Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Evoluční algoritmy a jejich aplikace

Bakalářská práce

Autor: Lukáš Syrový
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.

Hradec Králové srpen 2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 16.8.2017 Lukáš Syrový

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Barboře Tesařové, Ph.D. za metodické vedení a odborné rady při vypracování bakalářské práce.

Anotace

Bakalářská práce se věnuje algoritmům napodobujících evoluční procesy v přírodě – evolučním algoritmům a jejich využitím v reálném světě. Konkrétně práce popisuje gramatickou evoluci, která do kategorie evolučních algoritmů spadá, problematikou aproximace funkcí, aplikaci algoritmů na analýzu vodních průtoků, implementaci algoritmu a vytvoření programu v jazyce Java. Vytvořený program by měl být použitelný nejen pro analýzu vodních průtoku ale i pro jiné problémy, kde je nutná aproximace funkcí.

Annotation

Title: Evolutionary algorithms and their application

Bachelor thesis deals with algorithms simulating evolutionary processes in nature - evolutionary algorithms and their usage in the real world. Specifically, the thesis describes grammatical evolution, which belongs to the category of evolutionary algorithms, problems of approximation of functions, application of algorithms for water flow analysis, algorithm implementation and creation of Java program. The created program should be useful not only for water flow analysis but also for other problems where function approximation is required.

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

Při řešení optimalizačních úloh, kdy je třeba prohledávat velké stavové prostory, jsou tradiční matematické postupy časově neefektivní, z tohoto důvodu jsou tyto postupy nahrazovány tzv. heuristickými metodami, které generují řešení, která mají být blízká řešení optimálnímu.

Evoluční algoritmy jsou metodou, často využívanou pro řešení nelineárních regresních úloh tam, kde jiné postupy selhaly a nebyly schopny dodat přijatelné výsledky. Jedním z nejmladších rozšíření evolučních algoritmů je gramatická evoluce, jejíž hlavní výhodou je univerzálnost – díky gramatice zapsané pomocí Backus-Naurovy formy.

Tato práce je rozdělena na 4 části, kde první kapitolou je úvod do řešené problematiky a poslední kapitolou je ohlédnutí na vytvořenou práci a rekapitulace dosažených výsledků. Druhá část se zabývá teorií, vysvětluje základní principy evolučních algoritmů, především gramatickou evoluci, dále pak popisuje symbolickou regresi. Třetí část popisuje vytváření aplikace v jazyce Java, testování této aplikace a zhodnocení výsledků.

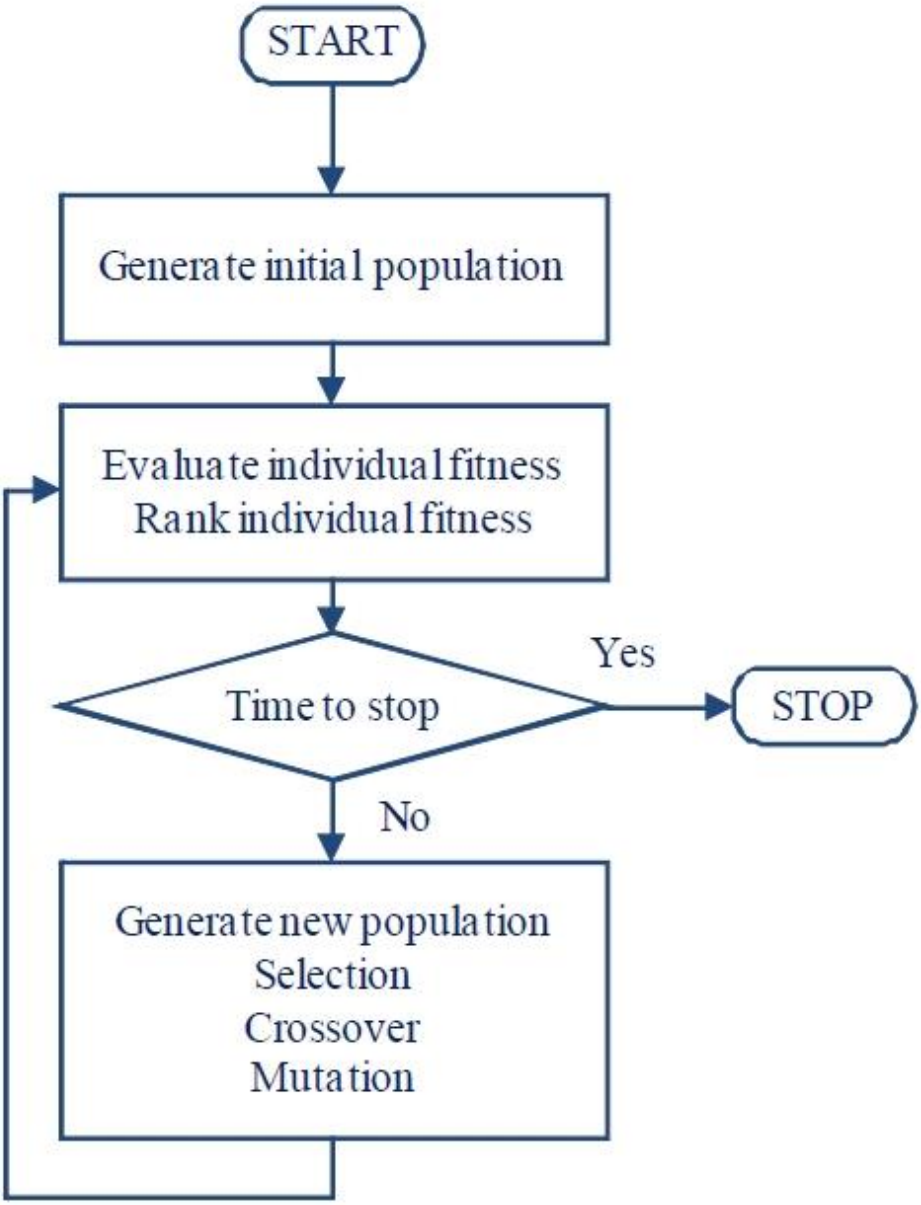
2 Teoretická část

V této kapitole jsou popsány základní principy evolučních algoritmů, jejich podtřídy genetické programování, a především gramatická evoluce. Dále jsou zde vysvětleny možné problémy evolučních algoritmů a nastíněna jejich potenciální řešení. Na konci kapitoly bude ještě představena problematika symbolické regrese a aproximace funkcí.

2.1 Principy evolučních algoritmů

Historie evolučních algoritmů (dále jen EA) sahá do 60. let minulého století, kdy se John Holland inspiroval přírodními procesy selekce a Darwinovou teorií evoluce a navrhl algoritmus pro řešení optimalizačních úloh, který napodobuje právě tyto procesy. (1) Tento algoritmus byl později znovuobjeven, dále rozveden a popsán Davidem E. Goldbergem a vznikl tak jednoduchý genetický algoritmus (2). Od té doby bylo navrženo mnoho modifikací evolučních algoritmů, základní princip však zůstal stejný.

EA pracuje s množinami jedinců o velikosti n, tyto množiny jsou v daném kontextu označovány za populace, v průběhu algoritmu jsou pak vytvářeny nové populace. Každý jedinec reprezentuje potenciální řešení daného problému, tito jedinci mají určitou šanci na přežití, která je stanovena pomocí tzv. fitness funkce, přežitím se zde rozumí výběr jedince pro reprodukci. Pomocí metody selekce, která na základě fitness hodnot vybere jedince pro aplikaci operátorů mutace a křížení. Tyto operátory získají na vstupu jednoho nebo více jedinců a jejich výstupem je stejný počet potomků pro novou populaci. Operátory pracují, dokud není naplněn počet jedinců do nové populace. Takto se opakují kroky ohodnocení/selekce/vytvoření nové populace až do splnění ukončující podmínky – nejčastěji dosažení stanoveného počtu generací, případně nalezení dostatečně kvalitního řešení. Průběh algoritmu je zobrazen na obrázku 1.



Obrázek 1: Průběh evolučního algoritmu, zdroj: (3)

2.1.1 Reprezentace jedinců

Jedinci jsou statické objekty, které po dobu životnosti nemění svoji strukturu, jejich úloha v rámci EA je reprezentace potenciálních řešení hledaného problému. (2) Nejčastěji jsou jedinci uloženi v podobě binárních řetězců, nicméně existují takové implementace EA, které využívají jiné struktury – např. stromovou strukturu, u jednoduchých genetických algoritmů mají všichni jedinci stejný počet bitů, některé variace EA však dokáží pracovat s jedinci rozdílné velikosti. V těchto řetězcích jsou zakódovány data potenciálních řešení, jelikož binární řetězec pravděpodobně není

požadovaný výstup programu. Z tohoto důvodu jsou také používány termíny genotyp, fenotyp a chromozom. Genotyp je genetická informace v ryzím formátu tak, jak je uložena v paměti – nejčastěji tedy binární řetězec, chromozom, v kontextu EA má stejný význam jako genotyp. Chromozom je konkrétní řešení. Pro funkčnost programu je tedy klíčové vymyslet vhodný mechanismus mapování genotypu na fenotyp, jedno z možných řešení nabízí gramatická evoluce (viz kapitola 2.2).

2.1.2 Fitness

Jedná se o číselnou hodnotu, která určuje kvalitu jedince a je stanovena pomocí tzv. fitness funkce, funkce, která musí být navržena pro řešený problém. Cílem evolučního algoritmu je tuto hodnotu minimalizovat/maximalizovat (podle toho, zda lepší jedinci = nižší fitness, nebo naopak) Pro dosažení kvalitních výsledků pomocí EA je vhodně navržená fitness funkce klíčová. (2)

2.1.3 Selekce

Úkolem selekce je výběr vhodných jedinců, kteří se mají stát rodiči, stejně jako v přírodě, kvalitnější jedinci (určeno pomocí fitness hodnoty) mají větší šanci na výběr, méně kvalitní jedinci mají stále šanci na postup díky přítomnosti prvku náhody. Nejčastěji používané metody selekce jsou turnajová a ruletová selekce, případě některé z jejich modifikací. Vybranou metodu selekce lze také rozšířit pomocí elitismu. (3)

Turnajová selekce

Nejprve je vytvořeno *m* skupin o *n* počtu jedinců (nejčastěji 2 jedinci), kteří jsou vybíráni zcela náhodně, následně je z každé skupiny vybrán nejsilnější jedinec. Výhodou turnajové selekce je její nízká výpočetní náročnost proti selekci ruletové. Nevýhodou je možnost situace, kdy při tvoření skupin budou vybráni pouze méně kvalitní jedinci (protože při tvoření skupin jsou vybíráni zcela náhodně) a kvalitní jedinci tedy nepostoupí do fáze reprodukce.

Ruletová selekce

U ruletové selekce je šance na výběr jedince přímo úměrná jeho fitness hodnotě, ruletovou selekci lze nejlépe popsat jako ruletové kolo, kde každý jedinec zabírá jeden výseč kola, na rozdíl od klasické rulety však nemají jednotlivé výseče stejnou velikost – jejich velikost odpovídá jejich fitness hodnotě. Větší výseče (lepší řešení) mají větší šanci, že se kolo zastaví právě u nich.

Pořadová selekce

Jedná se o modifikaci ruletové selekce, u které nastává problém v případě, kdy jednotlivé ohodnocení se liší enormně, např. v situaci, kdy jedno individuum má hodnotu 90 % součtu fitness hodnot všech jedinců, zbývající řešení si rozdělí pouze 10 % „hracího kola“, v takovém případě by byla šance na výběr ostatních jedinců velmi malá. Pořadová selekce přidělí každému jedinci pořadí od 1 do n tak, aby nejslabší jedinec měl hodnotu 1, nejsilnější hodnotu n a jedinci se stejnou kvalitou pořadí shodné. Následně se pokračuje stejně jako u ruletové selekce, pouze hodnoty fitness jsou nahrazeny pořadím. Porovnání ruletové a pořadové selekce je znázorněno na obrázku 2.

Obrázek 2: Porovnání ruletové selekce (vlevo) a pořadové selekce (vpravo), zdroj: autor

Elitismus

Nově vytvářená populace může, ale také nemusí obsahovat nejlepší jedince z předchozí generace, případně budou vybrány pro reprodukci, ale kvalita jejich potomků bude nižší, v rámci optimalizace, ztráta nejlepších jedinců nedává smysl. Z tohoto důvodu, Kenneth Alan De Jong navrhl koncept elitismu (3). Elitismus zajistí, aby předem určený počet nejlepších řešení vždy postoupil do příští generace bez jakékoliv modifikace.

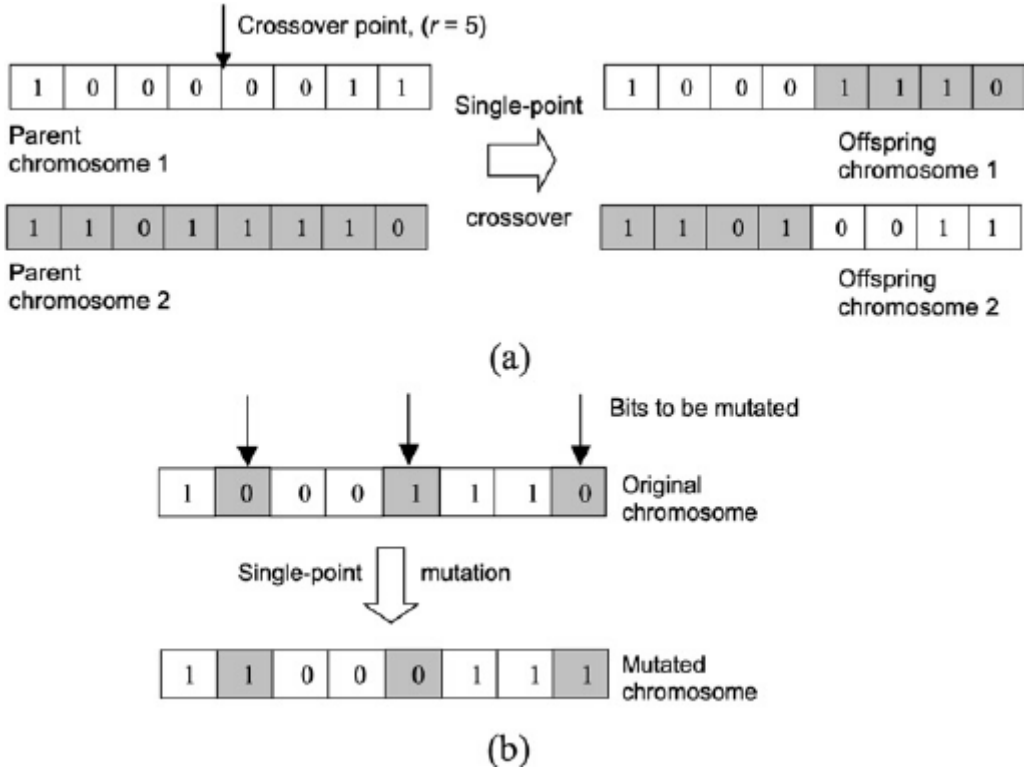
2.1.4 Operátory křížení a mutace

Po zvolení jedinců jsou pomocí jejich genotypů vytvořeny potomci pro příští generaci, k tomu slouží především operátor křížení, který na vstupu získá 2 jedince v roli rodičů a z nich vytvoří 2 potomky, a mutace – jednoho jedince zmutuje na jednoho potomka.

Princip křížení spočívá v „rozpůlení“ chromozomu rodičů na určitém indexu a následné výměně části chromozomu mezi rodiči – tím vznikají 2 nové genotypy. Před zkřížením je ještě spočítána pravděpodobnost mutace která určí, zda bude pár jedinců zkřížen, nebo ponechán v původním stavu.

Mutace invertuje gen na určitém indexu a vzniká tak nový potomek, v případě že se tedy jedná o binární řetězec, metoda invertuje vybranou 1. nebo 0. Mutace může být také vícebodová – invertuje 2 nebo více genů na náhodných indexech.

Kdy bude upřednostněna mutace před křížením stanovuje procento pravděpodobnosti mutace – obvykle se jedná o velmi nízkou hodnotu (0–3 %), většina potomků totiž musí být vytvořena křížením.



Obrázek 3: Křížení(a) a mutace(b), zdroj: (6)

2.1.5 Problém uvíznutí v lokálním optimu

V průběhu algoritmu může nastat situace, kdy algoritmus uvízne v lokálním optimu, které může být velmi odlišné od optima globálního (4). K takové situaci dochází zejména v pozdější iteraci populace, kdy většina jedinců jsou předky jen malého počtu jedinců z prvotní generace.

Pro minimalizaci šance na uvíznutí je klíčová vysoká diverzita v rámci populace, na kterou má vliv především selekce (5) (viz kapitola 2.1.3). Mezi nejčastěji používané metody selekce patří selekce ruletová, ovšem v případě, kdy se razantně liší ohodnocení jednotlivých řešení je vhodné zvolit její modifikaci – pořadovou selekci, která v těchto případech minimalizuje riziko uvíznutí v budoucích generacích. Dále, pokud je používán, snížení hodnoty elitismu může pomoci.

Mezi další možné řešení patří např. navýšení pravděpodobnosti mutace, velmi efektivním postupem bývá proměnlivá hodnota pravděpodobnosti mutace – pokud se již delší dobu nezlepšila nejlepší fitness – program navýší pravděpodobnost mutace a opět mutace sníží, pokud nalezne lepší fitness. Posledním řešením je vytváření malého počtu úplně nových jedinců pro každou generaci.

2.2 Genetické programování

Před představením gramatické evoluce je ještě nutné objasnit genetické programování (dále jen GP). Historie GP sahá do 90. let minulého století, kdy John R. Koza položil základy GP (6) a definoval případy pro ověření jejich funkčnosti. Přestože se první teoretické náznaky GP vyskytovaly již dříve, za zakladatele je považován právě John R. Koza. Díky jiné struktuře jedinců (oprati binárním řetězcům) je v dnešní době GP používáno pro řešení složitých problémů – GP je schopné tvořit celou programovou strukturu, zatímco jednoduché genetické algoritmy jsou používány spíše pro hledání jednoduchých řešení – např. hledání vhodné konstanty.

2.2.1 Reprezentace

Největším rozdílem GP od genetických algoritmů je reprezentace jedinců, na rozdíl od binárních řetězců jsou tvořeni stromovou strukturou, (7) ve které vnitřní

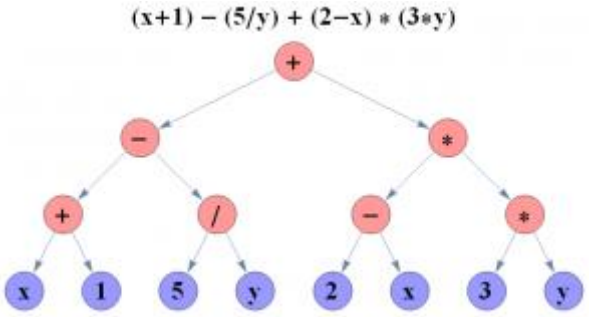
uzly představují funkce a koncové uzly terminály. Taková struktura má jednu značnou výhodu – jedinci mohou nabývat různých velikosti – nemají pevně danou velikost. Jedinci v GP jsou znázorněny na obrázcích 4 a 5.

2.2.2 Vytvoření počáteční populace

Pro dosažení uspokojivých výsledků pomocí GP je nutno vytvořit velmi rozsáhlou populaci (v řádech tisíců). Pro vytvoření počáteční populace existují 2 metody – úplná a růstová. Běžně jsou pro vytvoření populace využity obě metody v poměru 50/50. (4)

Úplná metoda

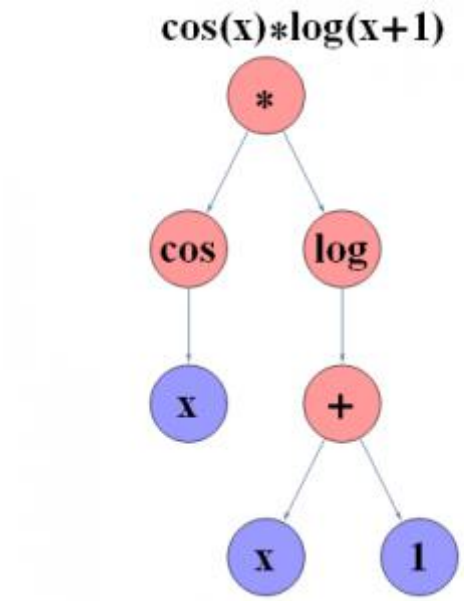
Tato metoda vytváří strom tak, že všechny koncové uzly jsou ve stejné hloubce v rámci jednoho stromu, neznamená tedy, že všichni jedinci tvořeni touto metodou mají stejnou hloubku.



Obrázek 4: Strom vytvořený úplnou metodou, zdroj: (10)

Růstová metoda

U této metody je nejprve stanovena maximální hloubka, následně jsou pro jednotlivé uzly zcela náhodně vybírány terminály nebo funkce z příslušných množin. V případě, že určitý uzel ještě nedosáhl maximální hloubky a byl vybrán terminál, příslušná větev je hotova. Pokud již uzel dosáhl maximální hloubky, vybírá se pouze z množiny terminálů.

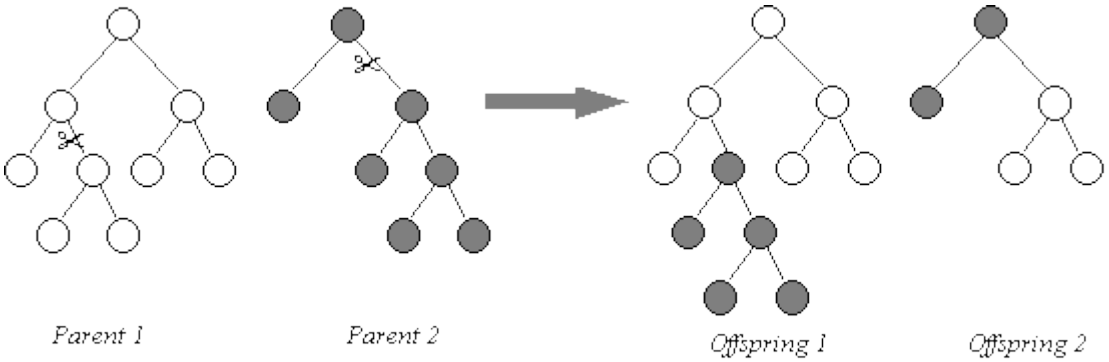


Obrázek 5: Strom vytvořený růstovou metodou, zdroj: (10)

2.2.3 Operátory křížení a mutace v GP

Protože je reprezentace jedinců v GP rozdílná, je logické, že také manipulace s jedinci bude jiná, to platí především u operátorů, kteří v GP využívají metod pro práci s podstromy.

Při křížení dvou rodičů je pro každého jedince náhodně vybrán uzel a podstromy, které na tyto uzly navazují jsou mezi rodiči zaměněny a vznikají tak potomci.



Obrázek 6: Křížení v genetickém programování, zdroj: (11)

Mutace vybere podstrom z náhodného uzlu a tento podstrom nahradí jiným, náhodně vytvořeným podstromem.

2.2.4 Problém narůstající velikosti při vytváření nových populací

Protože GP dokáže pracovat s jedinci různé délky a struktury, hrozí zde riziko nadměrného nárůstu délky jedinců v průběhu vytváření nových generací, tento jev

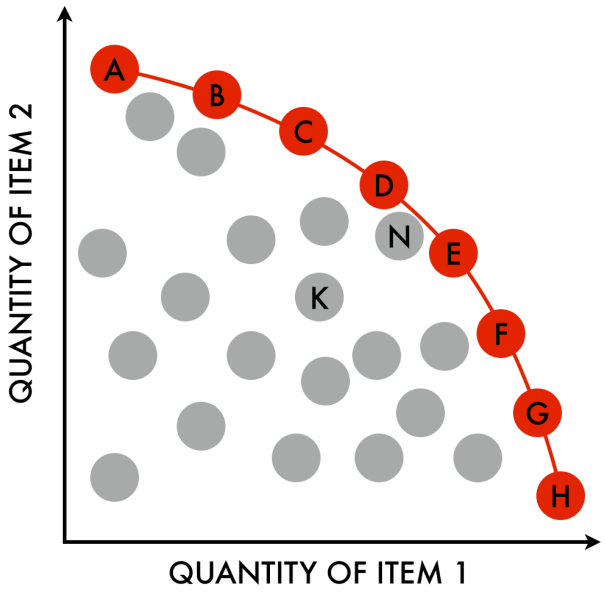
se je známý jako bloat efekt. (6) Tito nadměrně velcí jedinci již nepředstavují efektivní řešení, a navíc zpomalují průběh programu. Při implementaci by tedy mělo být zamezeno vzniku těchto řešení, pro to existují 2 postupy, které je vhodné kombinovat.

Prvním opatřením je penalizace dlouhých řešení, tedy zohledňování velikosti stromu ve fitness funkci – přičítat (případně odečítat, záleží, jak jsou nejlepší/nejhorší jedinci ohodnocováni) velikost uzlů vynásobenou stanovenou konstantou k fitness hodnotě jedince. Zde je nutné pečlivě zvážit hodnotu konstanty – zbytečné vysoká hodnota vede ke znehodnocení, případně ztrátě velmi blízkých řešení, s nízkou hodnotou jsou zase prosazována zbytečně velká řešení. Problém délky řešení x přesnosti řešení je možné popsat pomocí Paretova optima, které je popsáno v následující kapitole.

Druhým opatřením je stanovení maximální velikosti stromu a úprava operátorů tak, aby se řídily touto hodnotou. Toto opatření se týká především operátoru křížení, který vybírá náhodný uzel pro výměnu podstromu, uzly na obou rodičích musí být vybrány v takové hloubce, aby hloubka potomků nepřekročila stanovenou hodnotu.

2.2.5 Paretovo optimum

Paretovo optimum je předpis popisující vzájemný vztah dvou kritérií popsaný v roce 1897 Vilfredem Paretem (8). Paretovo optimum je vztah, ve kterém již nelze z (9)výšit efektivita jednoho kritéria bez toho, aby nebyla snížena efektivita kritéria druhého, V množině se nachází více optim, kde optima na hranici jsou přijatelné výsledky a vše pod touto hranicí jsou výsledky nepřijatelné. V kontextu genetického programování jsou kritéria přesnost řešení a velikost tohoto řešení, na uživateli je pak výběr, zda chce kratší/méně přesné nebo delší/přesnější řešení.



Obrázek 7: Paretovo optimum a zavislost dvou kritérií, červeně označené body jsou optimální výsledky, body pod hranicí (šedé) jsou nepřijatelné, zdroj: (14)

2.3 Gramatická evoluce

Jedná se o jednu z nejnovějších metod spadajících pod evoluční algoritmy, představenou Conorem Ryanem a jeho kolegy v roce 1998 (10). Gramatická evoluce v ryzí formě je vlastně genetické programování rozšířené o gramatiku a dokáže tak vytvářet programy v libovolném jazyce popsaném Backus-Naurovou formou. S tím je také úzce spojena reprezentace jedinců, na rozdíl od stromů v genetickém programování používá gramatická evoluce binární řetězce pro reprezentaci jedinců (11).

2.3.1 Backus-Naurova forma

Backus-Naurova forma (BNF) je typ bezkontextové gramatiky, která se používá pro zápis formálních jazyků. BNF je definována formou pravidel zapsaných jako množina terminálů a neterminálů, kteří jsou dále rozvíjeny (na jeden nebo více terminálů či neterminálů). Zápis pravidel má pevnou strukturu – na levé straně vystupují jednotlivé neterminály a na pravé straně terminály a neterminály pro rozvoj (11). Zápis BNF vypadá tedy takto:

<symbol> ::= <výraz se symboly>

V případě možnosti výběru z výrazu se symboly jsou jednotlivá pravidla oddělené svislou čarou.

```
<exp> ::= <val>|<exp><op><exp>
<val> ::= x|<num>
<num> ::= 1|2|3|4|5
<op> ::= -|+|*|/
```

Obrázek 8: Zápis matematických výrazů v BNF, zdroj: autor

2.3.2 Mapování genotypu na fenotyp

Při dekódování lineárního chromozomu je postupně čten binární řetězec zleva po částech stejné velikosti – kodonech. Kodony jsou sekvence bitů o pevně dané velikosti (nejčastěji 8 bitů) tvořící chromozom, každý kodon reprezentuje je dekódován na jedno pravidlo. Každý kodon je převeden do desítkové soustavy a vzniká tak hodnota c, pomocí této hodnoty bude přečteno pravidlo podle předpisu:

$$\text{Pravidlo} = c\%r$$

Kde *r* značí počet možností výběru v množině výrazu se symboly (vpravo) na daném řádku, pokud je čten první kodon, začíná se od prvního řádku. Hodnota zbytku po dělení značí index terminálu či neterminálu, v případě, že byl vybrán neterminál, mapování pokračuje dalším kodonem na řádku, který odpovídá přečtenému neterminálu (12).

Ukázka mapovacího procesu

Pro ukázku mapovacího procesu zvažujme gramatiku zobrazenou na obrázku 4 a následující chromozom:

00000001.00100000.00010001.00000010.00000100.00000101.01000000

Převedený na hodnoty:

1, 32, 17, 2, 4, 5, 64

Začínáme tedy hodnotou 1 a prvním řádkem, který nabízí 2 pravidla - <val> a <exp><op><exp>, index získaný zbytkem po dělení je 1, kodon je tedy nahrazen pravidlem <exp><op><exp>, to znamená další 3 neterminály pro rozvedení. Následující hodnota je 32 a počet pravidel je opět 2 (rozvádíme neterminál <exp>, zůstáváme tedy na stejném řádku), získáváme tak další neterminál k rozvinutí - <val> a posouváme se na druhý řádek a pomocí hodnoty 17 získáváme terminál x.

Protože jsme se dostali až k terminálu, pokračujeme dalším neterminálem získaným z prvního kodonu - <op>. Po přečtení všech kodonů by měla vzniknout následující funkce:

x+5

V tomto případě byla použita velikost kodonu 8 bitů, rozsah 0 – 255, ovšem nejvyšší počet pravidel neterminálů je 5 (<num>), stejně by tedy posloužila i menší velikost kodonu, což by vedlo k nižší zátěži výpočetního výkonu.

Situace, které mohou nastat během procesu mapování

Během procesu dekódování chromozomu nastane jedná ze tří siduaci:

1. Byly přečteny všechny kodony a převedeny na příslušné terminály – všechny neterminály byly rozvedeny
2. Nebyly přečteny všechny kodony, neterminály však byly rozvedeny
3. Byly přečteny všechny kodony, zbývají však neterminály k rozvedení

První případ je ideální, byly přečteny všechny kodony a byl získán validní fenotyp. V druhém případě zbývají ještě kodony k přečtení, nicméně všechny neterminály byly rozvedeny, v takovém případě jsou zbývající kodony ignorovány – stále však můžou tyto kodony využít potomci daného jedince.
Ve třetím se je obvyklý postup, po přečtení všech kodonů, začít číst kodony znovu zleva, zde však hrozí riziko zacyklení, kdy ani po opakovaném přečtení všech kodonů nebyly stále rozvedeny všechny neterminály. Z tohoto důvodu je vhodné stanovit maximální počet opakování čtení, kdy po dosažení tohoto počtu opakování je jedinec označen jako neplatný a je mu přiřazena nejhorší možná hodnota fitness.

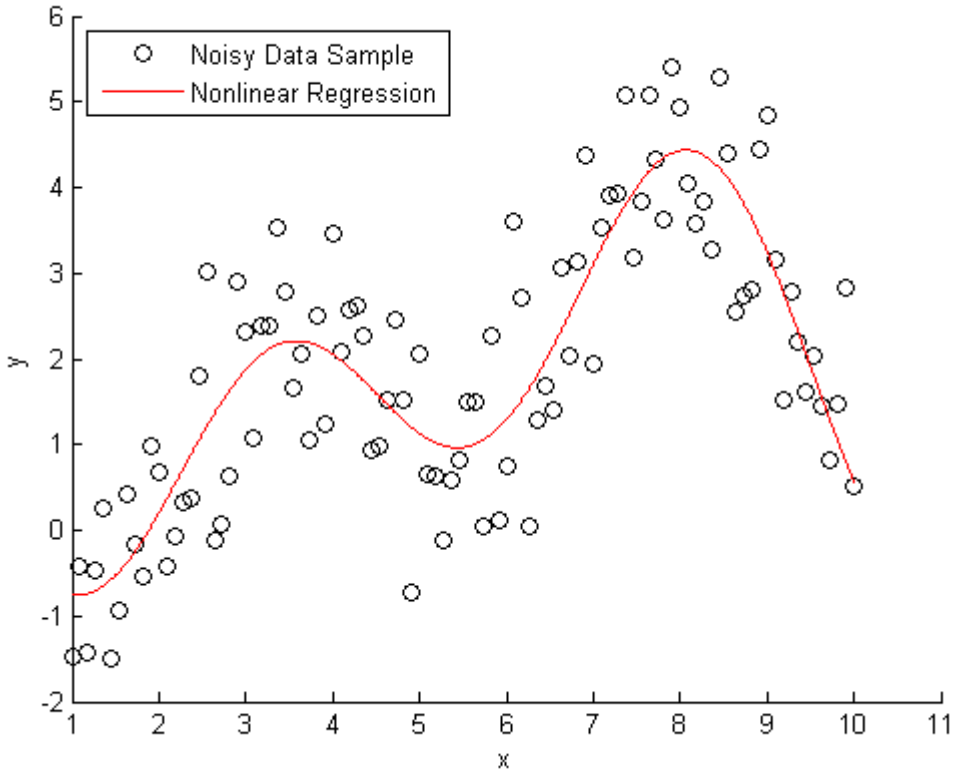
2.3.3 Optimalizace Gramatické evoluce

Stejně jako genetické programování má GE nespornou výhodu možnost práce s jedinci s proměnnou velikostí chromozomů, má také s tím spjatou nevýhodu – trpí bloat efektem. (13) Bloat a možnosti, jak mu předejít jsou popsány v kapitole 2.2.4. Za další optimalizaci by se dalo požadovat vhodně určená velikost kodonu, ne pro každý problém je potřeba velikost 8, jak už bylo naznačeno v ukázce mapovacího

procesu. Zvolená velikost však musí být dostatečně velká aby bylo možné vybrat libovolné pravidlo z gramatiky, a aby vybraná pravidla neměla značně větší šanci na výběr oproti ostatním pravidlům.

2.4 Symbolická regrese

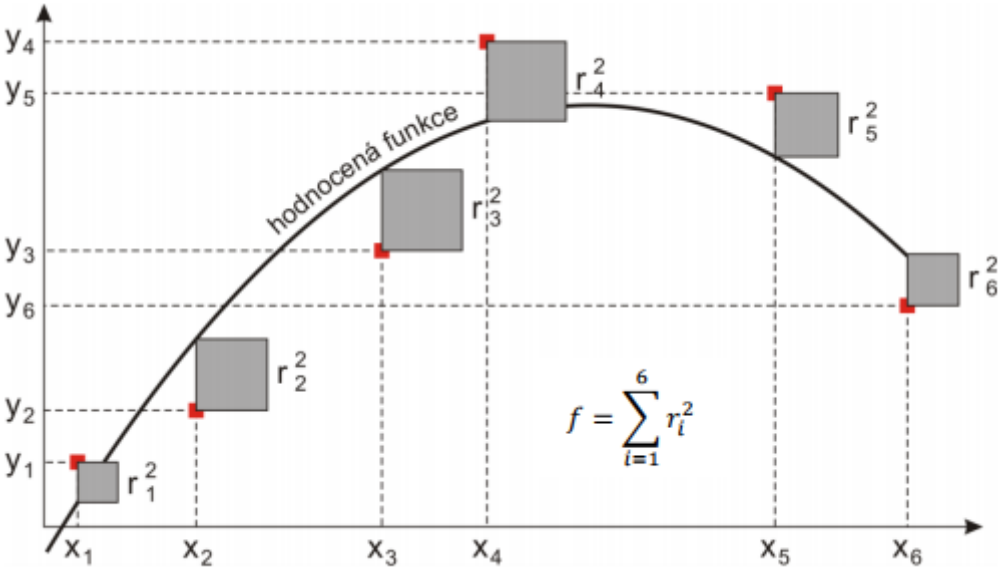
Po dlouhou dobu byla aproximace funkcí doména řešena pouze lidmi, v posledních několika desetiletích je však přenášena a řešena také pomocí výpočetní techniky – díky evolučním algoritmům, a především symbolické regresi. Symbolická regrese je úloha, jejíž cílem je identifikace matematického popisu skryté závislosti experimentálně získaných dat – hledá funkci, která nejlépe aproximuje data. (9) Pro dosažení požadovaného výsledku symbolická regrese náhodně kombinuje stavební bloky v podobě operátorů, funkcí, proměnných nebo konstant, pro vytváření nových kombinací využívá postupy genetického programování. Je velmi pravděpodobné, že se do budoucna zvýší důležitost symbolické regrese vzhledem k rostoucí komplexitě problémů řešených ve vědě a průmyslu. (14) Fitness hodnota je zde vypočítána podle velikosti odchylek – součet kvadratických odchylek a také podle složitosti navrhovaného řešení. Protože je gramatická evoluce velmi blízká genetickému programování, je možné používat symbolickou regresi tako pomocí gramatické evoluce.



Obrázek 9: Aproximace funkce pomocí symbolické regrese, zdroj: (20)

2.4.1 Součet kvadratických odchylek

Pro každý definovaný bod [x, y] se počítá druhá mocnina odchylky od zadané funkční hodnoty funkce *f(x)*, tato kvadratická odchylka se též nazývá reziduum. (11) Hodnotou účelové funkce je tedy součet čtverců – kvadratických odchylek.



Obrázek 10: Princip kvadratických odchylek, zdroj: (16)

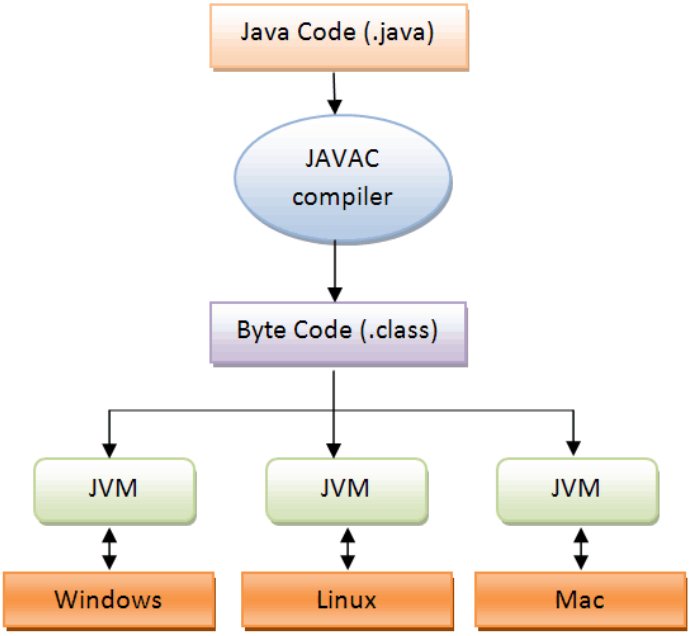
3 Praktická část

Tato kapitola popisuje strukturu vytvořeného programu v jazyce Java, jednotlivé třídy a jejich metody a použité technologie. Dále je v této kapitole popsán proces testování a prezentovány výsledky testování programu na vybraných reálných funkcích. Na konci kapizoly je popsáno využití programu pro aproximaci funkcí na datech vodních průtoků.

3.1 Tvorba a popis vytvořené aplikace

3.1.1 Jazyk Java

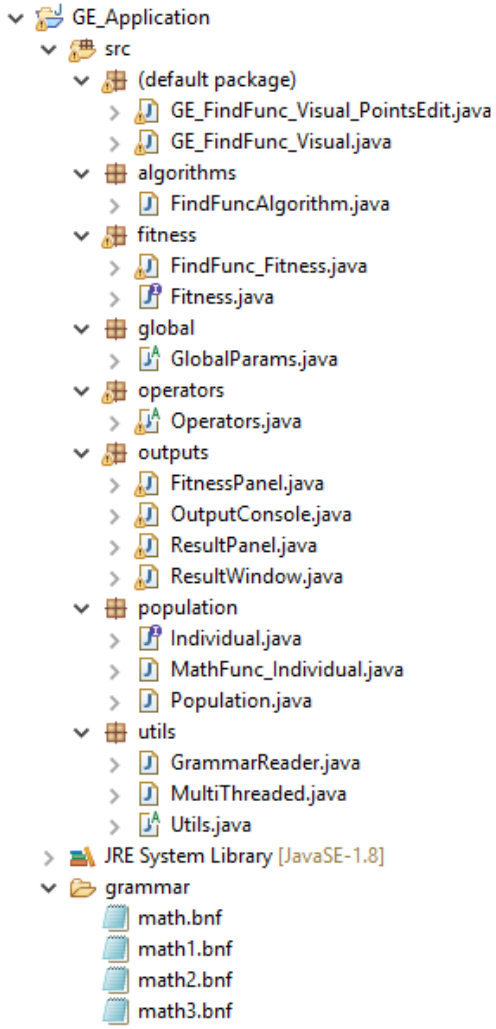
Java je univerzální, paralelní a oběktově-orientovaný programovací jazyk, (15) navržený tak, aby byl jednoduše srozumitelný pro většinu programátorů, tomu pomáhá i syntaxe podobná jazykům C++ a C#. Účelem jazyka je umožnit programátorům psát program tak, aby bylo možné spouštět program na libovolné platformě bez opakovaného kompilování. Pro spuštění programu na vybrané platformě je nutnývirtuální stroj pro danou platformu – Java Virtual Machine (JVM). Protože je Java tzv. „high-level“ jazyk, tedy jazyk s vyšší mírou abstrakce, tedy jazyk, v jehož syntaxi se nezadávají instrukce přímo pro hardware, java zahrnuje automatickou správu paměti – garbage collector, (15) který určitý objekt v paměti odstraní ve chvíli, kdy žádný jiný objekt v paměti tento objekt nereferencuje.



Obrázek 11: Architektura jazyka Java, zdroj: (22)

3.1.2 Implementace

Cílem aplikace je aproximace funkcí, nicméně je navržena tak, aby ji bylo možné použít i pro jiné úlohy, k čemuž je třeba pouze vhodně zapsaná gramatika a fitness funkce pro daný problém. Třídy, které pracují s operátory, selekcí a různé podpůrné třídy tedy nejsou vázány pouze na problém aproximace funkcí.



Obrázek 12: Struktura vytvořené aplikace, zdroj: autor

Třída global.GlobalParams

Třída GlobalParams ukládá klíčové parametry pro běh evolučního algoritmu, jedná se o abstraktní třídu (nevytváří potomky), která obsahuje veřejné statické atributy. Veškeré atributy mají nastavené defaultní hodnoty.

Tabulka 1: Atributy třídy GlobalParams

int populationSize	Stanoví velikost populace.
String grammarFilePath	Určuje cestu k souboru se zapsanou gramatikou.
int codonSize	Velikost kodonu.
double highestMutationPropability	Maximální pravděpodobnost mutace – v případě, že došlo v průběhu algoritmu

	k uvíznutí v lokálním optimu, je použita tato hodnota.
double baseMutationPropability	Pravděpodobnost mutace, ke které se algoritmus vrací, v případě, že uniknul z lokálního optima.
double mutationPropability	Základní pravděpodobnost mutace, na které algoritmus začíná.
Double crossoverPropability	Pravděpodobnost, zda bude dvojice jedinců zkřížena.
int numberOfGenerations	Maximální počet generací, po jeho dosažení je algoritmus ukončen.
int elitism	Hodnota elitismu.
int maxIndSize	Maximální možný počet kodonů jednoho jedince – zamezuje BLOAT.

Třída algorithms.FindFuncAlgorithm

Pro řešení problematiky aproximace funkcí je třída FindFuncAlgorithm nejdůležitější třídou v programu. Mimo gettery a settery obsahuje pouze metodu *execute,* která je volána z tříd GE_FindFunc nebo GE_FindFunc_Visual.

Protože metoda *execute* reprezentuje celý proces evolučního algoritmu, její popis je téměř identický popisu průběhu evolučního algoritmu. V průběhu této metody je volána většina meto, které budou popsány níže, algoritmus používá nastavení z třídy GlobalParams a v případě volání metod s více přetíženími volá tedy tu implementaci, která má minimum vstupních parametrů. Obsahuje 2 ukončovací podmínky, a to ukončení při dosažení maximálního počtu generací, nebo může být ukončena již dříve, že bylo nalezeno řešení s dostatečně nízkým fitness.

Součástí algoritmu této metody je také mechanismus pro navýšení pravděpodobnosti mutace v případě, že po zpracování daného počtu generací zůstává fitness hodnota stále stejná. Pokud je po navýšení pravděpodobnosti mutace je nalezeno lepší řešení, pravděpodobnost je vrácena na její původní hodnotu.

Tabulka 2: Popis ostatních tříd

population.MathFunc_Individual	
Tato třída reprezentuje jedince, implementuje rozhrání Individual a Comparable.	
MathFunc_Individual (String genome)	Konstruktor třídy, vstupním parametrem je chromozom uložený v textovém řetězci. V případě, že řetězec obsahuje jiné než povolené znaky (0-1), konstruktor vyvolá vyjímku.
String getGenome ()	Vrací chromozom v textovém řetězci.
void setFitness (double fitness)	Přiřazuje jedinci hodnotu fitness.
double getFitness ()	Vrací double hodnotu fitness.
int compareTo (Individual i)	Tato metoda porovná daného jedince s jinou instancí (vstupní parametr) na základě fitness, vrací hodnoty 1–v případě, že má lepší fitness jedinec, na kterém je metoda volána, nebo -1, pokud má lepší fitness jedinec v parametru metody.
population.Population	
V instancích této třídy jsou seskupeni všichni jedinci v rámci jedné generace. Dále je možné instanci přiřadit číslo generace a součet fitness hodnot všech jedinců.	
Population ()	Konstruktory třídy, v případě, že není nastaven vstupní parametr číslo generace, je nastaveno číslo generace na 0, pokud není nastaven parametr list jedinců, je vytvořen prázdný list v populaci.
Population (int generation)	
Population (List<Individual> individuals)	
Population (int generation, List<Individual> individuals)	
int getGeneration ()	Vrací číslo generace.
void setGeneration (int generation)	Nastavuje číslo generace.
List<Individual> getIndividuals ()	Vrací list jedinců v populaci

void setIndividuals (List<Individual> individuals)	Nastavuje list jedinců.
double getSumFitness ()	Vrací součet fitness hodnot všech jedinců v populaci.
void setSumFitness (double sumFitness)	Nastavuje součet fitness všech jedinců v populaci.
fitness.FindFunc_Fitness	
Třída implementuje rozhrání Fitness, které předepisuje hlavičky dvou metod – setFitnessValue a setFitnessValuetoPopulation. Dále obsahuje vnitřní třídu RunnableEval, která umožňuje paralelně ohodnocovat více jedinců.	
FindFunc_Fitness (FindFuncAlgorithm alg)	Konstruktor třídy, vstupním parametrem je instance třídy FindFuncAlgorithm, od kterého přebírá některá data.
void setFitnessValue (Individual individual)	Metoda nastavuje fitness hodnotu jednomu konkrétnímu jedinci. Nejprve je převeden chromozom na matematický předpis, následně je volána metoda computeValue z třídy RunnableEval, která pomocí předpisu počítá hodnoty y ze vstupních hodnot x.
setFitnessValuetoPopulation (Population p)	Metoda nastavuje fitness všem jedincům vstupní populace, nejprve danou populaci přiřadí třídě RunnableEval, následně volá z této třídy metodu run.
fitness.FindFunc_Fitness. RunnableEval	
Vnitřní třída třídy FindFunc_Fitness, dědí ze třídy MultiThreaded a v případě zpracování celé populace dokáže paralelně zpracovávat více jedinců.	
double computeValue(double x, String function)	Metoda má vstupní parametr hodnotu x, pro kterou má vypočítat hodnotu y a matematickou funkci zapsanou v textovém řetězci. Pomocí

	instance třídy ScriptEngine a její metody eval vrací hodnotu y.
String replaceX (double x, String function)	Tato metoda prohledá funkci zapsanou v textové řetězci a nahradí všechna x určenou hodnotou. Vrací nový řetězec.
void run()	Počítá fitness pro všechny jedince v populaci, po zpracování všech jedinců ještě nastaví populaci součet fitness hodnot všech jedinců – sumFitness.
Operators.Operators	
Abstraktní třída (nevytváří instance), jejímž úkolem je výběr vhodných jedinců pro reprodukci – selekce a aplikace operátorů mutace a křížení.	
String[] performCrossover (String gen1, String gen2)	Zohledňuje pravděpodobnost mutace, poté provede mutaci a vrátí pole se dvěmi textovými řetězci – genotypy potomků, nebo vrátí stejné jedince, kteří byli na vstupu funkce.
String performMutation (String gen)	Provede jednobodovou mutaci v náhodném bodě, vrací nový genotyp.
Population generateNewGeneration (Population p)	Tato metoda převolá metodu se stejným názvem s parametry získanými z třídy GlobalsParams. Standardně by měla být volána právě tato metoda (s jedním parametrem) tak, aby program pracoval s hodnotami z třídy GlobalParams.
Population generateNewGeneration (Population p, double mutationPropability, int elitism)	Metoda pro vytvoření celé nové generace, předpokládá, že celá populace ve vstupním parametru je již ohodnocena. Vytváří populaci o stejné velikosti, jako populace vstupní.
Individual performTournament (Population p, boolean minimization)	Vrací jedince ze vstupní populace vybraného pomocí turnajové selekce, Podle parametru

	minimization rozhoduje, zda nižší fitness = lepší (true), nebo naopak.
Individual performRoulette (Population p, boolean minimization)	Funguje podobně jako metoda performTournament, jedince vybere pomocí ruletové selekce.
utils.GrammarReader	
V instanci této třídy budou uloženy pravidla přečtená ze souboru se zapsanou gramatikou – vytváří listy headers (hlavičky) a rules (pravidla).	
GrammarReader ()	Konstruktor nejprve vytvoří instanci třídy java.io.FileReader pomocí cesty k souboru uložené ve třídě GlobalParams, poté vytvoří instanci java.io.BufferedReader a zavolá metodu createRules s touto instancí.
void createRules (BufferedReader buffer)	Metoda přečte soubor s gramatikou a naplní listy headers a rules.
List<String> getHeaders ()	Vrací list headers, tedy seznam neterminálů.
List<List<String>> getRules ()	Vrací list rules – seznam pravidel, která jsou rozvedena z daného neterminálu.
Utils.Utils	
Abstraktní podpůrná třída s užitečnými metodami využitelnými průběhu algoritmu.	
String generateBinaryString (int minCodons)	Generuje binární řetězec, používá se při vytváření počáteční populace. Vstupním parametrem je minimální počet kodonů jedince a volá metodu se stejným názvem doplněnou o vybrané atributy z třídy GlobalParams.
String generateBinaryString (int minCodons, int maxCodons, int codonSize)	Metoda vrací náhodně vytvořený binární řetězec, při tvorbě řetězce se řídí vstupními parametry minimální počet kodonů, maximální počet kodonů a velikost kodonu.

int[] convertBinaryCodons (String codon)	Volá metodu se stejným názvem s přidaným parametrem velikost kodonu, získaným ze třídy GlobalParams.
int[] convertBinaryCodons (String codon, int codonSize)	Tato metoda nejprve parsuje chromozom podle parametru velikosti kodonu a vytváří tak pole textových řetězců. Po rozparsování převede jednotlivé řetězce v poli do desítkové soustavy a vrací nové pole celých čísel.
String mapGenometoPhenotype (int[] codonsArray, int maxNumberOfCycles, GrammarReader gr)	Vstupními parametry jsou pole celých čísel s kodony (viz metoda convertBinaryCodons), maximální počet opakování čtení kodonů a instance třídy GrammarReader. Metoda nastaví potřebné atributy a zavolá metodu convertCodon s indexem 0.
void convertCodon (int index)	Metoda čte hodnoty kodonů z pole codons, vstupním parametrem je index kodonu z tohoto pole. Jednotlivé kodony jsou čteny a převáděny na terminály a neterminály, v případě, že metoda přečetla poslední kodon a stále nebyly rozvedeny všechny neterminály, začne číst kodony od začátku. Maximální počet opakování takového čtení je dán vstupním parametrem.
int[] checkIsExpandable (String rule)	Vstupním parametrem je pravidlo, metoda zjistí, zda jsou v tomto pravidle přítomny neterminály. Pokud ano – vrací pole s indexy hlaviček těchto neterminálů tak, jak jsou uloženy v instanci GrammarReader, jinak vrací pole o velikosti 1 s hodnotou -1.
Utils. MultiThreaded	

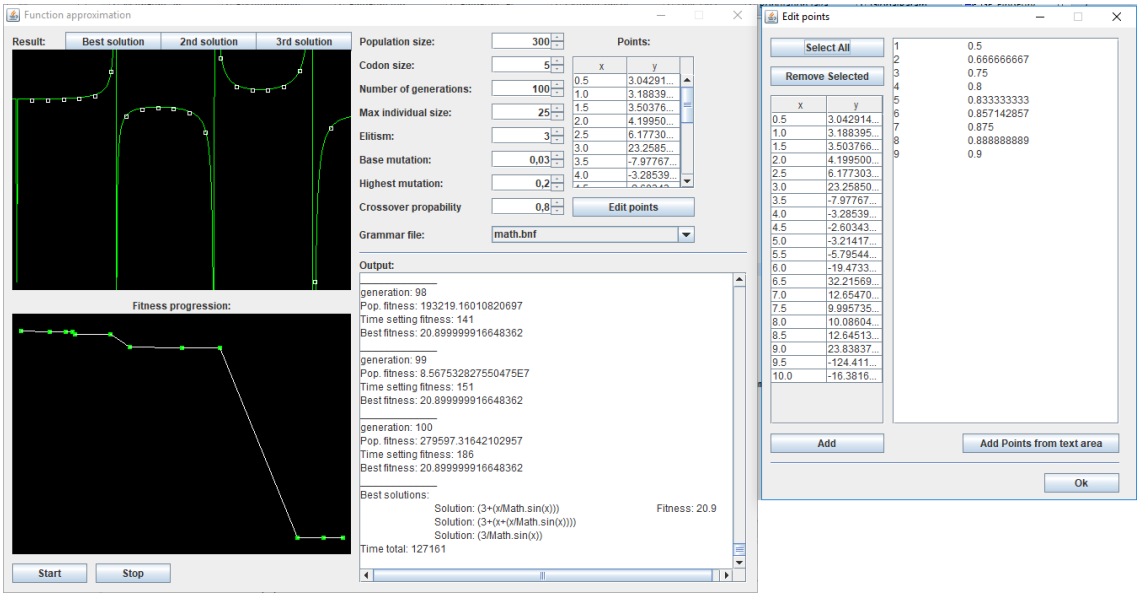
Jedná se o pomocnou třídu, pomocí které lze vybrané činnosti spouštět paralelně a urychlit tak program, Obsahuje vnitřní třídu Task, která dědí z třídy Thread. V aplikaci je třída MultiThreaded využita pro paralelní ohodnocení více jedinců.	
void startTask (Task task)	Spustí úlohu ze vstupního parametru a uloží ji do kolekce tasks.
Task getTask (int id)	Pomocí parametru id prochází kolekci tasks a vrátí úlohu s daným id. Pokud úloha není nalezena, vrací null.
boolean processing ()	Vrací true nebo false podle toho, zda jsou některé úlohy v kolekci tasks ještě aktivní.

Grafické rozhrání

Grafické uživatelské rozhrání – GUI (Graphical User Interface) je rozhrání které umožňuje uživateli ovládat aplikaci pomocí grafických prvků jako jsou tlačítka, menu, posuvníky a další. V jazyce Java je nejčastěji používáno rozhrání Swing, případně jeho předchůdce AWT. Pro účely tvořené aplikace bylo použito právě rozhrání Swing.

Celou aplikaci tvoří jedno hlavní okno a okno, pomocí kterého lze zadávat body pro aproximaci. Hlavní okno umožňuje nastavit všechny počáteční parametry pro spuštění algoritmu, včetně souboru s gramatikou (souborů se v programu vyskytuje více, navíc lze dle potřeby vytvářet nové), v průběhu algoritmu vypisuje důležité informace do textového výstupu a na spodním grafu zobrazuje vývoj fitness hodnot nejlepších jedinců. Po ukončení algoritmu jsou v horním grafu graficky znázorněny průběhy nejlepších řešení a jejich porovnání se zadanými body.

Okno pro zadávání bodů nabízí možnost zadávání bodů přímo do tabulky, případně body z tabulky vymazat. Další možností zadávání bodů je vložení bodů do textového pole, po stisknutí příslušného tlačítka jsou přečteny jednotlivé řádky a pokud jsou zadané body validní, program je vloží do tabulky. Tento způsob zadávání byl navrhnut pro rychlé zadávání bodů z excelu, případně jiných externích programů.



Obrázek 13: Grafické rozhraní aplikace, zdroj: autor

Po stisknutí tlačítka start program nejprve zkontroluje, zda byly zadány souřadnice pro aproximaci, poté nastaví hodnoty do třídy GlobalParams a spustí aplikaci. Aby bylo možné a uživatelským rozhraním manipulovat v průběhu algoritmu, spouští se algoritmus v novém vlákně, tak lze algoritmus v libovolný moment zastavit.

3.2 Testování aplikace

Pro účely testování byly použity body z existujících funkcí, v takovém případě by mělo být možné dosáhnout fitness hodnoty 0, tedy přesného řešení. Z důvodu, že v evolučních algoritmech figuruje silný prvek náhody, není možné dělat závěry z jednoho spuštění algoritmu pro každý test. Proto bude pro každý testovací případ algoritmus spuštěn padesátkrát, získaná data pak budou znázorněna pomocí grafu. Pro účely testování bude použita následující gramatika:

```
<exp> ::= <val>|<exp><op><exp>|<func>
<val> ::= x|<num>
<func> ::= Math.sin<exp>|Math.cos<exp>|Math.pow<exp><com><exp>
<num> ::= 1|2|3|4|5
<op> ::= -|+|*|/
<com> ::= ,
```

3.2.1 Testovací případ – hledání jednoduché funkce

V prvním testovacím případě bude hledána jednoduchá funkce s předpisem:

$$f(x) = \frac{\mathrm{x}}{\sin(\mathrm{x})} + 2$$

Pro aproximaci bude použito 20 bodů v intervalu x od 0,5 do 10. Protože se jedná o takto jednoduchou funkci, program by měl být schopný nalézt kvalitní řešení s menším počtem jedinců a s menším počtem generací, oproti pozdějším testovacím případům. Hodnoty parametrů evolučního algoritmu jsou popsány v následující tabulce.

Tabulka 3: Nastavení evolučního algoritmu pro první testovací případ

Velikost populace	800
Velikost kodonu	5 bitů
Počet generací	100
Maximální velikost jedince	20 kodonů
Elitismus	3
Základní pravděpodobnost mutace	0.03
Nejvyšší pravděpodobnost mutace	0.2

3.2.2 Testovací případ – hledání komplexní funkce

V druhém testovacím případě bude hledána funkce s předpisem:

$$f(x) = \sin(\mathrm{x})\frac{(\mathrm{x}+1)}{\cos(\mathrm{x})}$$

Z této funkce bude použito pro aproximaci 24 bodů v intervalu x od 4,5 do 16. Protože je tato funkce komplexnější (alespoň z hlediska aproximace daných bodů), je nutné použít větší počet jedinců a více generací, přesto však bude mít program pravděpodobně problém najít přesné řešení, nicméně nalezená řešení by měla být dostatečně blízká hledaným bodům.

Tabulka 4: Nastavení evolučního algoritmu pro druhý testovací případ

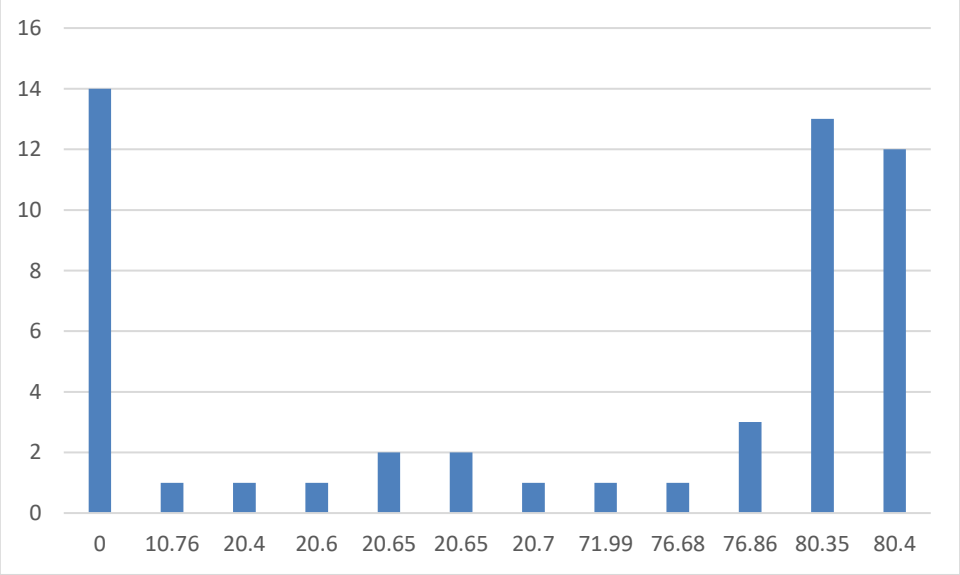
Velikost populace	1500
Velikost kodonu	5 bitů
Počet generací	130
Maximální velikost jedince	23 kodonů
Elitismus	3
Základní pravděpodobnost mutace	0.03
Nejvyšší pravděpodobnost mutace	0.2

3.2.3 Shrnutí výsledků prvního a druhého testu

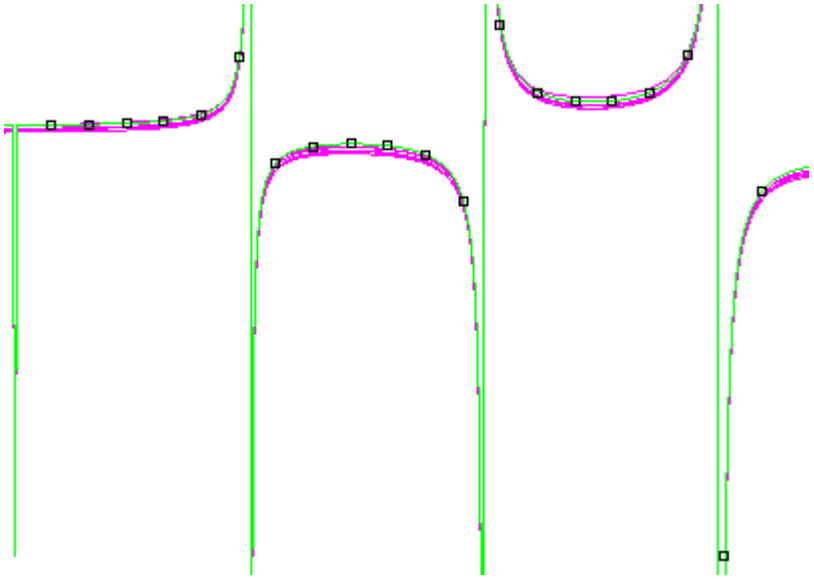
Jak se dalo očekávat, výsledky prvního testu byly vcelku úspěšné, v relativně krátkém časovém úseku, s nízkým počtem jedinců a generací dokázal nalézt uspokojivé výsledky. Nejčastější fitness hodnotou byla hodnota 0 – tedy přesná nalezena hledaná funkce, této hodnoty bylo docíleno z padesáti testů čtrnáctkrát, další nalezené funkce měly pouze minimální odchylky.

Tabulka 5: Tři nejlepší výsledky z prvního testu

Funkce	Fitness	Četnost
$\frac{\mathbf{x}}{\mathbf{sin}\ (\mathbf{x})} + \mathbf{2}$	0	14
$\frac{\mathbf{x}}{\mathbf{sin}\ (\mathbf{sin}(\mathbf{x}))} + \mathbf{2}$	10.76	1
$\frac{\mathbf{sin}\ (\mathbf{x}) + \boldsymbol{x}}{\mathbf{sin}\ (\mathbf{x})}$	20.4	1



Graf 1: Počet výskytů pro dané fitness hodnoty v prvním testu

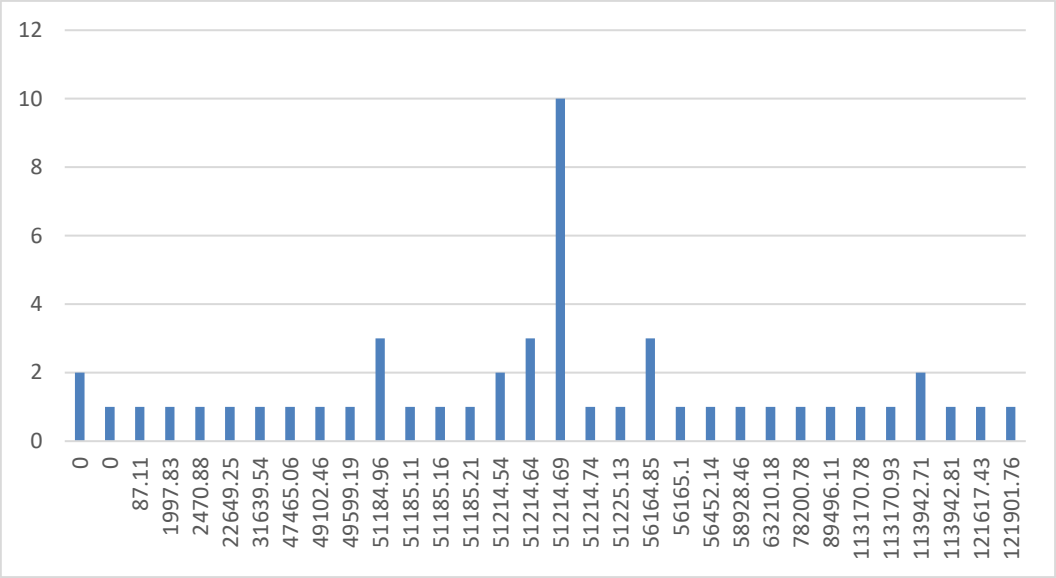


Obrázek 14: Porovnání nalezených funkcí se zadanými body pro první test, zdroj: autor

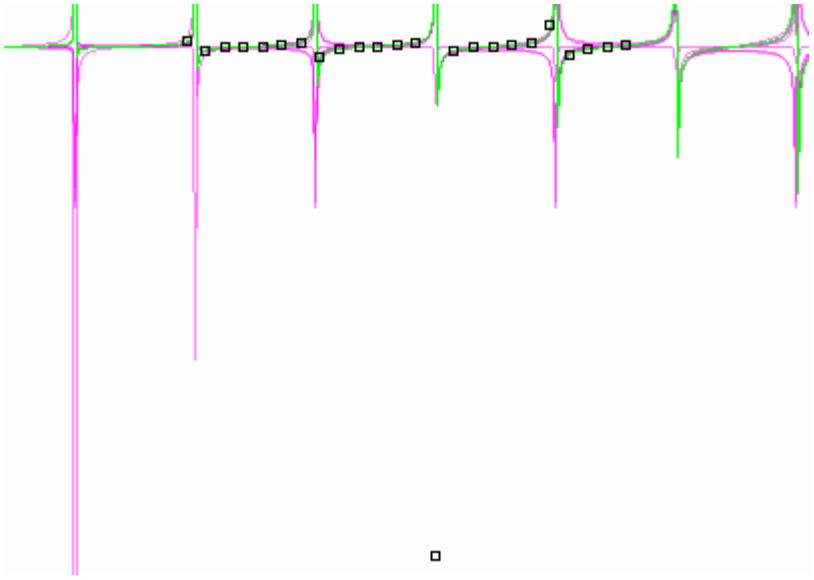
Druhý test, ačkoliv dosáhl přesného výsledku pouze dvakrát, nalezl pokaždé velmi blízké řešení, dá se navíc předpokládat, že v případě navýšení počtu generací a počtu jedinců by se přesný výsledek vyskytoval častěji.

Tabulka 5: Tři nejlepší výsledky z druhého testu + nejčastější výsledek

Funkce	Fitness	Četnost
$\sin(x)\frac{(x+1)}{\cos(x)}$	0	2
$\frac{\sin(x)}{\sin(\frac{\cos(x)}{x+1})}$	0.001	1
$\frac{\sin(x)}{\frac{\cos(x)}{x-\sin(5)}}$	87.109	1
(nejčastější výsledek) $\frac{x}{\frac{\cos(x)}{x\sin(x)}}$	51214.69	10



Graf 2: Počet výskytů pro dané fitness hodnoty ve druhém testu



Obrázek 15:Porovnání nalezených funkcí se zadanými body pro druhý test, zdroj: autor

3.2.4 Testovací případ – vliv nastavení parametrů na přesnost výsledků

V tomto testu bude pozorován vliv nastavení jednotlivých parametrů na kvalitu nalezených řešení, pro tyto testy bude použita funkce třetího polynomu s předpisem:

$$f(x) = 2x^3 + 3x^2 + 5\text{x} + 2$$

V následujících testech je spíše cílem nalezení ideálního nastavení programu oproti předchozím testům, kde bylo cílem získání optimálního řešení.

Postupně bude testován vliv počtu generací a počtu jedinců, pravděpodobnosti křížení a proměnné pravděpodobnosti mutace.

Tabulka 6: Naměřené výsledky z opakovaných testů vlivu počtu generací (nahoře) x počtu jedinců (vlevo)

	40	**80**	**120**	**160**
200	174213.6746	77676.1947	66432.4344	44252.60206
600	134547.7925	42404.26855	29658.3104	20317.90575
1000	53692.82945	30313.6148	25532.0911	19112.32675
1400	47707.713	29055.07535	22191.86395	17514.57725

Tabulka 7: Vliv pravděpodobnosti křížení na naměřené fitness

Křížení	**Průměrné fitness**
0.2	81538.2659
0.4	52703.0755
0.6	42953.1935
0.8	40441.8277
1.0	39595.81405

Tabulka 8: Vliv proměnné mutace na naměřené výsledky

Proměnná mutace	**Průměrné fitness**
0.03 (stejná jako základ)	68008.28894
0.1	51317.76598
0.2	52121.14606

Naměřené výsledky ukazují důležitost vhodného nastavení parametrů pro nalezení kvalitních řešení.

Z testu počtu jedinců a generací vyplívá, že s příliš malou velikostí populace nelze dosáhnout kvalitních řešení ani s použitím velkého počtu generací z důvodu vysoké šance na uvíznutí v lokálním extrému. Podobně není možné nalézt optimální výsledky ani při použití velkých populací a nízkého počtu generací, v takovém případě totiž nejsou jednotlivé chromozomy mezi sebou dostatečně překříženi. Dále test ukazuje, že šance na nalezení lepších výsledků neroste úměrně s počtem generací a velikostí populace – s příliš vysokými nastavenými hodnotami jsou výsledky obdobné jako u nižších hodnot.

Test pravděpodobnosti křížení jasně ukazuje, že s výšší pravděpodobnost klesají fitness hodnoty (jsou nalezeny lepší jedinci), je tedy vhodné nastavit vysokou pravděpodobnost křížení. Pouze mezi hodnotami 0.8 a 1.0 již není příliš zásadní rozdíl, hodnota 1.0 teoreticky může vést k uvíznutí v lokálním optimu.

Posledním testem byl vliv proměnné pravděpodobnosti mutace, ve výsledcích je porovnání vypnuté proměnné mutace (nastavení stejné hodnoty jako základní mutace) a pravděpodobností 10 % a 20 %. Výsledky dokazují důležitost tohoto parametru – proměnná mutace má silnou tendenci k uniknutí z lokálního optima.

3.3 Použití programu pro analýzu dat vodních průtoků

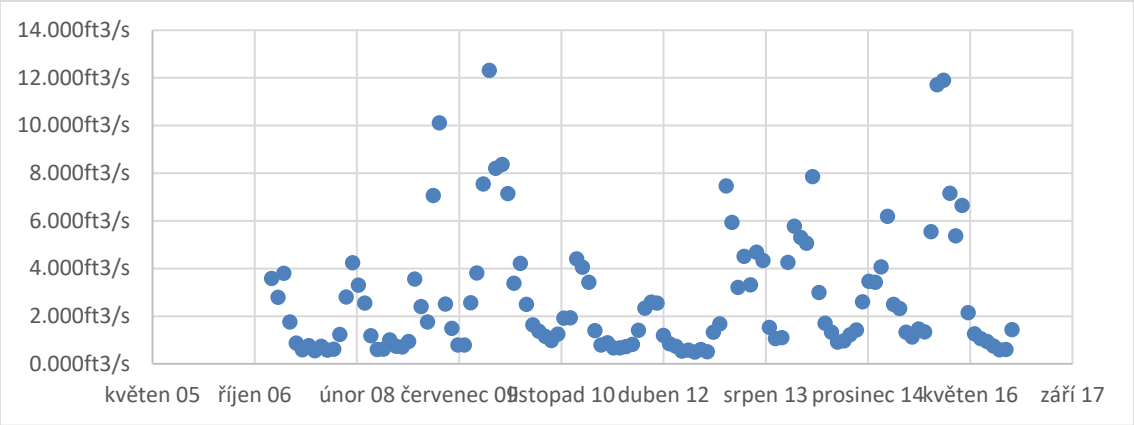
Aplikace byla otestována a jak ukazují výsledky testování, je také schopná docílit dobrých výsledků. V následujícím textu je popsán proces získání dat a aproximace mnohem většího počtu bodů, jejichž matematický předpis není znám.

3.3.1 Popis sběru dat

Pro potřeby bakalářské práce bylo nutné získat podrobná data v časovém období několika let, k tomu posloužila webová stránka USGS Water Data for the Nation (16), která nabízí data téměř všech pramenů v USA. Po rozevření vybraného průtoku na mapě stačí zadat časový interval naměřených dat pro zobrazení (u většiny průtoků jsou zaznamenána data za několik desítek let).

Hlavním kritériem pro výběr vhodného toku byl minimální vliv externích faktorů jako uzavření, či vypouštění blízké vodní nádrže, v přítomnosti těchto faktorů jsou totiž toky silně ovlivněny lidmi a nedají se tak snadno předpovídat budoucí stavy. Nejvhodnějším nalezeným kandidátem pro analýzu dat je tok č. 02349605, FLINT RIVER AT GA 26, NEAR MONTEZUMA, GA, ke kterému jsou evidována denní data již od roku 1904, navíc se v jeho blízkosti nenachází žádná vodní stanice, která by data ovlivnila.

Nejprve byla vybrána data za 10 let – od začátku roku 2007 do konce roku 2016, tato data byla následně zprůměrována v excelu tak, aby pro každý měsíc vznikla jedna průměrná hodnota – dohromady tedy 120 bodů pro aproximaci.



Graf 3: získaná průměrná data pro každý měsíc, takto byla data zadána do aplikace

3.3.2 Hledání vhodné funkce pomocí vytvořené aplikace

Pro aproximaci takto velkého množství dat bylo nutné nastavit velké množství jedinců a generací, dále se dalo předpokládat, že datům přiblíží pouze velmi komplexní funkce, proto byli nezbytní jedinci s podstatně větší maximální velikostí než u jedinců v testovacích případech.

Tabulka 9: Nastavení evolučního algoritmu pro aproximaci dat vodních průtoků v období 10. let

Velikost populace	5000
Velikost kodonu	5 bitů
Počet generací	500
Maximální velikost jedince	80 a 100 kodonů
Elitismus	3
Základní pravděpodobnost mutace	0.03
Nejvyšší pravděpodobnost mutace	0.2

Dále byla ještě rozšířena gramatika přidáním logaritmů, které by mohly být pro řešení klíčové. Výsledná gramatika tedy vypadá následovně:

```
<exp> ::= <val>|<exp><op><exp>|<func>
<val> ::= x|<num>
<func> ::= Math.sin<exp>|Math.cos<exp>|Math.pow<exp><com><exp>|Math.log<exp>
<num> ::= 1|2|3|4|5
<op> ::= -|+|*|/
<com> ::= ,
```

Protože průběh takto nastaveného algoritmu trval velmi dlouho, byl spuštěn pouze pětkrát – třikrát pro velikost jedince 80 a dvakrát pro velikost 100.

3.3.3 Hledání funkcí pro jednotlivé roky

Protože získaná data za časový úsek jsou velmi chaotická, byl program ještě spuštěn pro každý rok zvlášť s cílem nalezení podobností mezi jednotlivými funkcemi. V tomto případě nebude prohledáváno tolik bodů, a proto nemusejí být nastaveny přehnaně vysoké hodnoty parametrů.

Tabulka 10: Nastavení evolučního algoritmu pro hledání funkcí pro jednotlivé roky

Velikost populace	2000
Velikost kodonu	5 bitů
Počet generací	200
Maximální velikost jedince	35 kodonů
Elitismus	3
Základní pravděpodobnost mutace	0.03
Nejvyšší pravděpodobnost mutace	0.2

3.3.4 Souhrn dosažených výsledků

Funkce, které byly programem nalezeny, se částečně přibližují zadaným bodům a nalézají určitý vývoj v rámci let (podle nalezených funkcí průměrné množství proteklé vody během let roste), nicméně všechny nalezené funkce mají problém přiblížit se k bodům s nejvyššími hodnotami. Funkce jsou popsány v následující tabulce a znázorněny na obrázku č. 16.

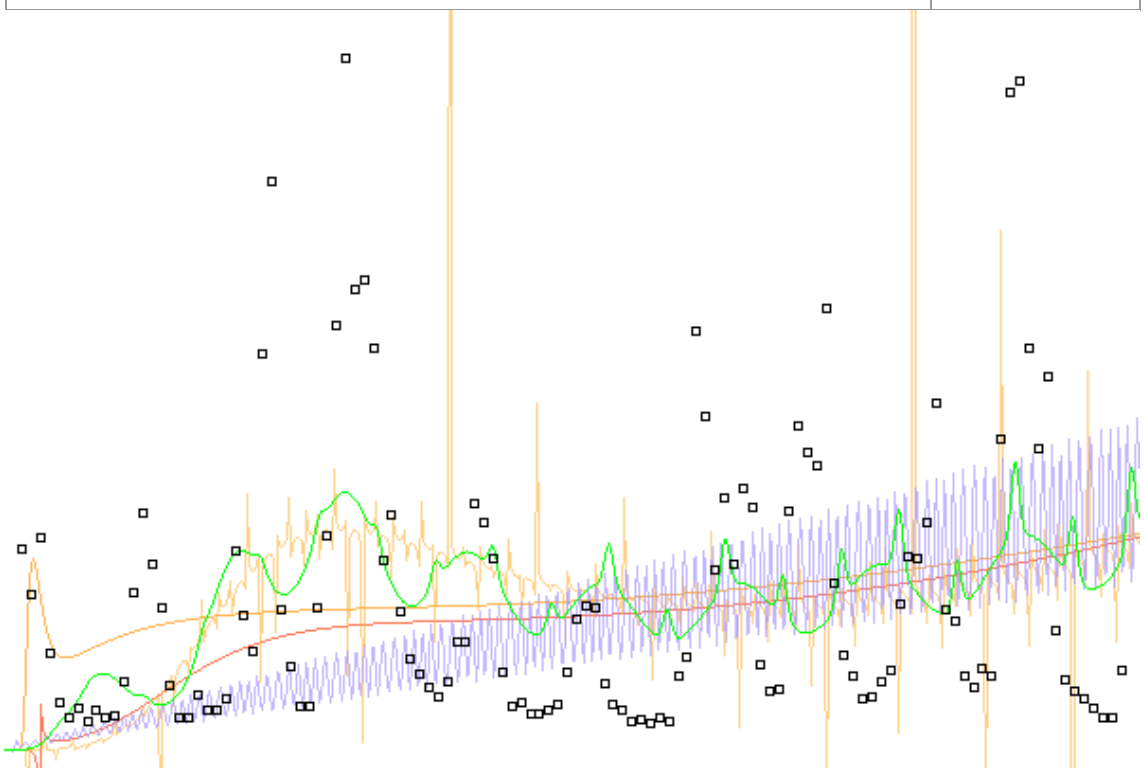
Zá zmínku stojí především třetí funkce, která je na obrázku vyznačena zelenou barvou, tato funkce má z nalezených řešení nejlepší fitness hodnotu a částečně napodobuje body v určitém jevu – podle naměřených dat proteče nejvíce vody ke konci roku/začátkem příštího roku, s blížícím se létem potom voda razantně klesne.

Dá se předpokládat, že program by dokázal nalézt ještě lepší výsledky v případě nastavení ještě větší velikosti jedinců, počtu generací a velikostí populace, zároveň by však razantně vzrostlo množství času pro průběh algoritmu.

Tabulka 11: Nalezené funkce pro vodní průtok za 10 let

Funkce	**Fitness**
$\cos\left(\log\left(x^2\right)\right) + \log\left(1 + 2x\right)\left(4x - \log\left(x^2\right)\right)^{2x^{2/x}}$	7.888×10^8

$\cos\left(\log\left(x^2\right)\right) + \left(\frac{\log\left(x^2\right)}{\log\left(\log\left(x\right)\right)}\right)\left(\frac{\log\left(x\right) + 6x}{\cos\left(\cos\left(\log\left(x\right)\right)\right)}\right)$	7.331×10^8
$\left(2^{\sin(x)}x^{\sin(x)} + 2^{\sin(\log(x))}\left(x^3\right)^{\sin(\log(x))} + 4x\right)\left(\sin\left(x^{\sin(1)}\right) + \log(2x + \log(2x) + 1)\right) + \left(\sin\left(x^{\sin(1)}\right) + \log(2x + \log(2x) + 1)\right)\left(x^2 + 2x - \sin(x)\right)^{\cos(\sin(\log(x)))}$	6.922×10^8
$\left(\frac{\sin(\log(2x)}{\frac{\log(x)}{x} - \sin(2x)} + \log(2x) - \sin(2x) + x\right)\left(\frac{\sin(2x + \log(x))}{\frac{\log(x)}{x} - \sin(2x)} + \left(\log(x) + 2x + 2x + \log(x)\right)^{\cos(\sin(\log(x)))}\right)$	7.2×10^8
$(12x + 10)\left(\frac{3x + 2}{x} + \sin(6x + \sin(x))\right)$	8.777×10^8



Obrázek 16: Porovnání zadaných bodů s nalezenými funkcemi, zdroj: autor

Dále byl program spuštěn pro jednotlivé roky za roky 2007 až 2011, nalezené funkce jsou celkem blízké hledaným bodům, především funkce napodobují trend vyšších hodnot na konci/začátku roku. V následující tabulce je pro každý rok vypsána nejlepší funkce.

Tabulka 12: Nalezené funkce pro jednotlivé roky

Rok	**Funkce**	**Fitness**
2007	$4\mathrm{x}\,(2\mathrm{x}+1)+(x+\sin(x))^{\cos(x)}$	2.376×10^7
2008	$\mathrm{x}^3+\ 4x^2+\ \frac{x\cos(x)}{\cos(x)+1}$	2.185×10^7
2009	$\frac{x-1}{\sin(x)+1}-\ 4x^2\,(\cos(x)-1)$	1.89×10^7
2010	$x^2+\ x^2\cos(x)+\ x^2\cos(x^2)+2x$	2.562×10^7
2011	$(x^2+2x)\cos(1)-\ {x^2}^{\cos(x)}\cos(x^2)$	2.101×10^7

Jak je v tabulce vidět, nalezené funkce pro všechny roky jsou silně závislé na goniometrických funkcích sinus a cosinus, díky kterým napodobují již zmíněný trend.

4 Závěr

Cílem bakalářské práce byl popis evolučních algoritmů, především gramatické evoluce, a jejich využití na problému reálného světa a vytvořit aplikaci, která tento problém řeší. Pro vysvětlení gramatické evoluce bylo taky nezbytné obeznámit čtenáře se zápisem gramatiky pomocí Backus-Naurovy formy (BNF). Protože řešený problém je spjatý hledáním matematické funkce, která se přibližuje zadaným bodům, práce se také zabývá symbolickou regresí a aproximací.

Aplikace je sice navržena pro řešení aproximace funkcí, nicméně (také díky BNF), je vytvořena tak, aby bylo aplikaci snadné modifikovat pro řešení jiných problémů, v takovém případě je nutné pouze vhodně zapsat gramatiku a navrhnout fitness funkci pro daný problém. Díky vývoji v jazyce Java není aplikace závislá na konkrétním operačním systému či architektuře a lze tak spustit na libovolném systému s nainstalovaným virtuálním strojem JVM (Java Virtual Machine).

V průběhu testování bylo prokázáno, že vytvořená aplikace je díky gramatické evoluci vhodná pro řešení aproximace funkcí, i v případě, kdy nebyla nalezena přesná řešení, aplikace předložila jiná, dostatečně kvalitní řešení, zároveň však byla prokázána důležitost vhodného nastavení programu pro nalezení vhodných řešení.

Pro hledání funkce z dat vybraného vodního průtoku za období deseti program předložil řešení, která se sice v určitých hodnotách značně vzdalovala zadaným bodům, přesto však funkce do určité míry sumarizují zadané body, navíc je pravděpodobné, že v případě nastavení více iterací populací by program dosáhl ještě lepších výsledků. Lépe si program dařil při spuštění pro jednotlivé roky. V tomto případě výsledné funkce dostatečně kopírovaly zadané body.

Cíl práce je tedy splněn, aplikace je stabilní, nedisponuje žádnými závažnými problémy či omezeními a je možné ji použít jako podpůrnou aplikaci při analýze dat vodních průtoků či v jiných oblastech, kde je vyžadována aproximace funkcí pro zadané body.

5 Seznam použité literatury

1. **Holland, John H.** *Adaptation in Natural and Artificial Systems.* Ann Arbor : University of Michigan Press, 1975. 0472084607.

2. **GOLDBERG, David E.** *Genetic algorithms in search, optimization, and machine learning.* Boston : Addison-Wesley Pub. Co.,, 1989. 0201157675.

3. **Kachitvichyanukul , Voratas.** Comparison of Three Evolutionary Algorithms: GA, PSO, and DE. *OAK Central.* [Online] [Citace: 24. 5 2017.] http://central.oak.go.kr/journallist/journaldetail.do?article_seq=11096&tabname=abst&resource_seq=-1&keywords=null.

4. **HYNEK, Josef.** *Genetické algoritmy a genetické programování.* Praha : Grada, 2008. 978-80-247-2695-3.

5. **De Jong, K.** *An analysis of the behavior of a class of genetic adaptive systems.* Ann Arbor : University of Michigan, 1975.

6. **Crossover and mutation.** ***ResearchGate.*** **[Online] [Citace: 2. 6 2017.] https://www.researchgate.net/figure/261287213_fig1_Fig-3-Crossover-and-mutation-a-Single-point-crossover-for-a-pair-of-chromosomes-of.**

7. Rocha, Miguel a Neves, José. ***Preventing Premature Convergence to Local.*** **Braga : Universidade do Minho.**

8. Koza, John R. ***Genetic programming: on the programming of computers by means of natural selection.*** **Cambridge : MIT Press, 1992. 9780262111706.**

9. Cramer, Nichael Lynn. A Representation for the Adaptive Generation of Simple Sequential Programs. [Online] http://homepages.sover.net/~nichael/nlc-publications/icga85/index.html.

10. Macháček, Martin. Genetické programování. ***Posterus.*** **[Online] [Citace: 10. 6 2017.] http://www.posterus.sk/?p=10198.**

11. Tsang, Edward. EDDIE In Financial Decision Making. ***University of Essex.*** **[Online] [Citace: 5. 7 2017.] http://cswww.essex.ac.uk/CSP/finance/Eddie/Overview/.**

12. Pareto, Vilfredo. ***Cours d'Économie Politique.*** **Lausanne : Université de Lausanne, 1897. 0002-7162.**

13. Drahošová, Michaela. *Symbolická regrese a koevoluce.* Brno : Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav počítačových systémů, 2011.

14. Pareto efficiency. *Wikipedia.* [Online] [Citace: 10. 7 2017.] https://en.wikipedia.org/wiki/Pareto_efficiency.

15. Ryan, Conor, Collins, J.J. a O'Neill, Michael. *Grammatical Evolution: Evolving Programs for an Arbitrary Language.* Boston : University of Limerick, 1998. Limerick.

16. Bezděk, Pavel. *Gramatická Evoluce – Java.* Brno : Vysoké učení technické v Brně, 2009.

17. Dempsey, Ian, O'Neill, Michael a Brabazon, Anthony. *Foundations in Grammatical Evolution for Dynamic Environments.* Berlin : Springer, 2009. 9783642003141.

18. Cleary, Robert. *Extending Grammatical Evolution.* Limerick : University of Limerick, 2005.

19. Zelinka, Ivan. Symbolic regression - an overview. *Lappeenranta University of Technology.* [Online] https://www.mafy.lut.fi/EcmiNL/older/ecmi35/node70.html.

20. Shure, Loren. Data Driven Fitting. *MathWorks.* [Online] [Citace: 19. 7 2017.] https://blogs.mathworks.com/loren/2011/01/13/data-driven-fitting/.

21. Golsing, James, a další. *The Java Language Specification, Java SE 8 Edition.* Boston : Addison-Wesley Professional, 2014. 9780133900699.

22. Viral Patel. Java Virtual Machine, An inside story!! *ViralPatel.* [Online] [Citace: 26. 7 2017.] http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/.

23. U.S. Geological Survey. USGS Water Data for the Nation. *USGS Water Data for the Nation.* [Online] U.S. Department of the Interior. https://waterdata.usgs.gov/.

6 Přílohy

1) CD s elektronickou verzí práce, vytvořenou aplikací a souborem ve formátu xlsx s podrobnými daty vodního průtoku.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Syrový Lukáš	Žďár nad Orlicí 170, Žďár nad Orlicí	I1300867

TÉMA ČESKY:

Evoluční algoritmy a jejich aplikace

TÉMA ANGLICKY:

Evolution algorithms and their application

VEDOUCÍ PRÁCE:

Ing. Barbora Tesařová, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je seznámit se s principy gramatické evoluce a následně vytvořit aplikaci, která bude pomocí GE generovat regresní modely.

1.Úvod
2.Teoretická část
2.2.Principy evolučních algoritmů
2.3.Gramatická evoluce
2.4.Symbolická regrese
3.Praktická část
3.1.Tvorba a popis vytvořené aplikace
3.2.Testování
3.3.Zhodnocení výsledků
4.Závěr

SEZNAM DOPORUČENÉ LITERATURY:

1. Hynek, Josef: Genetické algoritmy a genetické programování.Grada: Praha 2008
2. Holland, J. H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan, 1975
3. John R. Koza: Human-Competitive Results Produced by Genetic Programming
4. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in aaArbitrary Language
5. Vladimír Mařík, Olga Štepánková, Jiří Lažanský a kolektiv: Umělá inteligence

Podpis studenta: Syrový ..... **Datum:** 30.1.2017

Podpis vedoucího práce: ..... **Datum:** 30.1.2017