuje popsat situace a jejich proceduru řešení, která by se u ostatních knihoven obtížně dosahovala.

Pro představení práce s DEAPem byla využita ukázka `examples.gp.symbreg.py`, která se upravila pro potřeby testování. Ve tutoriálovém nastavení nenabízí zobrazení nejlepší nalezené funkce. Pro zobrazení výsledků tzv. haly slávy stačí dopsat příkaz `print(hof)`. Hledaná funkce byla $F = X^3 + X^2 + X$.

Pro provedení výpočtu byl jako nejlepší jedinec vrácen $F = X * X * X * X + X * X + X$. Po zjednodušení je funkce v podobě $F = X^4 + X^2 + X$.

Následující graf ukazuje halu slávy daného testu:



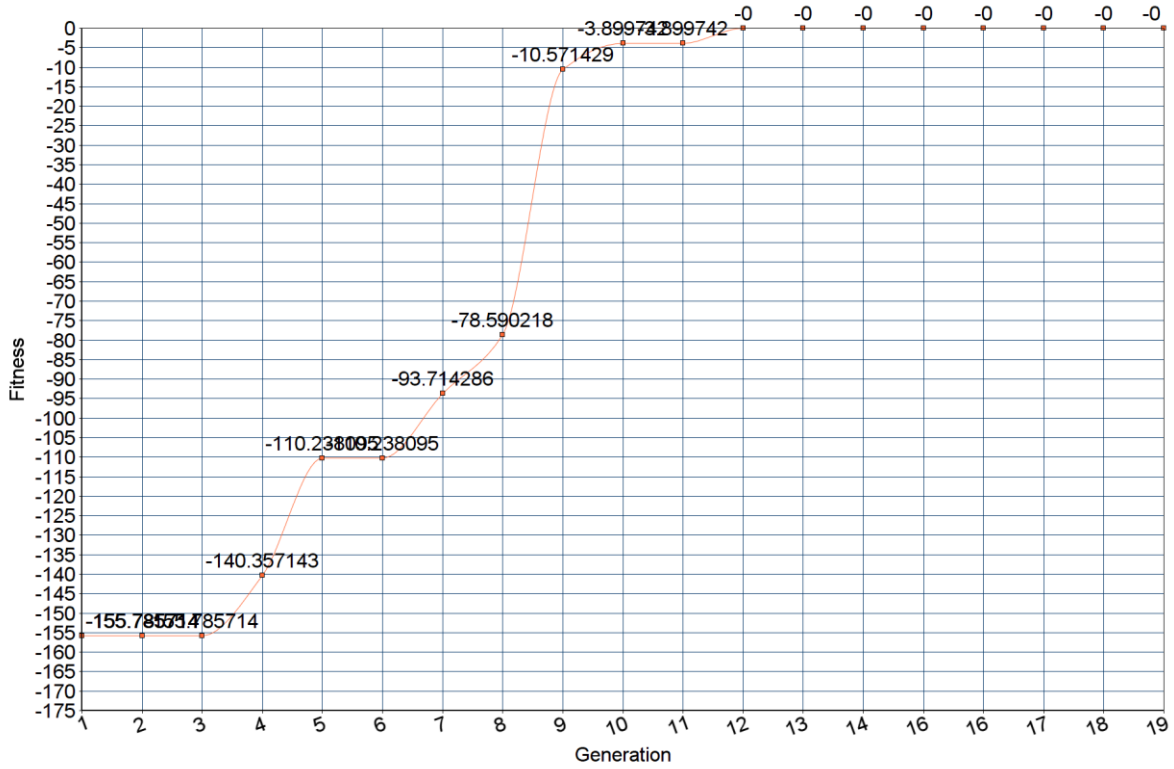
Obrázek 8: Ukázka DEAP Haly slávy

4.1 FlexGP

FlexGP je označení pro skupinu nástrojů využívající genetické programování. Všechny jejich kódy jsou volně přístupné, včetně příkladů a návodů.

FlexGP Learnes umožňují zpracovávat data lokálně, nebo v cloudu, pomocí souborů .csv. Zvládne zpracovat soubor i při nálezu chyby.

Jejich genetické programování se liší od konvenčních GP zejména v eliminaci přímého porovnání finálního výsledku programu s cílovou proměnou y. Místo toho naladí lineární kombinace všech podvýrazů programu s ohledem na cílový y. Poté porovná y s výsledkem regresního modelu. Následující graf ukazuje, jak během dvaceti generací o velikosti populace 100 našel optimální řešení, tedy fitness hodnota rovna 0 pro funkci $F = X^3 + X^2 + X$.



Obrázek 9: Ukázka vývoje FlexGP nástroje

4.1 KarooGP

KarooGP je aplikace napsaná v jazyce Python s využitím genetického programování. Podporuje práci na více jádrech procesoru a GPU díky TensorFlow, zpracovává data ve formě .csv. Nabízí uživatelské rozhraní a také různé režimy zobrazení, které mohou najít uplatnění při ukazování principu genetického programování.

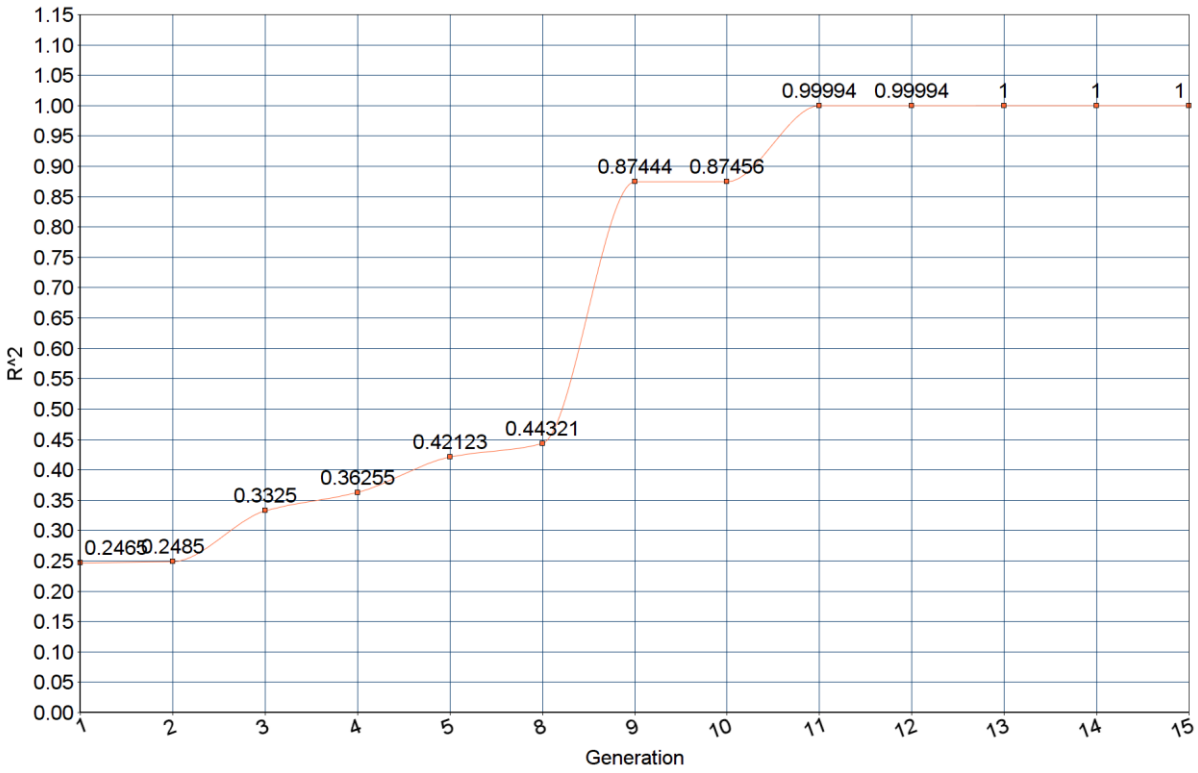
Požadavky pro spuštění KarooGP jsou Python 2.7.12 s `numpy 1.13.1`, `sympy 1.0`, `tensorflow 1.3.0`, `scikit-learn 0.18.1`, `matplotlib 2.0.0`. Pro Windows OS nebyl KarooGP implementován.

Nevýhoda KarooGP je absence velkého množství funkcí, přesněji podporuje jenom funkce +, -, *, /. Přestože v dokumentaci je uváděna i možnost využití funkcí sinus, kosinus, odmocniny a dalších, při testování těchto funkcí se vracely chyby už při tvoření druhé generace, kvůli kterým se nedařilo testy dokončit.

Další nevýhoda je neschopnost zpracovat poškozená data. V momentě výskytu prázdného prvku v souboru či nesprávného formátu čísla se celý program ukončí.

Po řadě testování se zvolila funkce $F = X^3 + X^2 + X$, která by mohla dostatečně otestovat úspěšnost KarooGP. Výstup programu lze po dokončení výpočtu sledovat v programu. MSE lze sledovat zapnutím pokročilého režimu vypisování.

Po testu vrátil v poslední generaci všechny chromozomy s funkcí $F = X^3 + X^2 + X$.



Obrázek 10: Ukázka vývoje generací v KarooGP

5 Implementace symbolické regrese

Pro implementaci genetického programování a symbolické regrese byl využit jazyk Java, verze 1.8. Aplikace běží v prostředí Java FX, verze 8. Součástí řešení programu je také XSL pro stylování výstupního protokolu.

Pro výpočet symbolické regrese se nejdříve načtou experimentální data, která jsou použita pro testování vypočtených formulí a následné porovnání vůči očekávanému výsledku. Poté jsou uživatelem vybrány použité funkce a terminály, které budou využity během založení a vytváření populací.

5.1 Nastavení vstupních parametrů

Pro řešení symbolické regrese bude uživatel aplikace moci nastavovat tyto parametry genetického algoritmu:

- velikost populace,
- počet generací,
- pravděpodobnost reprodukce,
- pravděpodobnost mutace,
- pravděpodobnost křížení,
- maximální hloubka stromu,
- maximální hloubka stromu po operaci,
- elitismus,
- decimace,
- tolerance.

Velikost populace je počet chromozomů, se kterými bude genetický algoritmus pracovat v každé generaci. Počet generací je počet iterací genetického algoritmu. Reprodukcí se rozumí kopírování chromozomů do další generace v nezměněné podobě (které probíhá na základě selekčního mechanismu, stejně jako křížení a mutace). Křížení vybírá dva chromozomy, z nichž vytvoří dva chromozomy pro další generaci pomocí vzájemné výměny dvou náhodně vybraných uzlů. Mutace vybírá na chromozomu náhodný uzel a ten pak nahradí náhodně vygenerovaným stromem, složeným z funkcí a terminálů. Hloubka stromu po operaci určuje, jak daleko mohou vygenerované stromy zasahovat.

Elitismus nabízí možnost kopírovat nejzdatnější chromozom ze současné generace do další generace beze změn. Neprojde tedy selekčním mechanismem ani na něm nebudou provedeny genetické operátory. Díky této funkci nemůže dojít k ztracení nejlepšího chromozomu poslední generace, který by jinak mohl být zracen buď nevýběrem selekčním mechanismem nebo úpravou genetickými operacemi jako mutace [3].

Decimace nabízí možnost dočasného navýšení velikosti populace, která je po určitém počku kroků výpočtu genetického algoritmu snížena na původní zadanou velikost populace. Velikost nové populace se rovná $N * T$, kde N je zadaná velikost počátečný populace a T je celočíselný parametr velikosti decimace. V práci je T nastaven na hodnotu 10. Počet vypočtených generací před aktivováním decimace je roven hodnotě T. V momentě redukce jsou všechny chromozomy srovnány dle hodnoty fitness funkce a ponecháno N nejlepších.

Tolerance je normalizovaná míra chyby nalezeného řešení. Při načtení dat se vypočte odhad ideální tolerance a uživatel může tuto hodnotu zvýšit či snížit. Obvykle se program snaží udržet míru tolerance na hodnotě 0. Při snížení této hodnoty může dojít k prodloužení celkového času výpočtu a zpřesnění výsledku řešení. Zvýšení hodnoty tolerance vede ke pravděpodobně rychlejšímu objevení řešení za cenu kvality nalezeného řešení.

Funkce jsou zvoleny a přidávány automaticky dle potřeby programem. V průběhu se jednotlivé skupiny funkcí (např. plus, mínus...) přidávájí do generování nové populace pomocí metody grow a tím se projeví ve výpočtu.

5.2 Algoritmus

Na začátku procesu vyhodnocování se zpracují uživatelem zadané parametry. Na začátku algoritmu se vytvoří populace ze zadaných funkcí a terminálů pomocí metody `grow`, která způsobí náhodné vytvoření stromů, kdy přidává terminály kdykoliv, nejpozději však v maximální hloubce stromu. Na základě počtu kategorií funkcí se tolikrát zakládá nový proces inicializace populace a nový celý výpočet GA. Poté, co je první náhodná populace založena, se iteruje tolikrát, kolik byl zvolen počet generací. Prvně je vybrán první nejlepší chromozom, poté je v každé iteraci na celé populaci provedeno procentuální zastoupení vybraných genetických operátorů, které upraví danou generaci. Proces se opakuje, dokud není splněna ukončovací podmínka.

V případě podezření na předčasnou konvergenci se výpočet opraví pomocí metody vnoření větší nové populace, metody nazvané *injection* [16].

5.2.1 Operátory

Genetické operátory (reprodukce, mutace, křížení) rozdělí populaci na tři části, kde každá část představuje podíl populace, která projde přiděleným genetickým operátorem. Rozdělení se počítá jako procentuální podíl, tedy reprodukce o hodnotě 1.0 způsobí, že se vybere na 100 % populace na reprodukci a zbylé dva operátory nebudou využity. Populace se rozděluje na jednotlivé operátory postupně, nejdříve se vybírá reprodukce, poté křížení a následně mutace.

Reprodukce na základě výběrového mechanismu získá chromozom, který duplikuje a přidá do nové generace. Reprodukce nepřináší žádný nový posun k lepším výsledkům, může však pomoci zachovat vítěze selekčního mechanismu [3]. V některých případech může reprodukce a elitismus splývat.

Křížení získává od výběrového mechanismu dva chromozomy ke zpracování, ze kterých následně vybere náhodný gen a pokusí se simulovat vzájemné nahrazení. Pokud by nahrazení způsobilo překročení maximální povolené hloubky stromu po křížení, což je jeden z možných uživatelem zadaných parametrů, je operace přerušena. Pokud je operace možná, provede se prohození vybraných genů, čím vznikají dva nové chromozomy [9].

Mutace vyžaduje jeden chromozom od selekčního mechanismu, na kterém náhodně vybere jeden gen. Zjistí si hloubku genu, a pokud je na hranici povolené maximální hloubky stromu po operaci, nahradí náhodně gen za terminál. V případě, že existuje prostor pro růst genu, je vybrána náhodná funkce pro nahrazení, ke které jsou posléze vytvořeny terminály dle arity dané funkce [9].

5.2.2 Implementace výběrového mechanismu

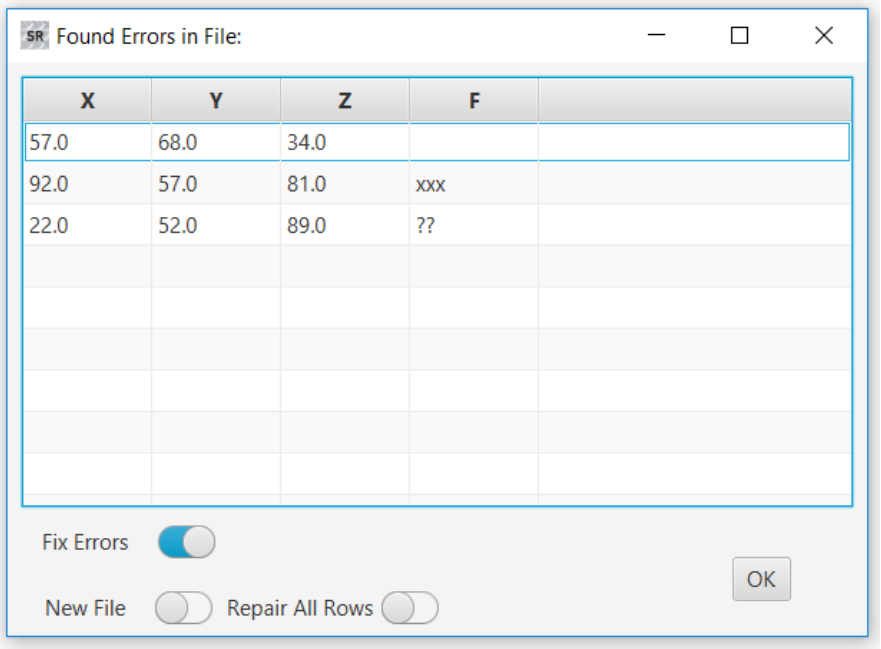
Výběrový, též selekční, mechanismus slouží pro vybrání chromozomů ke genetickým operacím. V práci je implementováno více variant turnajové selekce. Výchozí turnajová selekce nabízí výběr tří chromozomů.

Turnajová selekce vybere náhodně určitý počet chromozomů z populace, zvolí první chromozom jako vítěze a následně porovná jejich hodnotu fitness vůči o ostatním. Pokud je nalezen lepší chromozom, tedy chromozom s lepší fitness, stává se vítězem. Tato selekce zajišťuje náhodnější a rovnoměrnější výběr skrze populaci. V práci je implementována turnajová selekce s výběrem tří, dvou a pěti chromozomů.

5.3 Uživatelské rozhraní a ovládání

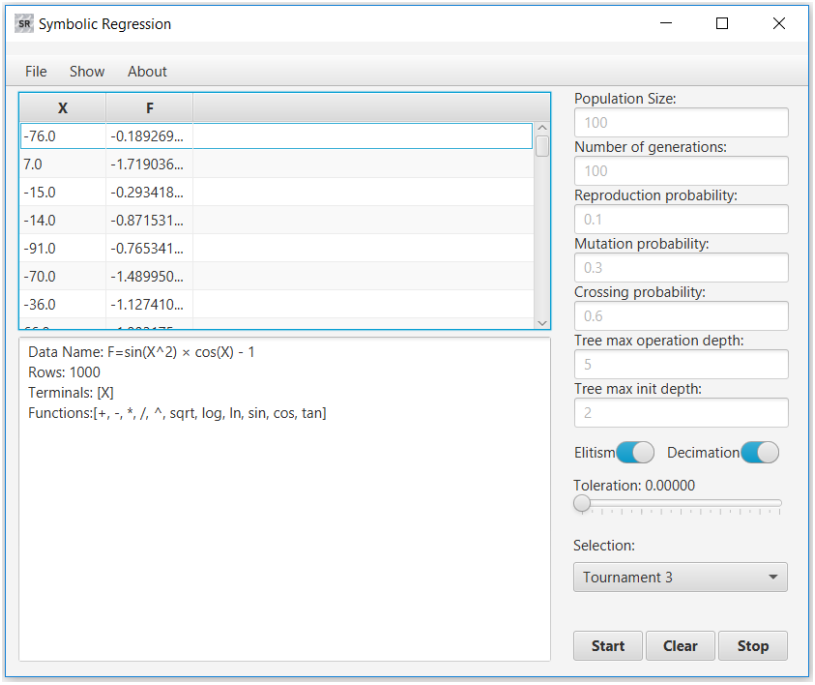
Uživatelské rozhraní aplikace slouží k ovládání výpočtu symbolické regrese a načtení k tomu potřebných dat a parametrů. Dále nabízí zobrazení a puštění konfigurace, otevření a zobrazení vypočtených konfigurací, upravení doplňujících nastavení a zobrazení výsledků programu v grafu. Uživatel má možnost zvolit data a vstupní parametry, například velikost populace, poměr křížení a jednotlivé funkce. V případě nevhodně zadaných parametrů se využijí výchozí hodnoty.

Při načtení souboru je uživatel upozorněn na možné chyby v souboru. Pokud je nalezen v souboru prvek, který je nezpracovatelný či chybí, je zobrazen v novém okně uživateli. Poté je nabídnuto soubor opravit a zda opravit všechny řádky dle později nalezené funkce či opravit řádky s nalezenou chybou, například chybějícím výsledkem. Následující obrázek ukazuje nález chyb v datech:



Obrázek 11: Ukázka okna s chybou v datech

Výpočet programu je možné ovládat třemi tlačítky (*Start*, *Clear*, *Stop*). *Start* spustí výpočet v novém vlákně, *Clear* vyčistí textový prostor a zruší přidané funkce a terminály. Pokud už běží výpočet, vyčištění funkcí a terminálů už výpočet neovlivní. Tlačítko *Stop* zastaví právě počítané vlákno a informuje uživatele o zastavení výpočtu. Zastavení výpočtu vrátí poslední nejlepší nalezené řešení a ukončí výpočet. Je tedy možné přes možnost zobrazení grafu se podívat na poslední nalezené řešení v grafické podobě.

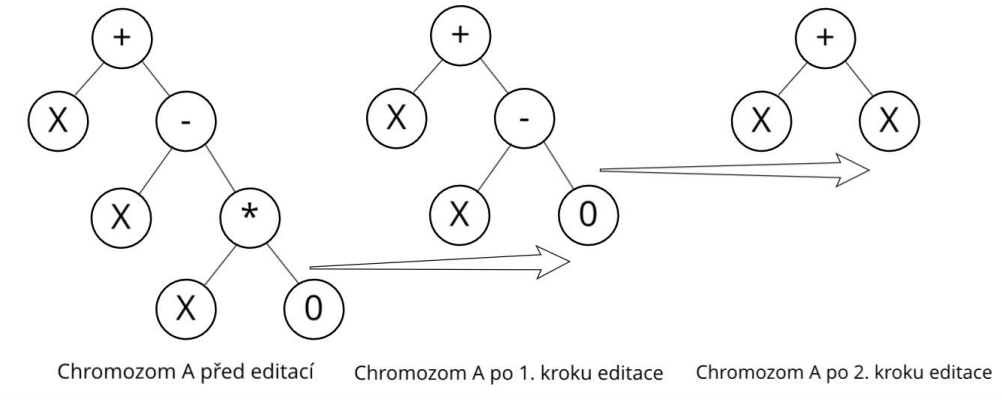


Obrázek 12: Ukázka programu po načtení dat

V okně *Setting* je možné nastavit způsob zobrazení výstupu programu. Je možné vypisovat informace o provádění genetických operací, o jednotlivých chromozomech a jejich fitness funkci. Ve výchozím nastavení jsou povoleny možnosti vytvoření logovacího souboru a výstupního protokolu a jejich otevření po dokončení výpočtu a vytvoření souborů.

5.4 Postprocesing

Výsledný výstup programu, tedy nejlepší chromozom poslední generace, je před vrácením ještě editován, aby se výraz, který chromozom představuje, zjednodušil. V případě přístupu k internetu se využije API Wolfram|Aplha [18]. V opačném případě je celý chromozom procházen a nachází se takové části, které lze jednoznačně určit. Pokud je taková část nalezena, rekurzivně se opakuje nahrazování, dokud další nahrazování není možné. Pokud byl chromozom pozměněn, je ještě na závěr opravena jeho aktuální hloubka.



Obrázek 13: Ukázka editace

5.5 Konfigurace

Konfigurace slouží pro zautomatizování většího počtu výpočtů s více druhy vstupních parametrů. Konfigurace se nastavuje v souboru typu xml, kde se definují parametry, které má program použít, např. počet spuštění, velikost populace, funkce, terminály. Pokud je program spuštěn přes konzoli, je do konzole vypisován ještě čas a pořadové číslo pro výpočet jednoho celého průběhu. V konfiguraci je možné navrhnout si také vlastní terminály, které není možné vybrat při manuálním použití aplikace. Je také možné spustit více konfigurací najednou, čímž lze získat rozdělené výsledky a tím jednodušeji porovnávat získané poznatky.

Spuštění konfigurace vypíná zapisování logů a výstupní protokol pro zvýšení výkonu. Pro sledování stručného postupu výpočtu je doporučeno spouštět program v konzoli, kde se průběžně vypisuje průběh postupu skrze konfigurace.

Puštění konfigurace proběhne prvně načtením testovacích dat a následně zvolením možnosti *Menu → Run Configuration.* Zobrazí se okno, kde se uživateli ukáže v needitovatelné podobě konfigurace a možnosti zrušení výpočtu a puštění výpočtu. Po kliknutí tlačítka *Start* je konfigurace puštěna.

Na obrázku dále je ukázka jedné konfigurace.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Application>
    <Configuration id="1">
        <NumberOfStarts>100</NumberOfStarts>
        <PopulationSize>50</PopulationSize>
        <NumberOfGenerations>50</NumberOfGenerations>
        <MaxDepthTreeInit>3</MaxDepthTreeInit>
        <MaxDepthTreeAfterCrossover>4</MaxDepthTreeAfterCrossover>
        <ReproductionProbability>0.1</ReproductionProbability>
        <MutationProbability>0.2</MutationProbability>
        <CrossingProbability>0.7</CrossingProbability>
        <Selection>0</Selection>
        <Elitism>true</Elitism>
        <Decimation>true</Decimation>
        <Functions>
            <Function>
                <Arita>1</Arita>
                <FunctionValue>sqrt</FunctionValue>
            </Function>
            <Function>
                <Arita>2</Arita>
                <FunctionValue>*</FunctionValue>
            </Function>
            <Function>
                <Arita>2</Arita>
                <FunctionValue>+</FunctionValue>
            </Function>
        </Functions>
        <Terminals>
        </Terminals>
    </Configuration>
</Application>
```

Obrázek 14: Ukázka konfigurace

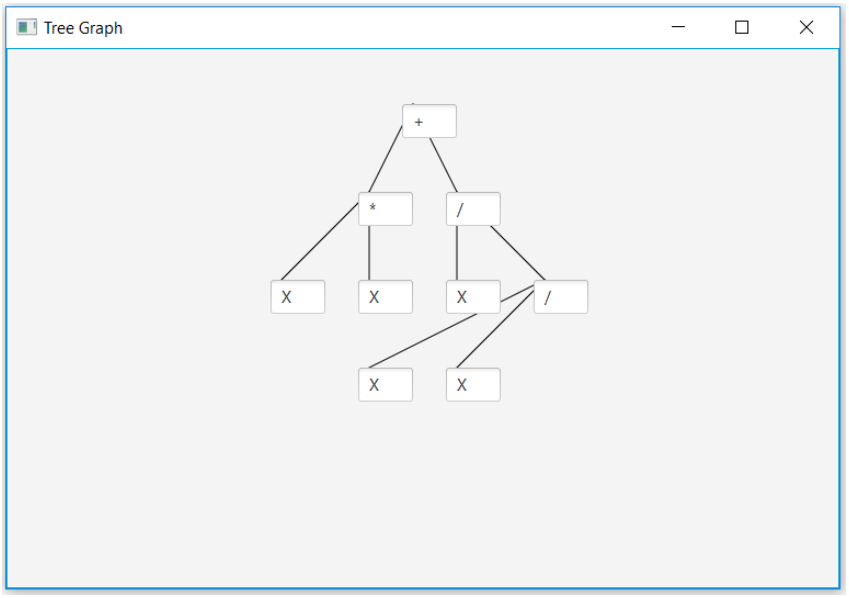
5.6 Výstupy

Po dokončení programu je do textového pole vytištěna objevená funkce a její přesnost. Přesnost představuje, v jaké procentuální míře se výstupnní funkce rovná oborem hodnot hledané funkci. 100 % přesnost znamená identickou funkci. Také je možné si zobrazit výsledky v podobě grafů.

5.6.1 Výstup manuálního puštění

Při vlastním spuštění se graf zobrazí přes položku menu *Show → Graph*. To zobrazí nové okno s vykresleným grafem, kde je vidět celý strom nejlepšího chromozomu. Graf je možné přibližovat a oddalovat.

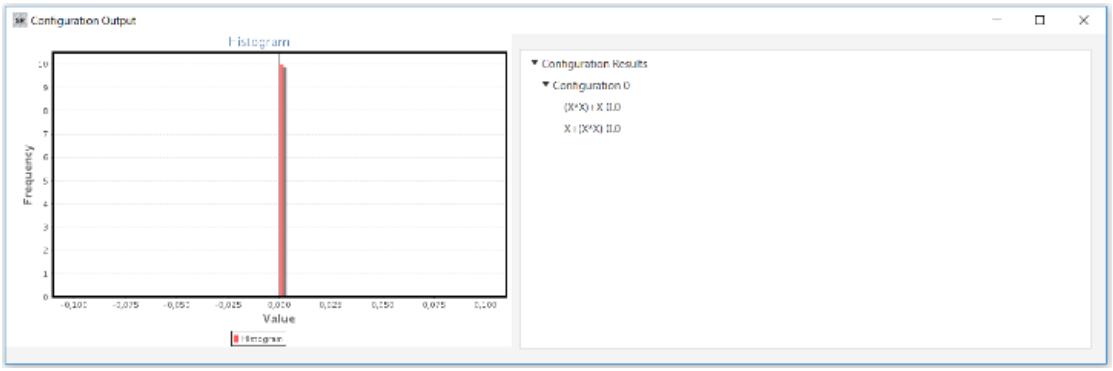
Graf je generovaný rekurzivním voláním, kdy při změně předka se zvýší hodnota výšky právě generované buňky prezentující gen chromozomu. Na vrcholu je kořen řešení, který je vždy funkcí. Spojnice ukazují aritu funkce, spojují rodiče a potomky, tedy operátory funkce. Listmy stromovitého grafu jsou vždy terminály. Ukázka okna grafu chromozomu:



*Obrázek 15: Graf chromozomu (X*X) + (X/(X/X))*

5.6.1 Výstup konfigurace

Výstupem konfigurace je *output_(název testovaného souboru).xml,* který lze otevřít pomocí možnosti *Menu → Open Output*. Ten obsahuje všechna nalezená řešení, jejich podobu stromu a jejich fitness hodnotu. Aplikace umí tento soubor otevřít a zpracovat obsah souboru do grafu, jak je uvedeno na obrázku níže. Pokud žádný soubor není zvolen, nebo je zvolen soubor s chybným obsahem, je uživatel upozorněn.



Obrázek 16: Ukázka výstupu konfigurace

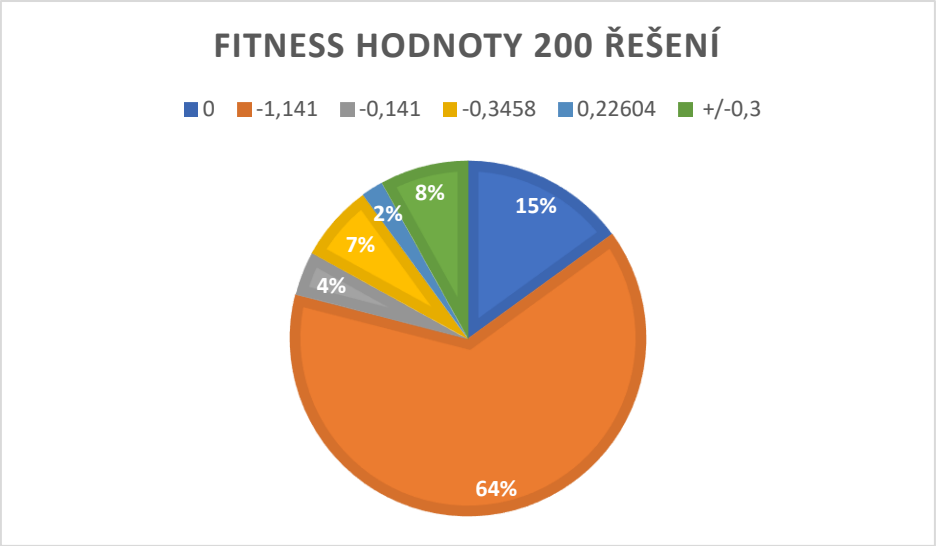
Graf představuje histogram fitness hodnot celé konfigurace. Osu X (Value) představuje fitness hodnoty výstupu, osa Y (Frequency) představuje počet nalezení daných fitness hodnot. Podoba řešení nemá na zobrazení v grafu vliv. Graf se dynamicky upraví dle objevených minimálních a maximálních fitness hodnot. Napravo se nachází TreeView, který ukazuje stromovitě strukturu vyppočtených konfigurací. Je možné tak vidět a vybrat si žádáné funkce dle velikosti i fitness funkce.

5.7 Řešení předčasné konvergence

Předčasná konvergence v genetickém programování znamená, že výsledek v průběhu genetického algoritmu směřuje k neoptimálnímu řešení. Důvodem může být přílišné upřednostňování chromozomů, nedostatečná velikost populace či špatně navrhnuté genetické operátory.

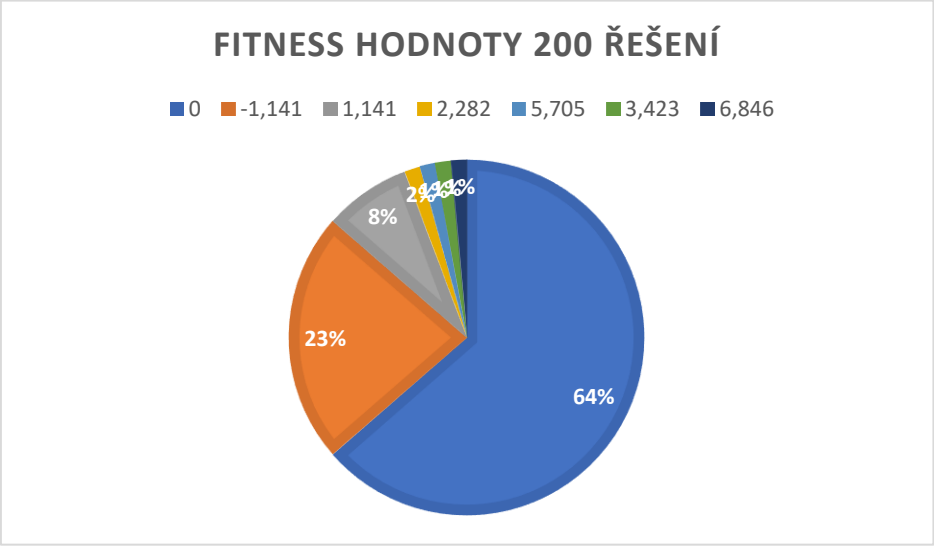
V práci se předčasná konvergence řeší v momentě podezření jejího vzniku, tedy pokud několik posledních nejlepších chromozomů v generaci mělo stejnou podobu a hodnotu fitness funkce; pak se spustí metoda *injection*. Ta se pokusí problém opravit zavedením nové populace o desetinásobně větší velikosti, podobně jako decimace na začátku inicializace populace. Aby nedocházelo k případné narůstající velikosti populace kvůli opakujícím se voláním *injection*, je nová populace znásobena pouze jednou za výskyt předčasné konvergence v průběhu jednoho výpočtu. To se může opakovat při vložení nových funkcí.

Následující grafy zobrazují stejné nastavení parametrů, kde první graf prezentuje řešení funkce $X^3 + X^2 + X$ bez *injection*, druhý graf s metodou *injection (Obrázek 16 a 17).*



Obrázek 17: Výstup funkce F=X^3+X^2+X bez možnosti detekce předčasné konvergence

První graf ukazuje, že nastavení parametrů nebylo zcela optimální. 64 % chromozomů uvízlo v lokálním minimu o fitness hodnotě -1.141, zatímco optimálních řešení pouhých 15 %.



Obrázek 18: Výstup funkce F=X^3+X^2+X s možností detekce předčasné konvergence

Jak je vidět na druhém grafu, metoda *injection* zvýšila šance nalezení optimálního řešení o 49 %. Kontrola předčasné konvergence a metoda *injection* je v programu trvale zapnuta.

5.8 Zdrojová data

Pro testování byla zvolena následující tabulka funkcí s náhodně vygenerovanými vstupními daty:

Functions	Fitcases
$F_1 = x^3 + x^2 + x$	1000 random points $\subseteq [-1,1]$
$F_2 = x^4 + x^3 + x^2 + x$	1000 random points $\subseteq [-1,1]$
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	1000 random points $\subseteq [-1,1]$
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	1000 random points $\subseteq [-1,1]$
$F_5 = sin(x^2)cos(x) - 1$	1000 random points $\subseteq [-1,1]$
$F_6 = sin(x) + sin(x + x^2)$	1000 random points $\subseteq [-1,1]$
$F_7 = log(x + 1) + log(x^2 + 1)$	1000 random points $\subseteq [0,2]$
$F_8 = \sqrt{x}$	1000 random points $\subseteq [0,2]$
$F_9 = sin(x) + sin(y^2)$	1000 random points $\subseteq [-1,1]$ x $[-1,1]$
$F_{10} = 2sin(x)cos(y)$	1000 random points $\subseteq [-1,1]$ x $[-1,1]$

Obrázek 19: Hledané funkce pro testování doporučené na gpbenchmarks.org [17]

6 Výsledky

Následující část zaznamenává získáné výstupy a uvádí použité parametry. Dle předešlé tabulky se vygenerovala testovací data, která byla využita při testování obou programů.

Pro porovnání byl zvolen z výše uvedených programů *EllenGP*. Ze všech uvedených je nejvíce podobný k *Symbolic Regression,* a to právě rozdělením parametrů, postupem vývoje chromozomů a jejich inicializací.

Mezi hlavní rozdíly v implementaci patří způsob uchovávání chromozomů; *EllenGP* pracuje na úrovni zásobníků, zatímco *Symbolic Regression* využívá stromovou strukturu.

Oba programy umožňují spuštění více testů najednou; *Symbolic Regression* skrze konfigurace, který chce pro své spuštění načtená data v programu a konfigurační soubor s nastavením. *EllenGP* nabízí tuhle možnost v odděleném programu nazvaném *RunTrials.exe*, který potřebuje pro své spuštění soubor ve formátu .txt, obsahující seznam souborů s daty a parametry. *RunTrials.exe* se spouští přes příkazovou řádku stejně jako *EllenGP.exe*.

Výsledky byly zjišťovány postupně, manuálním spouštěním programu, včetně načítání příslušných dat. Každý příklad testu byl spuštěn 10krát a následně vybrán nejlepší vrácený výsledek.

Cílem bylo vytvořit co nejbližší nastavení pro porovnatelné výstupy.

Použitá sestava:

- *OS: Microsoft Windows 10 Home 64 bit*
- *CPU: Intel(R) Core(TM) i7-4720HQ 2.60GHz, 4 jádra*
- *RAM: 12 GB DDR3*

6.1 Parametry vlastního programu

Následující tabulka představuje parametry programu, které byly použity při zjišťování výsledků programu *Symbolic Regression*:

Název parametru	**Hodnota parametru**
Počet populací	100
Počet generací	100
Pravděpodobnost mutace	0,3
Pravděpodobnost křížení	0,6
Pravděpodobnost reprodukce	0,1
Selekční mechanismus	0 (Turnajová selekce)
Počet prvků v turnajové selekci	3
Maximální hloubka stromu při vytvoření	2
Maximální hloubka stromu po operaci	5
Elitismus	Zapnuto
Decimace	Zapnuto

Tabulka 1: Tabulka parametrů

6.1 Výsledky vlastního programu

Očekávaná funkce	Výsledek	Přesnost	Doba výpočtu v sekundách	Počet generací
$\boldsymbol{F_1 = X^3 + X^2 + X}$	$F_1 = X\ (X\ (X + 1) + 1)$	100 %	23	1
$\boldsymbol{F_2 = X^4 + X^3 + X^2 + X}$	$F_2 = X\ (X\ (X\ (X + 1) + 1) + 1)$	100 %	28	2
$\boldsymbol{F_3 = X^5 + X^4 + X^3 + X^2 + X}$	$F_3 = 2\ (-1)\ X\hat{}3\ (X\hat{}2-3)$	99,63 %	89	43
$\boldsymbol{F_4 = X^6 + X^5 + X^4 + X^3 + X^2 + X}$	$F_4 = 3X\hat{}(X + 1)(X\hat{}X\ + (X\hat{}X)))$	99,61 %	124	53
$\boldsymbol{F_5 = \sin X^2 * \cos X - 1}$	$F_5 = sin(X\hat{}2)\ cos(X) - 1$	100 %	98	59
$\boldsymbol{F_6 = \sin X + \sin X^2 + X}$	$F_6 = sin(X) + sin(X^2) + X$	100 %	164	58
$\boldsymbol{F_7 = \log(X + 1) + \log(X^2 + 1)}$	$F_7 = 2X - \log(X^X)$	98,72 %	109	78
$\boldsymbol{F_8 = \sqrt{X}}$	$F_8 = \sqrt{X}$	100 %	61	1
$\boldsymbol{F_9 = \sin X + \sin Y^2}$	$F_9 = sin(((Y*X) + (cos(sin(X))\ /(Y\hat{}Y))))$	96,28 %	123	62
$\boldsymbol{F_{10} = 2 * \sin X * \cos Y}$	$F_{10} = 2 * sin(X * \cos(Y\hat{}2))$	98,72 %	179	53

Tabulka 2: Výsledky Symbolic Regression

Přesnost udává, v jaké míře nalezená funkce odpovídá očekáváné funkci. 100 % znamená, že funkce podávají stejné výsledky, bez ohledu na podobu funkce.

Přesnost se vypočítává pomocí aritmetického průměru vstupních očekávaných výsledků a fitness hodnoty zvoleného nejlepšího chromozomu. Udává se v procentech. Procenta vyšší od nuly značí vyšší odchylku aritmetického průměru od hledané funkce a pravděpodobně podávají rozdílné výsledky pro stejná vstupní data. Výstup výpočtu odpovídá vzorci:

$$Přesnost = 100 - \left(\frac{F_{ch}}{\frac{1}{n}\sum_{i=1}^{n} x_i} * 100 \right),$$

kde F_{ch} je fitness hodnota nejlepšího chromozomu (tedy chromozomu s nejnižší nalezenou fitness hodnotou), n je počet očekavaných výsledků a x_i je hodnota řádku i vstupních očekávaných dat.

6.3 Parametry EllenGP

Následující tabulka představuje parametry programu, které byly použity při zjišťování výsledků programu *EllenGP*:

Název parametru	**Hodnota parametru**
Počet populací	1000
Počet generací	100
Pravděpodobnost mutace	0,3
Pravděpodobnost křížení	0,6
Pravděpodobnost reprodukce	0,1
Výběrový mechanismus	1 (Turnajová selekce)
Počet prvků v turnajové selekci	3
Maximalní počet výpočtů	0 (neomezeně)
Elitismus	Zapnuto

Tabulka 3: Parametry EllenGP

Všechny parametry se nastavují ručně v souboru formátu .txt, neobsahuje všechny možnosti nastavení, je možnost je manuálně doplnit.

Počet populací byl zvolen 10krát větší než *Symbolic Regression* z důvodu simulování možnosti decimace, která v *EllenGP* není.

6.4 Výsledky EllenGP

Očekávaná funkce	Výsledek	Přesnost	Doba výpočtu v sekundách	Počet generací
$\boldsymbol{F_1 = X^3 + X^2 + X}$	$F_1 = ((x + ((x * x) * x)) + x)$	99,94 %	24	64
$\boldsymbol{F_2 = X^4 + X^3 + X^2 + X}$	$F_2 = ((x * x) + ((x * x) + x))$	92,48 %	38	42
$\boldsymbol{F_3 = X^5 + X^4 + X^3 + X^2 + X}$	$F_3 = ((x * x) * x)$	94,68 %	41	49
$\boldsymbol{F_4 = X^6 + X^5 + X^4 + X^3 + X^2 + X}$	$F_4 = (x * x)$	77 %	52	33
$\boldsymbol{F_5 = \sin X^2 * \cos X - 1}$	$F_5 = cos(exp(cos((sin((x * x)) + cos(x)))))$	83,11 %	46	57
$\boldsymbol{F_6 = \sin X + \sin X^2 + X}$	$F_6 = sin(x)$	57,51 %	54	78
$\boldsymbol{F_7 = \log(X + 1) + \log(X^2 + 1)}$	$F_7 = (log(log(x)) + log(x))$	99,5 %	3	3
$\boldsymbol{F_8 = \sqrt{X}}$	$F_8 = \sqrt{X}$	100 %	2	2
$\boldsymbol{F_9 = \sin X + \sin Y^2}$	$F_9 = (cos((x/y)) * (sin(x) * cos(exp(sin(cos(sin(y)))))))$	26,3 %	42	17
$\boldsymbol{F_{10} = 2 * \sin X * \cos Y}$	$F_{10} = (cos((sin(x) - cos(y))) - cos((x - x)))$	84,39 %	39	83

Tabulka 4: Výsledky EllenGP

Každá funkce byla spuštěna desetkrát na EllenGP se stejnými daty a podobným nastavením jako ve vlastním programu. Možnost decimace EllenGP nenabízí. Z množiny výsledků byl poté vybrán nejlepší (ne nejčastější) nalezený výsledek, tedy chromozom s nejvyšším koeficientem determinace. Přesnost EllenGP odpovídá vypisovanému R^2 (koeficient determinace).

Uvedený počet generací ukazuje, v jaké generaci bylo nalezeno konečné řešení. EllenGP neukončuje výpočet ani když najde zcela optimální výpočet.

Z pokusů na EllenGP plyne, že je velmi náchylný na předčasnou konvergenci a neposkytuje žádnou možnost její detekce a opravy.

6.3 Porovnání výsledků

Následující tabulka porovnává přesnost a rychlost výpočtu dané funkce programů *Symbolic Regression* a *EllenGP*.

	Symbolic Regression		***EllenGP***	
Funkce	**Přesnost**	**Rychlost v sekundách**	**Přesnost**	**Rychlost v sekundách**
$F_1 = X^3 + X^2 + X$	100 %	23	99,94 %	13
$F_2 = X^4 + X^3 + X^2 + X$	100 %	118	92,48 %	13
$F_3 = X^5 + X^4 + X^3 + X^2 + X$	99,63 %	89	94,68 %	15
$F_4 = X^6 + X^5 + X^4 + X^3 + X^2 + X$	99,61 %	128	77 %	15
$F_5 = \sin X^2 * \cos X - 1$	100 %	98	83,11 %	46
$F_6 = \sin X + \sin X^2 + X$	100 %	164	57,51 %	54
$F_7 = \log(X + 1) + \log(X^2 + 1)$	98,72 %	109	99,5 %	39
$F_8 = \sqrt{X}$	100 %	61	100 %	2
$F_9 = \sin X + \sin Y^2$	96,28 %	123	26,3 %	48
$F_{10} = 2 * \sin X * \cos Y$	98,72 %	179	84,39 %	39

Tabulka 5: Porovnání výsledků programů

7 Závěr a doporučení

Z porovnání plyne, že *Symbolic Regression* nabízí větší přesnost, ale za cenu delšího výpočetního času. Je to dáno hlavně implementací detekcí předčasné konvergence a schopnosti opravy. Předčasná konvergence je častým problémem GA. *EllenGP* neobsahuje řešení předčasné konvergence a jeho výsledky mají tendenci zůstat v lokálním maximu.

Pro načtení dat pro *Symbolic Regression* stačilo vybrat z nabídky grafického rozhraní a vybrat soubor. Zpracoval soubory typu .xls stejně jako .csv, a v případě chyb v datech na ně upozornil a dokázal data zpracovat. *EllenGP* zpracovává pouze soubory typu .csv a v případě chyby datech či chyby v nastavení parametrů (například chybějící údaj, jiný typ řádové čárky, uvedení proměnné navíc) program spadne bez varování.

Program *Symbolic Regression* využívá pro generování nových jedinců metodu *grow*, kterou omezuje pomocí hloubky stromu, zatímco *EllenGP*, využívající stejnou metodu, omezuje počet uzlů ve stromu. Omezení hloubkou umožňuje nacházet větší variace řešení a tím se dostat z lokálního extrému pravděpodobněji. Metoda *grow* se ukazuje jako nejefektivnější metoda tvoření stromů.

Program *Symbolic Regression* je jednodušší na ovládání i pro uživatele, který nemá zkušenosti s genetickým programováním. *EllenGP* vyžaduje vyšší znalost genetického programování a případně nutné úpravy v souborech pro samotné zprovoznění programu.

Symbolic Regression nabízí lepší zpracování výsledku, a to vytvořením výstupního protokolu v PDF souboru, který je vygenerovaný z XML. Tento XML soubor je posléze transformován pomocí XSLT do PDF formátu. Soubor *Template.xsl* je možné editovat a umožňuje tak pokročilému uživateli úpravu podoby výstupních dat.

Symbolic Regression nabízí také okamžité uplatnění zjištěné funkce, a to právě na možnosti opravy vstupních dat. Pokud byla v datech objevena chyba (například chybějící výsledek či špatný formát čísla), program daný řádek/řádky opraví dle předchozích nastavení uživatele při zobrazení okna s chybou v datech. *EllenGP* podobnou funkcionalitu nenabízí.

Symbolic Regression nabízí jazykové mutace programu. Ve výchozím stavu se pouští v anglickém jazyce, ale je možné doplnit soubor *localization.properties* o další jazyk.

Při nadbytečné velikosti populace nemá decimace na výsledek valný vliv. Přesto je doporučeno mít decimaci povolenou pro větší inicializační variaci a následně rychlejší průběh celého výpočtu. Decimace způsobí pozitivní dopad na celkový potřebný čas výpočtu, aniž by klesla dostačující velikost populace. V případě, kdy není známé, jaké parametry (např. počet chromozomů v populaci) je ideální nastavit, decimace dokáže tento problém vyřešit. Její rozšíření počáteční populace vede k rozšíření variability a následné zmenšení zvyšuje průměrnou kvalitu chromozomů skrze populaci.

Reprodukce má velmi malý vliv na nalezení vhodného výsledku, a proto je doporučeno držet tuto hodnotu minimální, či dokonce nulovou.

Mutace dokáže vytvořit různé varianty řešení, ale často vybere chromozom, který byl už ideální a pozmění ho na horší. V případě vyšší hodnoty mutace je doporučováno zapnutí elitismu, který zajistí přežití elitních chromozomů, tedy chromozomů s nejlepší fitness funkcí v generaci.

Křížení se ukazuje jako nejlepší metoda pro nalezení řešení, a proto se doporučuje pro nastavení křížení jako nejvyšší hodnoty ze všech tří možných genetických operací.

Volba více chromozomů do selekce umožnila přesnější výběr lepších chromozomů, za cenu výpočetního času a také rizika předčasné konvergence. Z těchto důvodů je doporučeno nechat výchozí nastavení turnajové selekce výběru tří chromozomů.

Cílem této bakalářské práce bylo seznámení se s hlavními principy genetického programování, jejich využití při řešení problému symbolické regrese a implementování vlastní aplikace pro řešení symbolické regrese za pomoci genetického programování. Tento problém byl řešen vlastní aplikací v Javě, nazvanou *Symbolic Regression*.

Byly popsány postupy využití genetického programování. Byl vysvětlen princip genetického algoritmu, reprezentace chromozomů, tvoření nových generací, vyhodnocování fitness funkce chromozomů a jejich výběr na základě jejich fitness. Dále se práce zabývala problémem symbolické regrese a jejím řešením za pomocí genetického programování.

Dle výše uvedéného lze shrnout, že bylo v rámci této bakalářské práce dosaženo stanovených cílů.

8 Seznam použité literatury

[1] Goldberg, D.E. & Holland, J.H. Machine Learning (1988) 3: 95. https://doi.org/10.1023/A:1022602019183

[2] Liu, J. & Lampinen, J. Soft Comput (2005) 9: 448. https://doi.org/10.1007/s00500-004-0363-x

[3] KOZA, John, R. Genetic Programming on the Programming of Computers by Means of Natural Selection. Cambridge: The MIT Press, 1998. 813 s. ISBN 0-262-11170-5.

[4] OŠMERA, Pavel. *Evoluční algoritmy a jejich aplikace: Evolutionary algorithms and their applications*. V Praze: České vysoké učení technické, 2008. ISBN 978-80-01-04192-5.

[5] BANZHAF, Wolfgram, NORDIN, Peter, KELLER, Robert E., FRANCONE, Frank D., Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. San Francisco: Morgan Kaufmann Publishers, Inc., 1998. 480 s. ISBN 1-55860-510-X.

[6] *Advances in genetic programming*. Cambridge, Mass.: MIT Press, 1994. Complex adaptive systems. ISSN 1524-8828.

[7] Geyer-Schulz Andreas: Fuzzy Rule-Based Expert System and Genetic Machine Learning (2nd ed.), Heidelberg, Physica-Verlag, 1997.

[8] Ryan C., Collins J., Neill M.O. (1998) Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf W., Poli R., Schoenauer M., Fogarty T.C. (eds) Genetic Programming. EuroGP 1998. Lecture Notes in Computer Science, vol 1391. Springer, Berlin, Heidelberg

[9] MILLER, Brad L., et al. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 1995, 9.3: 193-212.

[10] HYNEK, Josef. Genetické algoritmy a genetické programování. Grada, 2008. ISBN 24760575.

[11] FONSECA, Carlos M., et al. Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization. In: *Icga*. 1993. p. 416-423.

[12] KOZA, John R. Genetic Programming II Videotape: The Next Generation. Cambridge, MA: MIT Press, 1994.

[13] NORDIN, Peter; FRANCONE, Frank; BANZHAF, Wolfgang. Explicitly de ned introns and destructive crossover in genetic programming. *Advances in genetic programming*, 1995, 2: 111-134.

[14] POLI, Riccardo; LANGDON, William B. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 1998, 6.3: 231-252.

[15] AUGUSTO, Douglas Adriano; BARBOSA, Helio JC. Symbolic regression via genetic programming. In: *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*. IEEE, 2000. p. 173-178.

[16] R. Tin´os and S. Yang. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genetic Programming and Evolvable Machines, 8(3):255–286, May 2007.

[17] JAMIOLAHMADI, Saeed a Ahmad BARARI. A Genetic Programming Approach to Model Detailed Surface Integrity of Additive Manufacturing Parts. *15th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2015*. 2015, **2015**(3), 2339--2344. ISSN 2405-8963.

[18] *Wolfram|Alpha* [online]. Champaign, IL, 2009 [cit. 2019-03-13]. Dostupné z: products.wolframalpha.com/api

9 Přílohy

A. Seznam obrázků

B. Seznam tabulek

C. Uživatelská dokumentace

Pro spuštění programu je potřeba mít nainstalovaný JRE (Java Runtime Environment), minimálně ve verzi 1.8.

Aplikace využívá symbolické regrese na vybraná data. Cílem aplikace je najít takovou popisnou funkci dat, která nejlépe vystihuje vztah mezi daty a jejich výsledky a případně upozornit na chyby ve vstupních datech a nabídnout jejich opravu za pomoci právě později vypočtené funkce.

Grafické rozhraní aplikace se skládá z menu, prostoru pro zobrazení tabulky, textového výstupu a po pravé straně sloupce pro zadávání parametrů a ovládací tlačítka. V menu je možnost vybrat soubor, spustit konfiguraci, otevřít výstup vypočtené konfigurace, otevřít okno s doplňujícím nastavením, ukončit aplikaci, zobrazit graf nejlepšího chromozomu a zobrazit informace o verzi programu. Tabulka slouží pro zobrazení vybraného načteného souboru, textová oblast pak slouží pro informování o průběhu programu a výpočtu.

Pro spuštění procesu je potřeba načíst odpovídající soubor ve formátu .xlsx nebo .csv zvolením možnosti *Menu – LoadFile*. Ostatní parametry jsou dobrovolné (použijí se výchozí hodnoty, které jsou zobrazeny šedou barvou, pokud je pole nevyplněné).

Vstupní data musí být pro správné načtení .xlsx souboru ve formátu podobném jako v níže uvedeném příkladu. První řádek, hlavičky, definuje názvy terminálů. Poslední sloupec je použit jako sloupec očekávaných výsledků. To znamená, že tabulka by měla vždy mít poslední sloupec výsledky empirických dat. Data mohou být neúplná, program na chyby v souboru upozorní a opraví, pokud tak uživatel zvolí. Velikost tabulky není omezená.

Po načtení dat se zobrazí okno s informací, zda byly nalezeny chyby v souboru. Pokud se žádné chyby v souboru nevyskytují, je zobrazena prázdná tabulka s odpovídajícím popiskem okna. V případě chyb v datech se zobrazí závadné řádky. V obou případěch uživatel kliknutím tlačítka ok okno uzavře, popřípadně zaškrtne, jak má být naloženo s chybami v datech.

R1	R2	I1	I2	R
40	10	0,25	1	8
5	5	2	2	2,5
10	40	1	0,25	8
6	4	1,666	2,5	2,4
4	8	2,5	1,25	2,666
1	4	10	2,5	0,8
20	20	0,5	0,5	10
5	20	2	0,5	4
4	6	2,5	1,666	2,4

Obrázek 20: Ukázka vstupních dat ve formátu xlsx (Naměřená data při napětí U = 10V) Zdroj:[U]

Vstupní data pro .csv soubor musí být pro správné načtení podobné jako v níže uvedeném příkladu. První řádek deklaruje hlavičky, poslední pojmenování, stejně jako v případě .xlsx souboru, označuje sloupec výsledků.

```
X,Y,Z
1,2,3
3,4,5
6,7,8
9,10,11
12,13,14
```

Obrázek 21: Ukázka csv souboru dat

Parametry

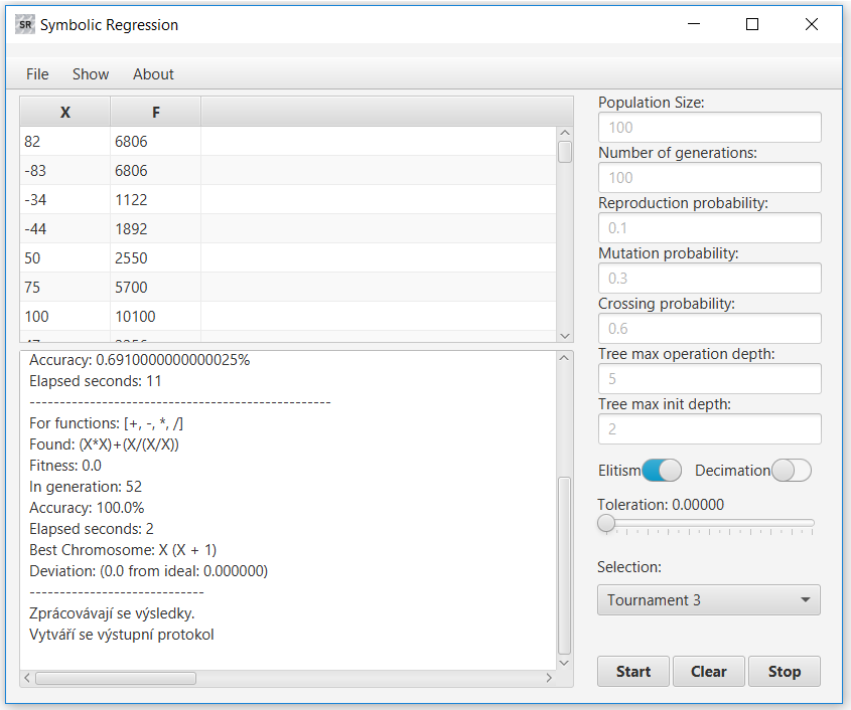
- Population Size: určuje velikost populace v generaci. Větší velikost má značný vliv na nalezení lepšího řešení, zároveň ale zvýší dobu výpočtu.
- Number of Generations: určuje počet generací. Větší počet zvýší pravděpodobnost nalezení lepšího řečení, ale za cenu vyšší časové náročnosti.
- Reproduction probability: hodnota je zadávána v intervalu <0; 1>. Určuje pravděpodobnost, s jakou se využije reprodukce při tvoření nových chromozomů.
- Mutation probability: hodnota je zadávána v intervalu <0; 1>. Určuje pravděpodobnost, s jakou se využije mutace při tvoření nových chromozomů.
- Crossing probability: hodnota je zadávána v intervalu <0; 1>. Určuje pravděpodobnost, s jakou se využije křížení při tvoření nových chromozomů. Doporučuje se toto číslo nastavit na vyšší hodnotu než mutace a reprodukce.

- Selection: volba turnajové selekce. Při volbě turnajové selekce je ještě možné zvolit počet chromozomů, který jde do soutěže (tento počet je uveden v závorce). Jako výchozí selekční mechanismus je vybrána turnajová selekce o třech chromozomech.
- Elitismus: povolí kopírování nejlepších chromozomů beze změny do další generace.
- Decimation: znásobí počáteční velikost populace pro zvýšení variace chromozomů a v později přebytek na základě fitness eliminuje.
- Toleration: Normalizovaná míra tolerance chyby výstupní funkce.

Ovládací prvky

- Start: tlačítko, které spustí samotný proces vytváření chromozomů a jejich výpočet.
- Clear: tlačítko, které vyčistí textový prostor a vyčistí seznam zvolených funkcí.
- Stop: tlačítko, které zastaví právě probíhající genetický algoritmus.

Po spuštění výpočtu, tedy po kliknutí na tlačítko Start, se na obrazovce spustí výpis nejlepších nalezených chromozomů v jednotlivých kategoriích. Na konci výpočtu se vypíše nejlepší nalezený chromozom a odchylka od ideální hodnoty. Na začátku textového pole je vypsán název souboru dat, počet řádků souboru, jaké terminály a funkce budou použity ve výpočtu a poté jaký byl zvolen nejlepší chromozom, zobrazený ve zjednodušené podobě nakonec odchylka od ideální hodnoty, tedy od hodnoty nula.



Obrázek 22: Ukázka výstupu programu pro test F=X^2+X

V nastavení lze upravit, zda se má po dokončení výpočtu vytvořit a zobrazit hlášení o výpočtu. To obsahuje všechny operace prováděněné během genetického algoritmu, včetně oznámení o fitness hodnotách chromozomů, vložení funkcí, konvergence a postupu skrze generace.

Součástí nastavení je i možnost vytvoření výstupního protokolu v podobě PDF, ve výchozím stavu je tato možnost povolena. Tento soubor pak obsahuje adresu dat, počet řádků dat, seznam terminálů, funkcí a statistiku vstupních dat. Také obsahuje závěrečné nejlepší řešení a celkový čas potřebný pro nalezení řešení. Výstupní protokol může vypadat následovně:

Symbolic Regression Output Protocol

Data Path	..\GP\F=2*sin(X)*cos(Y)
Rows	1000
Terminals:	[X, Y]
Functions:	[+, -, *, /, ^, sqrt, log, ln, sin, cos, tan]
Standart Deviation:	1.0100034653405898
Mean:	-0.02201399740926073
Min:	-1.9984136683727074
Max:	1.9784216588744838
Median:	-0.019555890900658672
Number of Generations:	100
Size of Population:	100
Tree Max Init Depth:	2
Tree Max Depth After Operation:	5
Crossover:	0.6
Reproduction:	0.1
Mutation:	0.3
Elitismus:	True
Decimation:	True
For functions:	[+, -, *, /, ^, sqrt, log, ln, sin, cos, tan]
Found:	2 * sin(X) * cos(Y)
Fitness:	0.0
In generation:	53
Accuracy:	100.0%
Elapsed seconds:	129
Total Elapsed Time:	02:06::878
Best Chromosome:	2*sin(X)*cos(Y)
Deviation:	0.0 from ideal: 0.000000

Obrázek 23: Ukázka výstupního protokolu

D. Technická dokumentace

Projekt se skládá ze tří hlavních balíčků: *model*, *view* a *controller*. Balíček *model.genetic* obsahuje třídy pro implementaci genetického algoritmu, *controller* obsahuje ovládací prvky grafického rozhraní, v balíčku *model* obsluhu dat pro genetický algoritmus a *view* obsahuje hlavní spouštěcí třídu a třídu pro zobrazení grafů.

Balíček *view* obsahuje třídy:

- MainApp: spouští hlavní okno aplikace.

Balíček *controller* obsahuje třídy:

- ErrorWindowController: třída ovládající zobrazení nalezených chyb v souboru a nastavení možností opravy souboru.
- XFMLController: řídící třída hlavního okna, zodpovědná za zpracování zadaných parametrů pro GPController,
- SettingWindowController: třída ovládající další nastavení, týkající se hlavně výstupů programu,
- ConfigurationWindowsController: třída ovládající zobrazení histogramu vypočtených řešení konfigurací a zobrazení stromu konfigurací.
- OutputWindowController: třída ovládající zobrazení grafu nejlepšího vypočteného chromozomu.

Balíček *model* obsahuje třídy:

- IReader (interface): rozhraní pro načítaní souboru,
- CSVReader: načítání .csv souborů,
- XLSXReader: načítání .xlsx souborů,
- IWriter (interface): rozhraní pro zápis do souboru,
- CSVWriter: zápis do .csv souboru,
- XLSXWriter: zápis do .xlsx soubor,
- Localizator: umožňuje změnu jazyka dle nastavené kultury,
- Messenger: záznamník zpráv a třída odpovědná za textový výstup,

- GPThread: vlákno, které obsahuje GeneticAlgorithm a předává mu potřebné parametry,
- ConfigurationThread: vlákno, které obsluhuje puštění konfigurací.
- Graph: třída obsluhující zobrazení grafu chromozomu,
- DataHandler: třída obalující data pro aplikaci,
- FileHandler: třída obalující data pro práci s načteným souborem.

Balíček *model.genetic* obsahuje třídy:

- Gen: základní třída představující uzel ve stromu (chromozomu),
- Terminal: vychází z Gen, představuje list stromu, tedy konečný bod,
- Function: vychází z Gen, představuje uzel stromu, tedy operační část,
- SelectedGen: odkaz na Gen, který byl vybrán nějakým selekčním mechanismem,
- Editation: třída nabízející metody pro optimalizaci stromů za cenu snížení jejich diversity,
- Fitness: třída představující fitness hodnotu chromozomu,
- Chromosome: třída daná kořenovým genem a fitness, představuje strom,
- GeneticAlgorithm: třída zodpovědná za celý výpočet GA.

Projekt je vyvíjen v Javě verze 1.8. s podporou JavaFX verze 8.0.131. Projekt obsahuje volně licencované knihovny třetích stran (ControlsFX, FilenameUtils, XSSF, FOP).

ControlsFx je open source projekt pro JavaFX, který se poskytuje rozšiřující UI prvky a další ovládácí nástroje, které základní JavaFX neposkytuje.

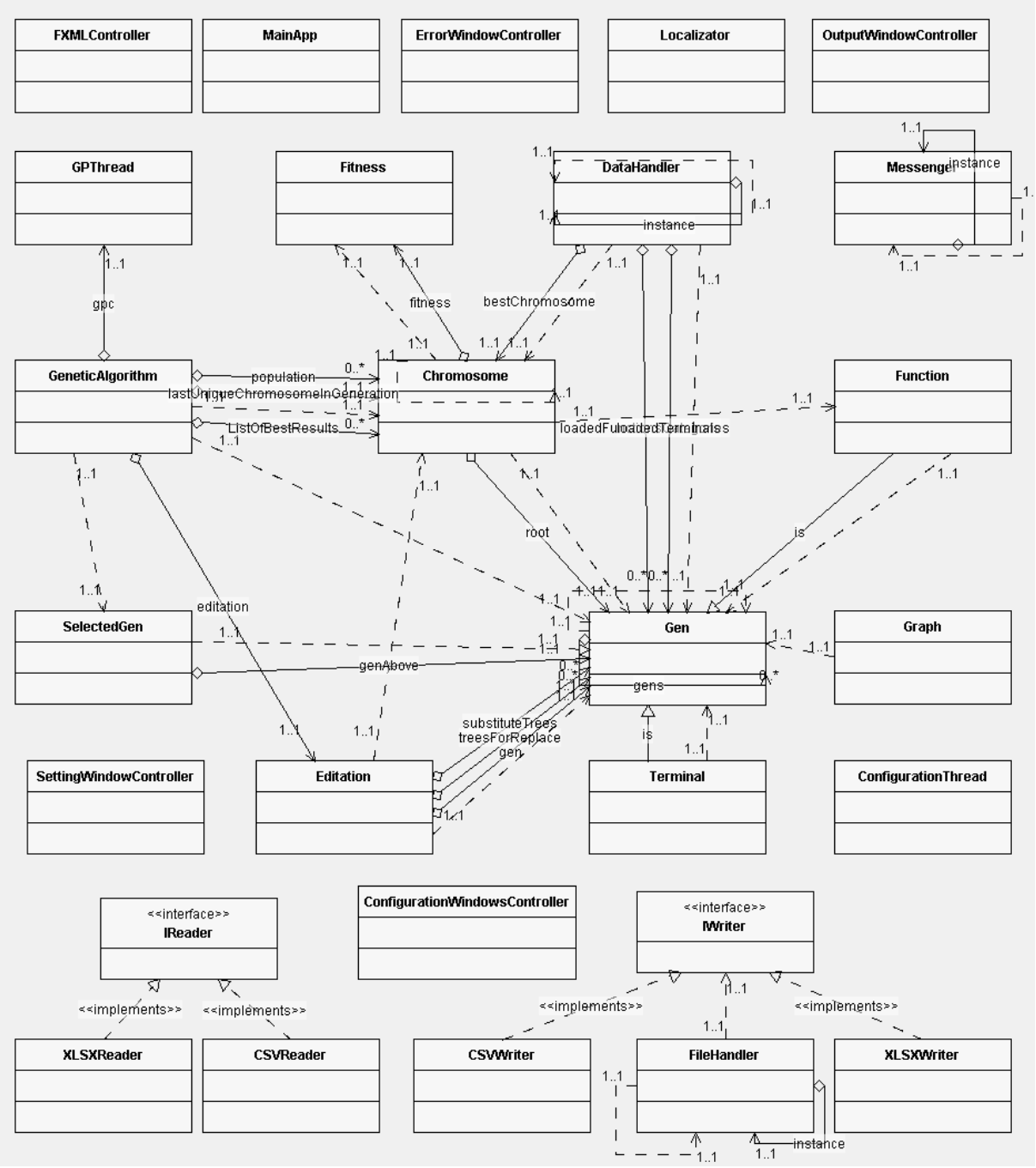
FilenameUtils je open source utilita na práci se soubory a s cestami k soborům. Například poskytuje jednodušší práci se zjišťováním typu souborů.

XSSF je open source projekt pro otevírání, čtení, ukládání a zavírání Excel souborů.

Apache FOP je open source formátovač tisku řízený formátovacími objekty typu XSL.

E. Diagram tříd

Následovný diagram popisuje strukturu programu *Symbolic Regression*.



Obrázek 23: Diagram tříd

F. Obsah přiloženého CD

`src` – zdrojové kódy programu

`app` – samotný .jar a potřebné soubory ke spuštění

`examples` – ukázky výstupů konfigurací a výstupních protokolů

`test data` – testovací data

`text` – obsahuje elektronickou verzi textu této práce

`uml` – uml diagramy ve vyšším rozlišení