Genetické programování v prostředí Mathematica

Genetic programming in Mathematica

Bc.Martin Macháček

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Martin MACHÁČEK**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Genetické programování v prostředí Mathematica**

Zásady pro vypracování:

1. **Seznamte se s algortimy Genetického programování (GP)**
2. **Vytvořte v prostředí Mathematica tyto algoritmy**
3. **Algoritmy otestujte na vhodně zvolených a uznávaných testovacích příkladech**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. O'REILLY, Una-May, et al. Genetic Programming Theory and Practice II. USA : Springer Science + Business Media, Inc., 2005. 337 s. ISBN 0-387-23253-2.
2. ZELINKA, Ivan, OPLATKOVÁ, Zuzana, ŠENKEŘÍK, Roman. Aplikace umělé inteligence : aneb vybrané statě z evolučních algoritmů. 1. vyd. Zlín : Univerzita Tomáše Bati ve Zlíně, 2010. 151 s. ISBN 978-80-7318-898-6.
3. KVASNIČKA, Vladimír, POSPÍCHAL, Jiří, TIŇO, Peter. Evolučné algoritmy. 1. vyd. Bratislava : Vydavateľstvo STU, 2000. 223 s. ISBN 80-227-1377-5.
4. MAŘÍK, Vladimír, et al. Umělá inteligence(4), 1.vyd., Praha: Academia 2003, ISBN 80-200-1044-0, Kapitola 5, Genetické programování a vybrané problémy evolučních výpočtů, s. 128-170.
5. HYNEK, Josef. Genetické algoritmy a genetické programování. 1. vyd. Praha : Grada Publishing, a.s., 2008. ISBN 978-80-247-2695-3, Kapitola 13, Genetické programování, s. 123-134.

Vedoucí diplomové práce: **doc. Ing. Ivan Zelinka, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem této diplomové práce je popsání a vytvoření algoritmu genetického programování, který je řazen do skupiny evolučních algoritmů. V teoretické části jsou popsány principy a funkčnost algoritmu genetického programování. Výsledkem praktické části jsou funkce vytvořené v prostředí Mathematica, které v celku tvoří algoritmus genetického programování. Správnost algoritmu je testována na všeobecně uznávaných testovacích příkladech navržených speciálně pro algoritmus genetického programování.

Klíčová slova:

Optimalizace, evoluční algoritmus, genetické programování, symbolická regrese

ABSTRACT

The aim of this work is discribing and making the genetic programming algorithm which belong to evolutionary algorithms. In theoretical part are discribing principles and functionality of the genetic programming algorithm. The result of practical part are function created in computation system Mathematica and they together create the genetic programming algorithm. The correctness of algorithm is tested on classical examples, designed especially for the genetic programming algorithm.

Keywords:

Optimization, evolutionary algorithm, genetic programming, symbolic regression

Na tomto místě bych chtěl poděkovat svému vedoucímu diplomové práce prof. Ing. Ivanu Zelinkovi, Ph.D., zejména za zajímavé téma pro mou práci, dále pak za odborné vedení, vstřícný přístup a čas, který mi věnoval.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně ……

Podpis diplomanta

OBSAH

ÚVOD

V dnešní moderní době se v každém odvětví lidské činnosti dostáváme do situací, ve kterých narazíme na technický problém. Vzhledem k naší rostoucí technické vyspělosti zároveň roste i složitost těchto problémů. Stále častěji jsou to problémy tak složité, že je nejsme schopni vyřešit běžnými analytickými metodami nebo se setkáváme s problémy, kde nevíme, jak je vyřešit nebo jaká má být struktura ideálního řešení. Také proto v posledních 30 letech vznikla a vyvíjela se řada algoritmů, která by dokázala řešit širokou škálu problémů, u nichž jsou argumenty účelové funkce definovány v různých oborech nebo jsou na řešení kladena různá ekonomická či fyzikální omezení. Tyto algoritmy jsou nazývány jako tzv. evoluční algoritmy a čím dál častěji se k nim, při řešení různých technických problémů uchylují odborníci z různých technických oborů.

V této práci jsem si dal za cíl vytvořit popis algoritmu genetického programování a jeho jednotlivých částí. Genetické programování se řádí mezi tzv. evoluční algoritmy, které jsou hojně využívány k řešení optimalizačních problémů. Evoluční algoritmy vycházejí z principů Darwinovské evoluční teorie.

Mým dalším cílem bylo současně vytvořit ve výpočetním prostředí Mathematica funkce, které v celku budou tvořit algoritmus genetického programování. Vzhledem ke skutečnosti, že genetické programování lze aplikovat na širokou škálu problémů, snažil jsem se vytvořit funkce tak, aby také výsledný algoritmus byl aplikovatelný na různé problémy jen s minimálním zásahem do kódu.

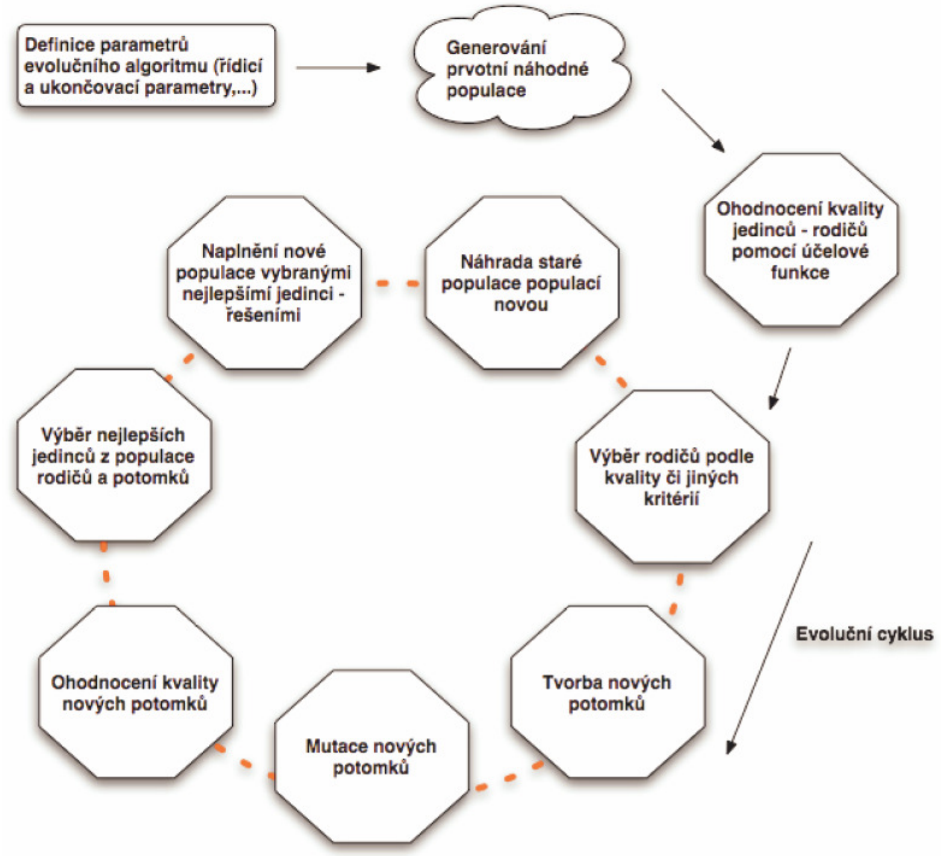
I. TEORETICKÁ ČÁST

1 EVOLUČNÍ ALGORITMY

Evoluční výpočetní technicky (EVT) jsou numerické algoritmy, které vycházejí ze základních principů Darwinovy teorie evoluce, jejíž hlavní ideou je předávání rodičovského genomu novým potomkům a následné uvolnění životního prostoru těmto potomkům. Technologie EVT stojí a padá na existenci tzv. evolučních algoritmů (EA), které v podstatě tvoří většinu EVT. Mimo EA existují ještě další rozšíření, jako genetické programování, kterým se tato práce zabývá, evoluční hardware apod.[2]

Evoluční algoritmy představují netradiční přístup při hledání optimálního řešení složitých optimalizačních problémů, které nejsou řešitelné klasickými technikami. Evoluční algoritmy v současnosti patří mezi základní nástroje moderní informatiky v případech hledání řešení v extrémně složitých situacích, kdy použití standardních deterministických metod založených na technikách úplného prohledávání není možné. Ukazuje se, že evoluční metafora je velmi efektivním přístupem k řešení těchto složitých problémů a to zejména v případech, kdy nepotřebujeme optimální řešení problému, ale vystačíme si i s kvalitním suboptimálním řešením.[2], [3]

Podle klasické Darwinovy teorie evoluce je uznáváno evoluční dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým lepším potomkům, kteří se stávají novými rodiči. Schéma principu evoluce je znázorněno na obrázku (Obr.1). [2]



Obr. 1.:Evoluční cyklus [2]

Evoluční algoritmy nejsou populární jen proto, že jsou moderní a odlišné od klasických algoritmů, ale hlavně pro fakt, že v případě vhodného aplikování jsou schopny nahradit člověka.[2]

1.1 Optimalizační algoritmy

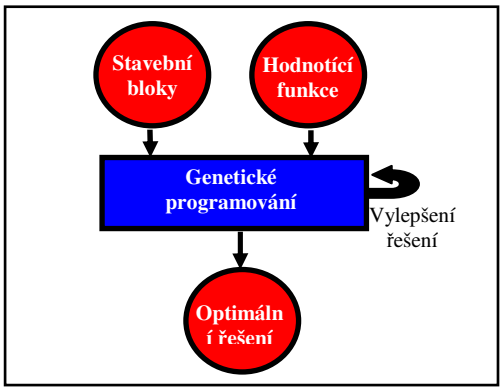
Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou nevhodné či nereálné. Při vhodné implementaci mohou být aplikovány tak, že není potřeba častého uživatelského zásahu do činnosti zařízení, v nichž jsou použity. Většina problémů inženýrské praxe může být definována jako optimalizační úlohy, např. nalezení optimální trajektorie robota či optimální tloušťky stěny tlakové nádoby, optimální nastavení parametrů regulátoru atd. Jinými slovy, řešený problém lze převést na matematickou úlohu danou vhodným funkčním předpisem, jejíž optimalizace vede k nalezení argumentů účelové funkce, což je jejím cílem. Příkladů lze nalézt nespočetně. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí, přičemž definiční

obory těchto argumentů mohou být různorodého charakteru. Navíc v rámci optimalizace mohou být uplatňovány různé penalizace a omezení nejen na dané argumenty, ale také na funkční hodnotu optimalizované funkce. Řešení takového optimalizačního problému analytickou cestou je mnohdy možné, nicméně značně komplikované a zdlouhavé. [2]

2 GENETICKÉ PROGRAMOVÁNÍ

Genetické programování (GP) je jedna z evolučních výpočetních technik (EVT), která slouží k řešení optimalizačních problémů pomocí počítačů, vycházející z principů přirozeného výběru podle Darwinovy evoluční teorie. V GP počítače automaticky řeší problémy, aniž by byla uživateli známa nebo definována forma nebo struktura optimálního řešení.

Zjednodušeně lze říci, že stanovíme základní stavební bloky, ze kterých může být řešení složeno, a zároveň stanovíme analytickou metodu, která bude popisovat, jak dobře dané řešení zadanému problému vyhovuje. Ostatní je záležitostí samotného GP. Zjednodušený postup je znázorněn na obrázku (Obr.2).



Obr. 2.:Schéma GP

Historie GP sáhá až do roku 1957, kdy se Friedberg snažil vytvořit samoučící algoritmus. Program byl reprezentován jako sekvence instrukcí, jejichž kombinací měl vzniknout výsledný program. [9]

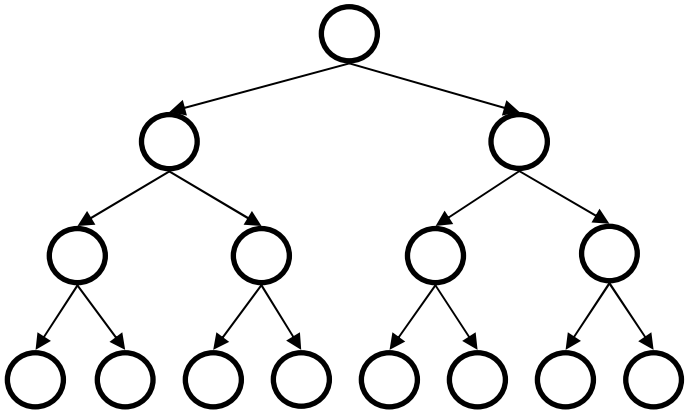
Rozkvět GP však začíná teprve v 80. letech 20. století, kdy se objevují teoretické práce vědců Smitha a Cramera. Smith a Cramer ve svých vědeckých pracích o evolučních algoritmech definují genetické operace křížení a mutace a zabývají se representací jedinců pomocí stromové struktury. [9]

Za zakladatele GP je nicméně považován britský vědec John R. Koza, který ve svých pracích položil teoretické základy GP. Koza také definoval mnoho všeobecně uznávaných testovacích příkladů pro ověření funkčnosti GP. Jako příklad lze uvést učení Boolean funkcí, problém umělého mravence nebo příklady z oblasti symbolické regrese. Koza také standardizoval používání stromové datové struktury pro vyjádření jedinců v GP. [8], [9]

S rozvojem počítačů na konci 20. století a na začátku 21. století nachází GP uplatnění v mnoha oblastech, např. v medicíně, ekonomii a finančnictví, fyzice, astronomii, robotice, v oblasti komunikačních technologií, vojenských technologií nebo biochemii.

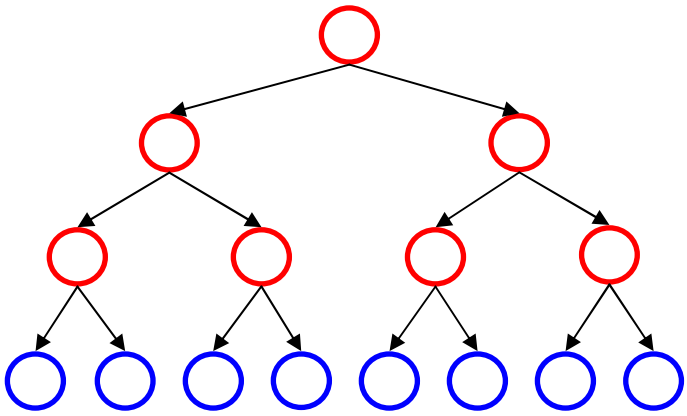
2.1 Reprezentace

Programy, výrazy nebo algoritmy, které v GP představují hledané řešení, jsou obvykle reprezentovány ve formě tzv. syntaktických stromů. Příklad syntaktického stromu je uveden na obrázku (Obr.3). [4]



Obr. 3.:Syntaktický strom

Syntaktický strom popisuje strukturu programu a skládá se z uzlů a listů. Z každého uzlu vychází nejméně jedna hrana. Listy představují koncové vrcholy stromu. [5]



Obr. 4.:Syntaktický strom s vyznačením uzlů a listů

Vedle reprezentace pomocí syntaktických stromů existují ovšem také jiné implementace GP, ve kterých jsou řešení reprezentována např. ve formě obecných orientovaných grafů (systém PADO – Paralel Algorithm Discovery and Orchestration) nebo jako síť uzlů v kartézském souřadnicovém systému (CGP – Cartesian Genetic Programming). Další

reprezentace jedinců je možná s využitím kontextové nebo bezkontextové gramatiky (GGGP – Grammar-based Genetic Programming). [4]

Nicméně pro účely dalšího výkladu bude v této práci použita stromová struktura reprezentace.

2.2 Terminologie

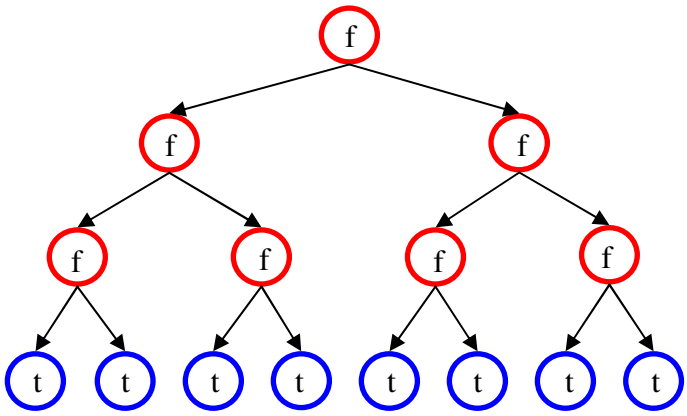
2.2.1 Jedinec, populace

Stejně jako u ostatních EVT, tak i v případě GP se setkáváme s pojmy jedinec a populace. Jedinec představuje v GP jedno aktuální konkrétní řešení daného problému a populací se rozumí množina předem stanovené velikosti, obsahující daný počet jedinců. [4]

2.2.2 Funkce, terminály

Každý jedinec má svou specifickou strukturu a je konstruován ze dvou množin symbolů. Z množiny terminálů $T = \{t_1, t_2, ..., t_m\}$ a z množiny funkcí $F = \{f_1, f_2, ..., f_n\}$. Každá konkrétní funkce $f_i \in F$ má specifický počet argumentů $b_1, b_2, ..., b_j$. V závislosti na konkrétním řešeném problému mohou být funkcemi standardní aritmetické operace (sčítání, odčítání, násobení a dělení), standardní matematické funkce (sin, cos, tan, cotg, abs, apod.), logické operace (and, or, not, xor, apod.), podmíněné příkazy (if-then-else, apod.), iterativní operátory (do, while, apod.) a jiné. Terminály mohou být proměnné, konstanty nebo funkce bez argumentů, které mají nějaký vedlejší efekt (MoveLeft, TurnRight, apod.). [6]

Při reprezentaci jedinců pomocí syntaktických stromů představují uzly funkce a listy reprezentují terminály. Počet hran vycházejících z každého uzlu představuje počet argumentů funkce. Počet argumentů dané funkce se také označuje jako arita této funkce. [5]



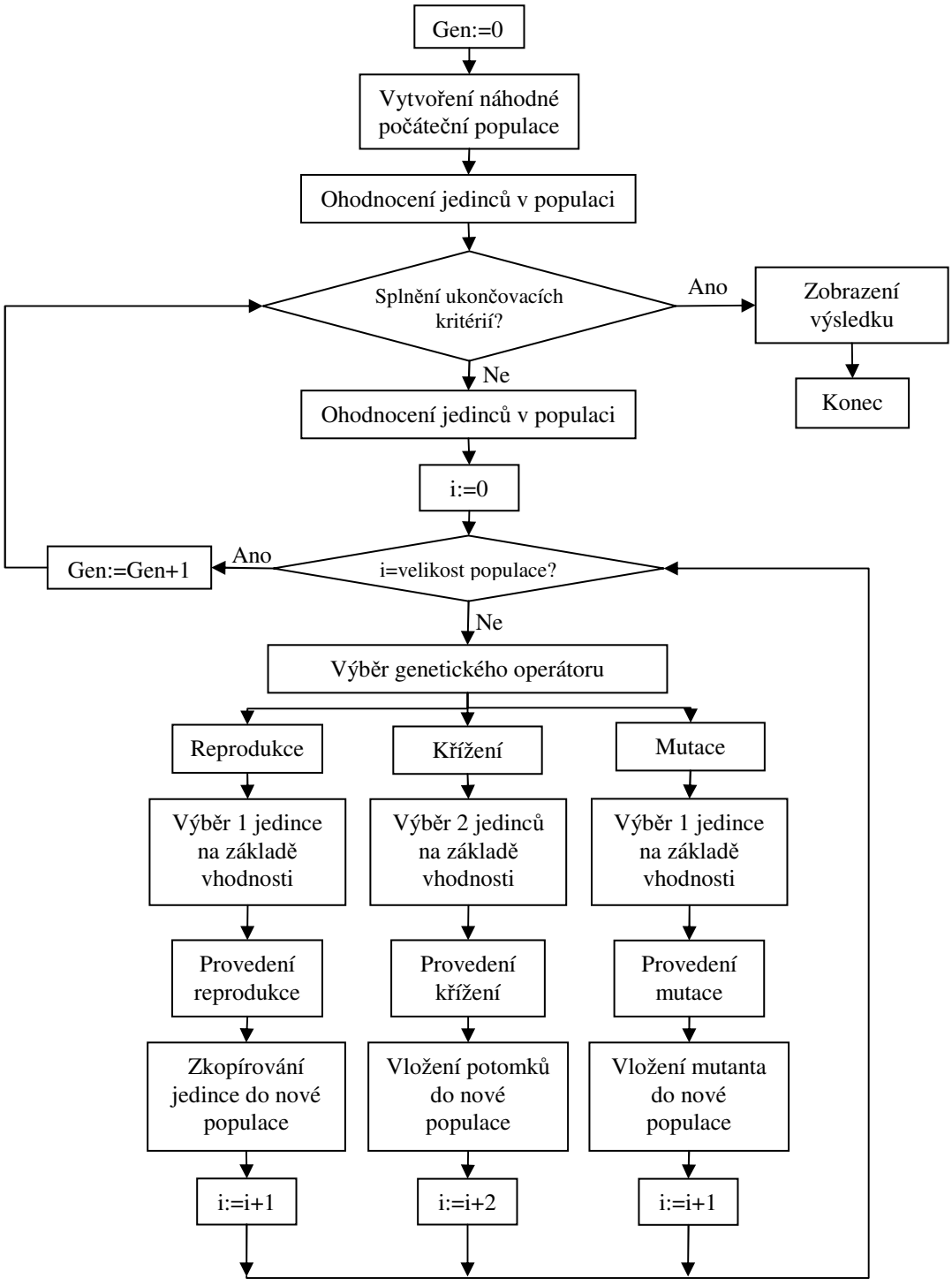
Obr. 5.:Reprezentace jedince

Na množiny *F* a *T* jsou kladeny dva základní požadavky. Prvním je tzv. požadavek uzavřenosti množiny funkcí a množiny terminálů. Jinými slovy, je nutné, aby výstup libovolné funkce a terminálu mohl figurovat jako kterýkoliv argument jakékoliv jiné funkce. Druhým požadavkem je tzv. úplnost a postačitelnost množiny funkcí a množiny terminálů vzhledem k dané úloze. Rozumí se tím, že prvky množiny funkcí a množiny terminálů musí být vhodně zvoleny v závislosti na řešené úloze. [5], [6]

2.3 Algoritmus genetického programování

2.3.1 Schéma algoritmu genetického programování

Algoritmus GP se skládá z několika částí, které na sebe logicky navazují. Schéma algoritmu GP je znázorněno na obrázku (Obr.6).



Obr. 6.:Algoritmus GP

Samotný algoritmus začíná vygenerováním počáteční populace. Populace má předem definovanou velikost a je volena v závislosti na složitosti řešeného problému. Po vytvoření počáteční populace dojde k ohodnocení všech jedinců v populaci předem stanovenou analytickou metodou. Následuje kontrola splnění ukončovacích podmínek. Pokud tyto podmínky nejsou splněny, dochází k vytvoření nové generace jedinců. Nejprve se zcela

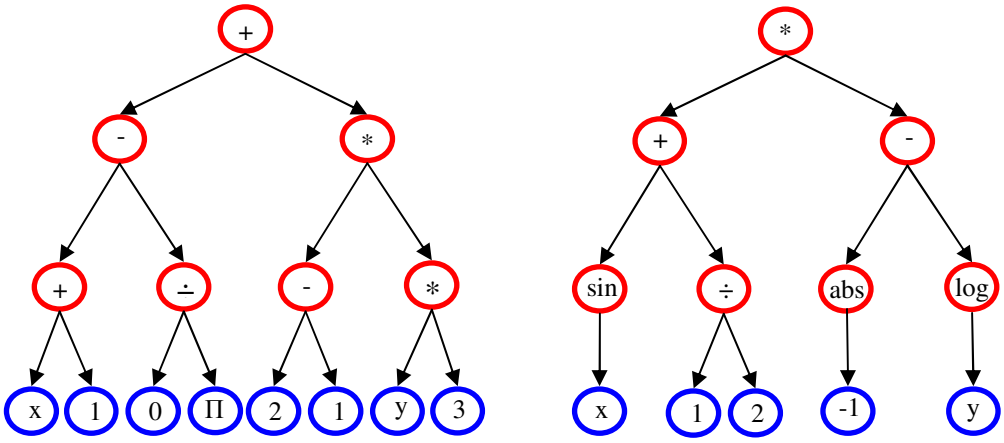
náhodně vybere tzv. genetický operátor – reprodukce, mutace nebo křížení. Následně se tento genetický operátor aplikuje na jedince vybrané na základě jejich vhodnosti. Počet vybraných jedinců závisí na zvoleném genetickém operátoru. Při reprodukci a mutaci je vybrán jen jeden jedinec, pro křížení je třeba mít jedince dva. Aplikací genetických operátorů vzniknou potomci, kteří jsou vloženi do nové populace. Potomci jsou tímto postupem vytvářeni tak dlouho, dokud není velikost nové populace stejná jako velikost populace původní. U nově vzniklé populace dochází opět k ohodnocení všech jedinců a celý proces se opakuje do té doby, dokud nejsou splněny stanovené ukončovací podmínky. Ukončovací podmínky jsou dvojího typu. Jednak může dojít k ukončení evoluce, pokud došlo k vytvoření předem stanového počtu generací nebo v případě, že se v dané generaci vyskytuje optimální řešení nebo dostatečně kvalitní suboptimální řešení. [1], [3], [6]

2.4 Inicializace populace

V GP jsou jedinci v počáteční populaci vytvářeni náhodnou kombinací funkcí a terminálů z daných množin *F* a *T*. Zároveň je na začátku evoluce nutné definovat maximální hloubku syntaktických stromů v počáteční populaci. Maximální počáteční hloubkou se rozumí počet hran vedoucích z kořene stromu k nejvzdálenějšímu uzlu. Ve většině případů se používá počáteční hloubka $D_{\max} = 6$. Pro generování počátečních stromů o dané hloubce existuje několik metod, z nichž nejpoužívanější jsou úplná metoda, rostoucí metoda a metoda půl na půl. [5]

2.4.1 Úplná metoda

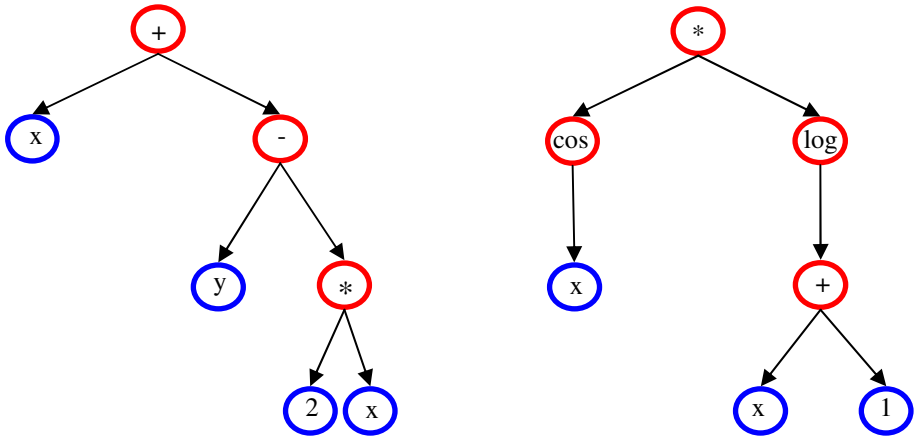
U této metody jsou vytvářeny stromy, ve kterých se všechny listy nacházejí ve stejné hloubce. Neznamená to ovšem, že všechny počáteční stromy budou mít stejný počet uzlů nebo stejný tvar. Tvar syntaktického stromu je ovlivněn aritou náhodně vybraných funkcí na pozicích uzlů. Vytváření stromů probíhá tak, že se v maximální hloubce vybírají pouze terminály a v ostatních hloubkách se volí pouze funkce. [5]



Obr. 7.:Tvorba populace úplnou metodou

2.4.2 Růstová metoda

Při použití růstové metody vznikají mnohem rozmanitější syntaktické stromy, protože v každé vrstvě s výjimkou poslední vrstvy se rozhoduje zcela náhodně, zda bude použita funkce nebo terminál. Jakmile je vybrán terminál, je příslušná větev stromu hotova bez ohledu, zda se dosáhlo požadované maximální povolené hloubky. [5]



Obr. 8.:Tvorba populace růstovou metodou

2.4.3 Lineární půl na půl metoda

V praxi se ovšem doporučuje používat metodu půl na půl, což znamená, že polovina jedinců v počáteční populaci je vytvořena úplnou metodou a druhá polovina je vytvořena rostoucí metodou. Zároveň by měly být rovnoměrně zastoupeny syntaktické stromy různých hloubek. Tedy, např. při velikosti populace $N=100$ a maximální hloubce $D_{max}=6$ bude 50 stromů generováno úplnou metodou (z toho 10 stromů s hloubkou 2, 10

s hloubkou 3, atd.) a 50 stromů rostoucí metodou (z toho 10 stromů s hloubkou 2, atd.). Tato metoda zajišťuje vznik populace s dostatečně bohatou škálou jedinců. [5]

Stejní jedinci v počáteční populaci mohou podle Kozy nepřijatelně snížit genetickou rozmanitost populace. Tudíž je žádoucí, ale ne nezbytné, při vytváření počáteční populace zabránit tomu, aby se v této populaci vyskytovali dva stejní jedinci. [6]

2.5 Vhodnost

Při vytváření počáteční populace a v průběhu evoluce vznikají na základě předdefinovaných funkcí a terminálů jedinci s velmi rozmanitým složením. Jelikož toto vytváření probíhá zcela náhodně a nekontrolovatelně, existuje pravděpodobnost vzniku jedinců, kteří budou pro řešení zadaného problému v porovnání s ostatními jedinci v populaci méně vhodní nebo zcela nevhodní. Odpověď na to, jak moc je jedinec vhodný pro řešení daného problému nám dává určení vhodnosti všech jedinců. Ohodnocení jedinců v populaci je hlavním a často jediným prostředkem k rozpoznání optimálního řešení. Ohodnocení může být prováděno mnoha různými způsoby. [6]

2.5.1 Hrubá vhodnost

První způsob ohodnocení je úzce spjat s účelovou funkcí definovanou pro daný problém. Pro tento způsob se používá anglický název *raw fitness* (volně přeloženo jako hrubá vhodnost). Na tuto vhodnost nejsou kladeny žádné omezení. Hrubá vhodnost je úzce spjata s konkrétním řešeným problémem a je vyjádřena terminologií použitou u daného problému. V podstatě se dá říci, že se jedná o účelovou funkci. [6]

$$F_R = f(x)$$

Například u problému tzv. umělého mravence bude F_R představovat počet kousků jídla sněženého mravencem a např. u symbolické regrese bude F_R definována jako součet absolutních hodnot odchylek správného řešení a aktuálního vypočteného řešení. [6]

Protože hrubá vhodnost je definována značně volně a bez jakéhokoliv omezení, není použitelná v GP na všechny problémy. Obvykle pouze poskytuje zpětnou vazbu uživateli. Spíše je v GP zvykem převádět hrubou vhodnost na tzv. standardizovanou vhodnost (*standardized fitness*). [6]

2.5.2 Standardizovaná vhodnost

Standardizovaná vhodnost je jinou formulací hrubé vhodnosti. Formulujeme ji tak, že menší hodnota značí lepší řešení. Pokud u konkrétního problému značí menší hodnota hrubé vhodnosti lepší řešení, potom je standardizovaná vhodnost rovna hrubé vhodnosti:

$$F_S = F_R$$

Je vhodné a žádoucí, aby hodnota standardizované vhodnosti pro jedince představujícího optimální řešení byla rovna 0. [6]

Pokud představuje větší hodnota hrubé vhodnosti lepší řešení, je standardizovaná vhodnost určena rozdílem maximální hodnoty hrubé vhodnosti a vypočtené hrubé vhodnosti pro aktuálního jedince, tedy:

$$F_S = F_{R\max} - F_R$$

Jediné podmínky kladeny na standardizovanou vhodnost jsou ty, že standardizovaná vhodnost nesmí být záporná a menší hodnotou vhodnosti je reprezentován vhodnější jedinec. Je běžné, ale ne absolutně nezbytné definovat optimálního jedince hodnotu 0. [6]

2.5.3 Upravená vhodnost

Standardizovaná vhodnost není omezená maximální hodnotou, která může být přiřazena ohodnocenému jedinci. Tato skutečnost může být poněkud matoucí a proto je jednodušší, pokud lze říci, zda je konkrétní jedinec lepší než jiný a především jak moc je lepší. Z tohoto důvodu převádíme standardizovanou vhodnost na tzv. upravenou vhodnost (*adjusted fitness*). Tato vhodnost je určena podle vzorce:

$$F_A = \frac{1}{1 + F_S}$$

Hodnoty jsou v rozsahu od 0 do 1, přičemž větší hodnota značí lepšího jedince, což je přesně naopak než u standardizované vhodnosti. Upravená vhodnost tedy přesně určuje, jak moc je jedinec lepší v porovnání s jiným jedincem v populaci. [6]

2.5.4 Normalizovaná vhodnost

Vedle výše zmíněných typů vhodnosti existuje ještě jeden užitečný a využívaný typ vhodnosti, tzv. normalizovaná vhodnost (*normalized fitness*). Je určena následovně:

$$F_N = \frac{F_S}{\sum_{i=1}^{N} F_S}$$

Pro normalizovanou vhodnost platí, že může nabývat hodnot v rozmezí 0-1, větší hodnota značí lepšího jedince a součet normalizovaných vhodností u všech jedinců v populaci je roven 1. [6]

2.6 Výběr rodičů

U genetických operátorů, jež budou popsány za chvíli, je nutné na začátku vybrat vhodné jedince, na které budou tyto operátory aplikovány. Tento výběr se děje několika způsoby, z nichž standardně používané jsou tzv. turnajová selekce a výběr na základě vhodnosti jedince. [6]

2.6.1 Turnajová selekce

Nejčastěji používanou metodou pro výběr jedinců, kteří se budou účastnit genetických operací v GP, je turnajová selekce. Z populace je náhodně vybrán určitý počet jedinců (nejčastěji dva jedinci). Tito jedinci se navzájem porovnají a jako rodič je posléze vybrán jedinec s lepší hodnotou normalizované vhodnosti. Turnajová selekce sleduje pouze to, který jedinec je lepší než jiný. Neříká nám jak moc je lepší. Výhodou této výběrové metody je, že nedochází k upřednostňování nejlepších jedinců. Kdyby tomu tak bylo, vedlo by to k rapidnímu snížení rozmanitosti populace. Turnajová selekce zajišťuje, že i jedinec s průměrnou vhodností má šanci stát se rodičem a účastnit se genetických operací. [6], [7]

2.6.2 Výběr na základě vhodnosti

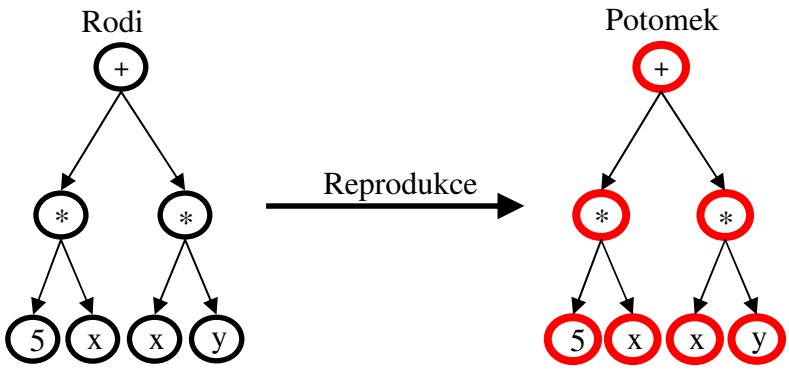
Při této metodě výběru jedinců se využívá vlastnosti normalizované vhodnosti. Jak již bylo zmíněno výše, součet normalizovaných vhodností všech jedinců v populaci je roven 1. Platí tedy, že $\sum_{i=1}^{N} F_N(i) = 1$. Náhodně se zvolí číslo $k \in \langle 0;1 \rangle$, následně se prochází populace a sčítají se normalizované vhodnosti jedinců, dokud je součet vhodností menší než zvolené číslo k. $F_N(j); \sum_{i=1}^{j} F_N(i) \leq k$. Při výběru na základě vhodnosti mají jedinci s lepší vhodností větší pravděpodobnost výběru než jedinci s menší vhodností. [6], [7]

2.7 Genetické operace

Genetické operace jsou procesy, při kterých vznikají (s výjimkou genetické operace reprodukce) noví jedinci. Volba genetického operátoru, který bude použit pro tvorbu nového jedince, je závislá na tzv. frakci. Frakce genetické operace určuje podíl jedinců, kteří budou vytvořeny danou genetickou operací. Genetické operátory jsou vzájemně výlučné, což znamená, že nemohou být aplikovány současně. Mezi genetické operace se řadí operace reprodukce, křížení a mutace. Cílem každé z těchto operací je vytvoření nového potomka. Křížení je typicky aplikováno s nejvyšší pravděpodobností, parametr frakce křížení (*crossover fraction*) určuje, kolik jedinců vznikne operací křížení. Nejčastěji je parametr frakce křížení volí $p_C \doteq 90\% = 0{,}9$ nebo více. Naproti tomu, parametr frakce mutace (*mutation fraction*) je mnohem menší, typicky $p_M \doteq 1\% = 0{,}1$. Pokud je $p_C + p_M < 1$, je použit také operátor reprodukce a parametr frakce reprodukce (*reproduction fraction*) se určí jako $p_R = 1 - (p_C + p_M)$. [6]

2.7.1 Reprodukce

Při reprodukci je pomocí jedné ze selekčních metod vybrán jeden jedinec z populace a jeho kopie je vložena do nové populace. Reprodukce jako jediná z genetických operací vytváří přímou kopii jedince, což vede k tomu, že jedinec má v další generaci větší šanci být vybrán jako rodič. [7]



Obr. 9.:Genetická operace - reprodukce

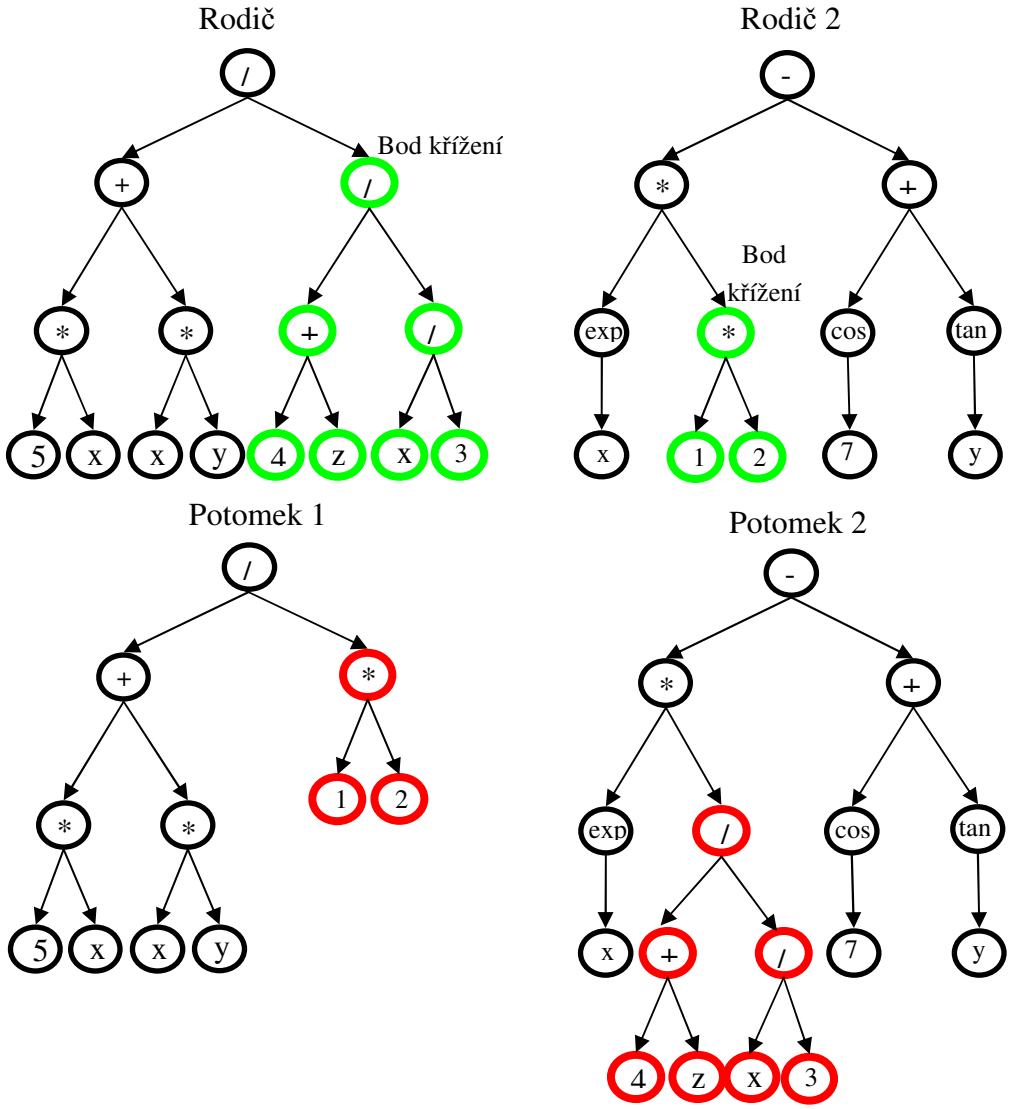
2.7.2 Křížení

Další genetickou operací je operace křížení. Během této operace vznikají ze dvou náhodně vybraných jedinců (rodičů) noví potomci, kteří mají strukturu složenou z obou rodičů. Při

křížení je běžné, že do nové populace bereme jen jednoho jedince. Je možné také definovat verzi křížení, kdy vznikají potomci dva, ale konečné rozhodnutí je záležitostí programátora. Body křížení nejsou vybírány s rovnoměrným rozdělením, protože při použití tohoto rozdělení dochází k výměně jen malého množství genetického materiálu. Koza proto doporučuje, aby z 90% docházelo ke křížení v místech funkcí (bodem křížení bude vnitřní uzel) a z 10% v místech terminálů (bodem křížení bude vnější uzel). Existují 3 typy křížení a to křížení podstromů, jednobodové křížení a arita-2 křížení. [7]

2.7.2.1 Křížení podstromů

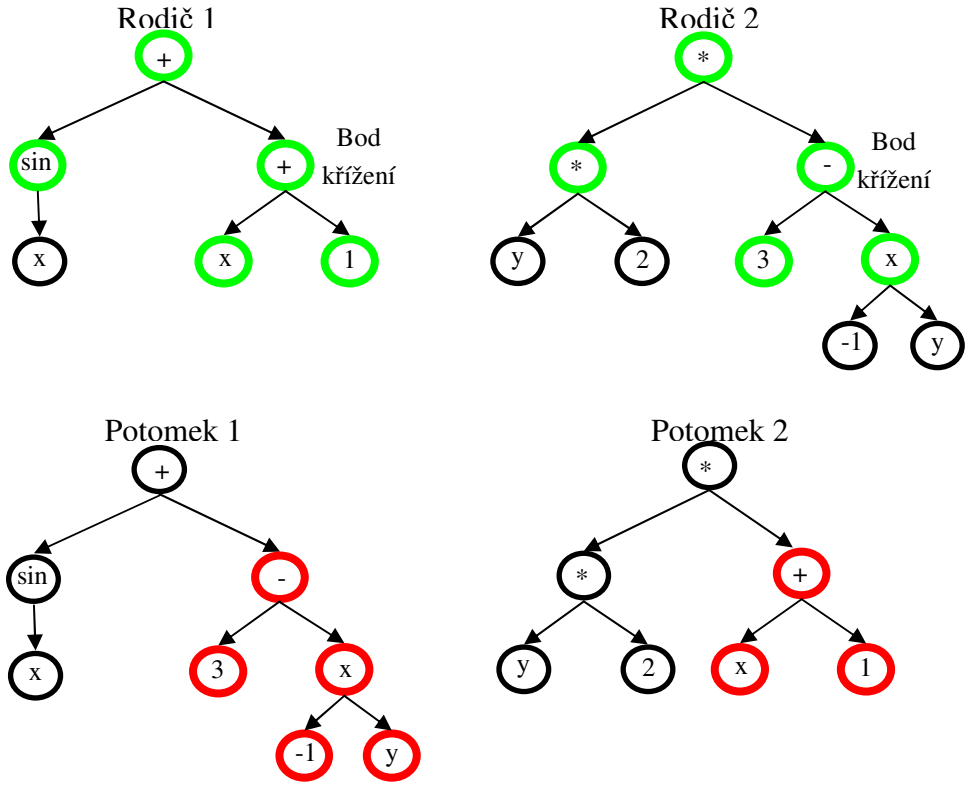
U tohoto druhu křížení se nejprve pomocí zvolené selekční metody vybere první rodič a dále se náhodně zvolí uzel, kde dojde ke křížení. Následně se vybere opět selekční metodou druhý rodič a také jeden náhodně zvolený uzel. Vybrané podstromy se v daných uzlech prohodí a vzniknou dva jedinci, kteří jsou následně vloženy do nové populace. Nejčastěji se však do další generace bere jen jeden potomek. Tato záležitost už je v rukou samotných tvůrců programu. [7]



Obr. 10.:Křížení podstromů

2.7.2.2 Jednobodové křížení

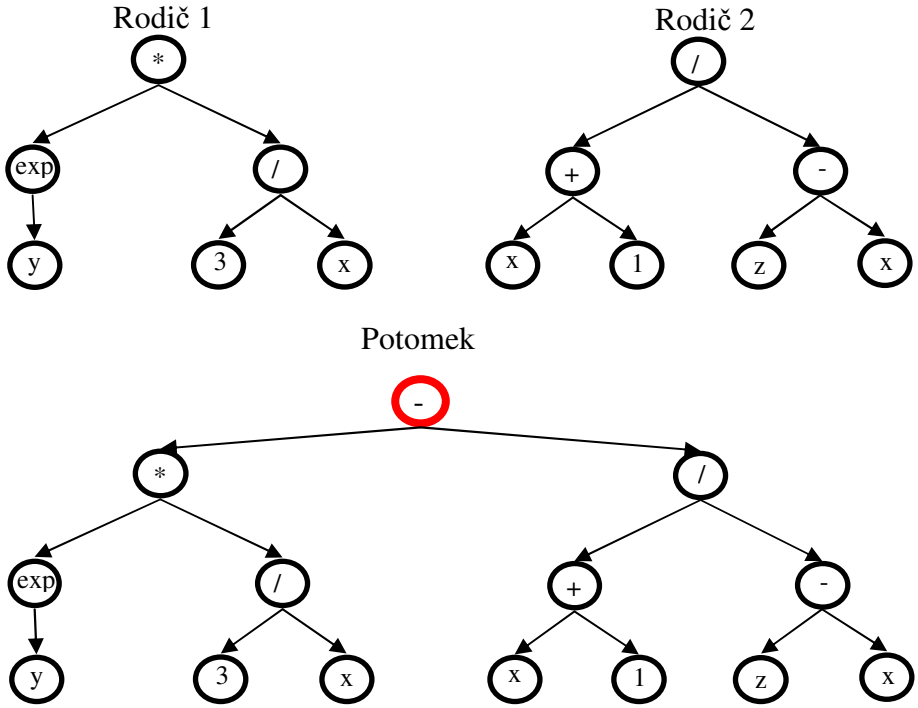
Dalším typem křížení je tzv. jednobodové křížení (*one-point crossover*). Při tomto druhu křížení se určí části stromů, které jsou z topologického hlediska společné pro oba jedince. V této části se vybere uzel, který je tudíž společný pro oba jedince a tento uzel se stane bodem křížení. Jednobodové křížení zaručuje, že nebude docházet k přílišnému růstu stromů u potomků. [7]



Obr. 11.:Jednobodové křížení

2.7.2.3 Arita-2 křížení

Posledním typem křížení je tzv. arita-2 křížení. Princip spočívá v náhodném výběru funkce s aritou 2 z množiny funkcí. Jako první argument bude sloužit první rodič, druhým argumentem bude druhý rodič. [7]



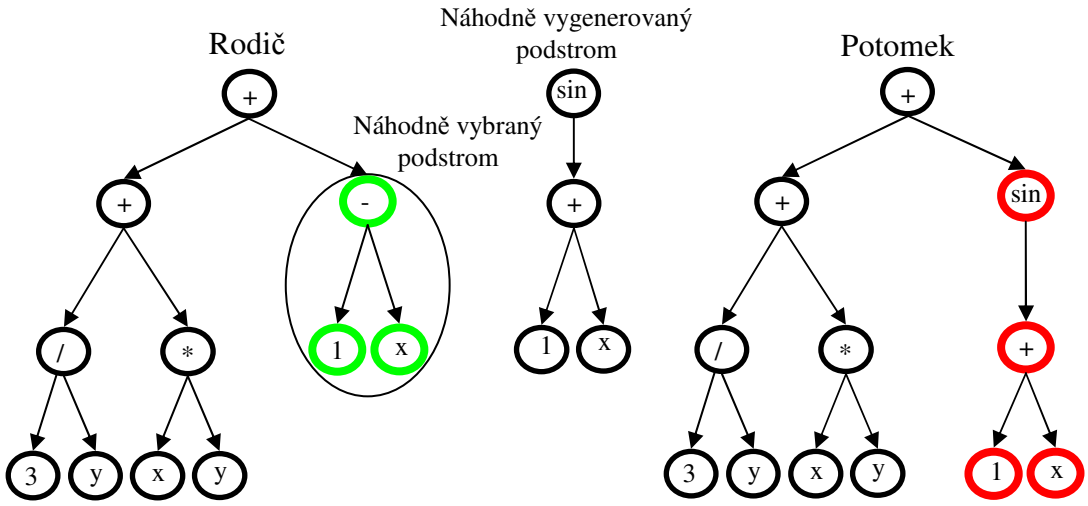
Obr. 12.:Arita-2 křížení

2.7.3 Mutace

Genetická operace mutace pracuje pouze s jedním jedincem. Není příliš používaná, jelikož při této operaci dochází k náhodným změnám struktury jedince a to má ve většině případů za následek zhoršení vhodnosti nově vzniklého jedince. Dalším důvodem je, že při častějším aplikování mutace by algoritmus měl charakter náhodného prohledávání. Existuje mnoho typů mutací, nicméně nejčastěji používané jsou mutace podstromu, mutace podstromu se zachováním velikosti, bodová mutace, mutace konstanty, systematická mutace konstant, vyzvedávající mutace, smršťující mutace a permutace. [7]

2.7.3.1 Mutace podstromu

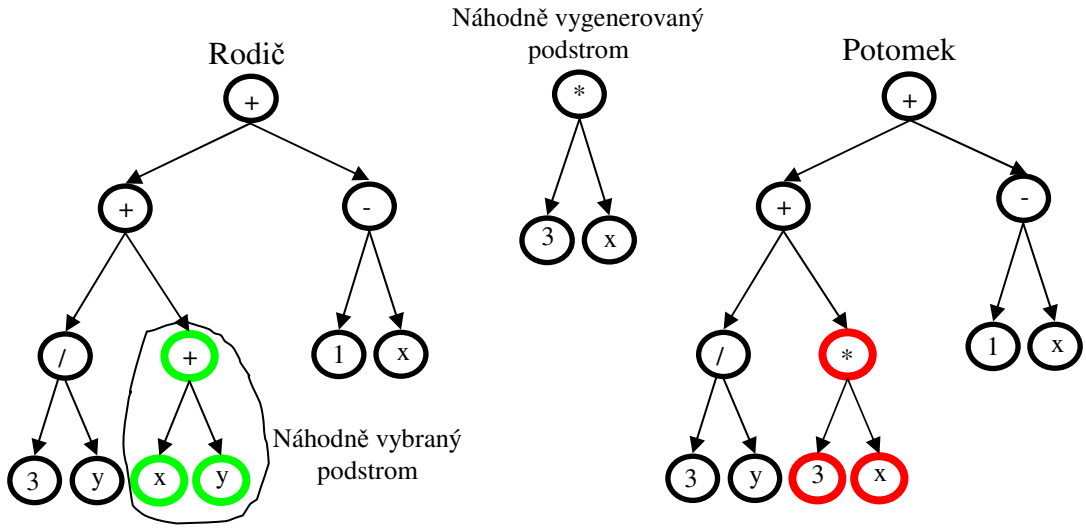
Nejvíce používaný druh mutace je tzv.mutace podstromu (*subtree mutation*). Při této mutaci je náhodně vybraný podstrom nahrazen jiným náhodně vytvořeným podstromem. [7]



Obr. 13.:Mutace podstromu

2.7.3.2 Mutace podstromu se zachováním velikosti

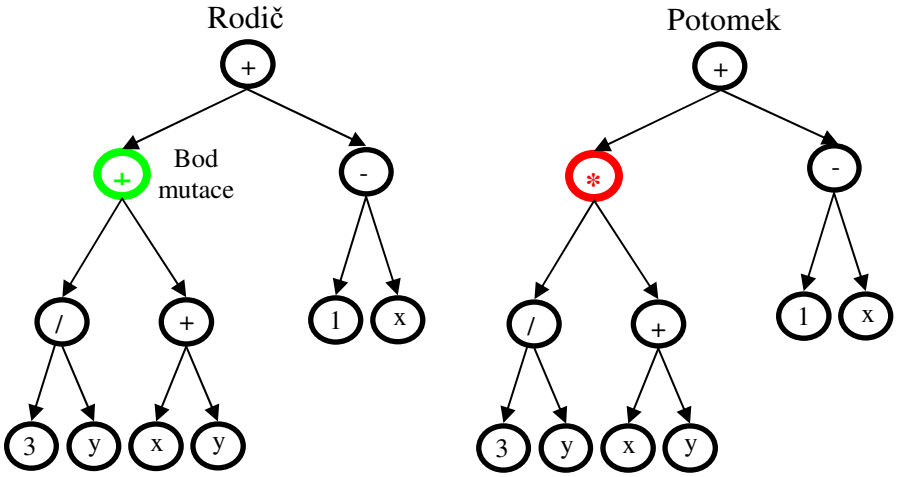
Mutace podstromu se zachováním velikosti nově vzniklého jedince (*size-fair subtree mutation*) je téměř totožná s předchozím typem mutace. Náhodně vybraný podstrom je nahrazen jiným náhodně vytvořeným podstromem, přičemž se koriguje jeho velikost. Velikost d_n náhodně vytvořeného podstromu je buď stejná jako u nahrazovaného podstromu nebo je volena v rozsahu $d_n \in (\frac{d_s}{2}, \frac{3d_s}{2})$, kde d_s je velikost nahrazovaného podstromu. [7]



Obr. 14.:Mutace podstromu se zachováním velikosti

2.7.3.3 Bodová mutace

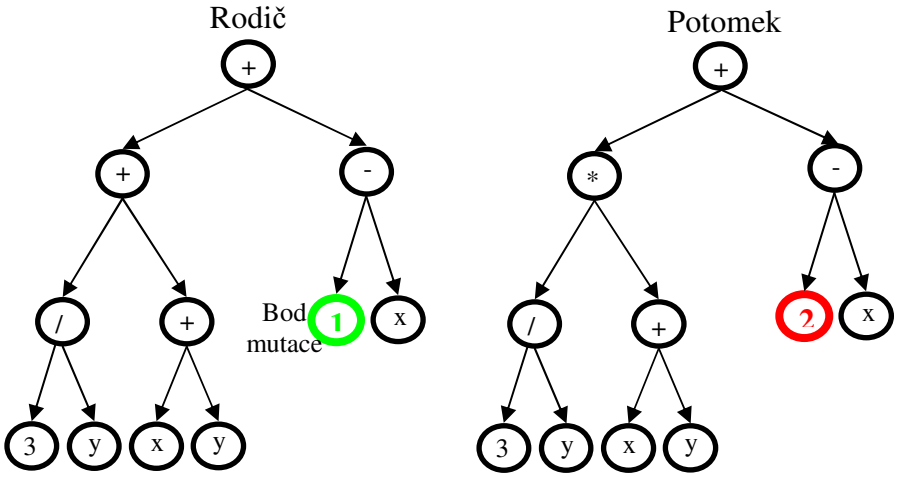
Dalším typem mutace je tzv. bodová mutace (*point mutation*). Při této mutaci je náhodně vybraný uzel obsahující prvek z množiny funkcí nahrazen jiným prvkem z této množiny. Nahrazované funkce musí mít stejný počet argumentů. Argumenty funkce zůstávají neměnné. [7]



Obr. 15.:Bodová mutace

2.7.3.4 Mutace konstanty

Při této mutaci dochází k tomu, že se náhodně změní hodnota konstanty jednoho z terminálů. Případně dochází k mutaci konstanty přidáním náhodného šumu podle Gaussova rozdělení. [7]



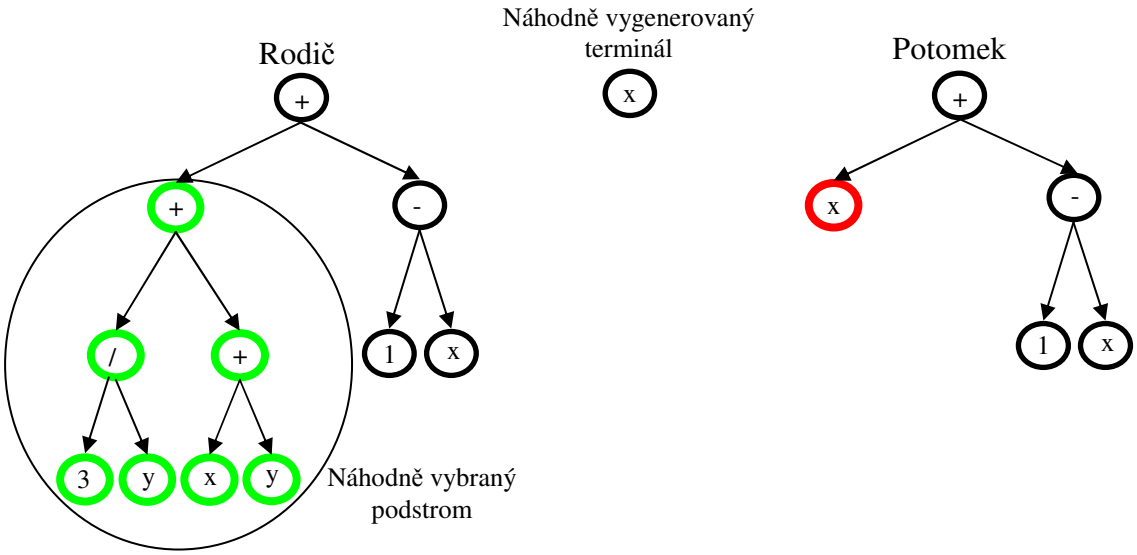
Obr. 16.:Mutace konstanty

2.7.3.5 Systematická mutace konstant

Tato mutace se používá jen ke konci evoluce. Je zde snaha vyladit konstanty nějakou optimalizační metodou. Například nelineární metodou nejmenších čtverců nebo pomocí simulovaného žíhání. [7]

2.7.3.6 Vyzvedávající mutace

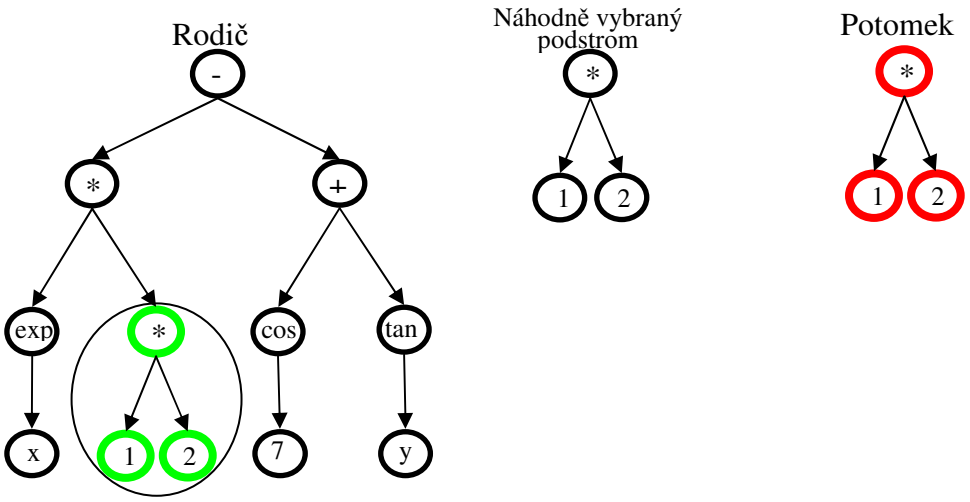
Jiným typem mutace, který je používán je tzv. vyzvedávající mutace (*shrink mutation*). Při této mutaci je náhodně vybraný podstrom nahrazen náhodně vybraným terminálem. Hlavním cílem této mutace je snížení velikosti určité větve jedince. Ve většině případů dochází ke snížení velikosti celého jedince. [7]



Obr. 17.:Vyzvedávající mutace

2.7.3.7 Smršťující mutace

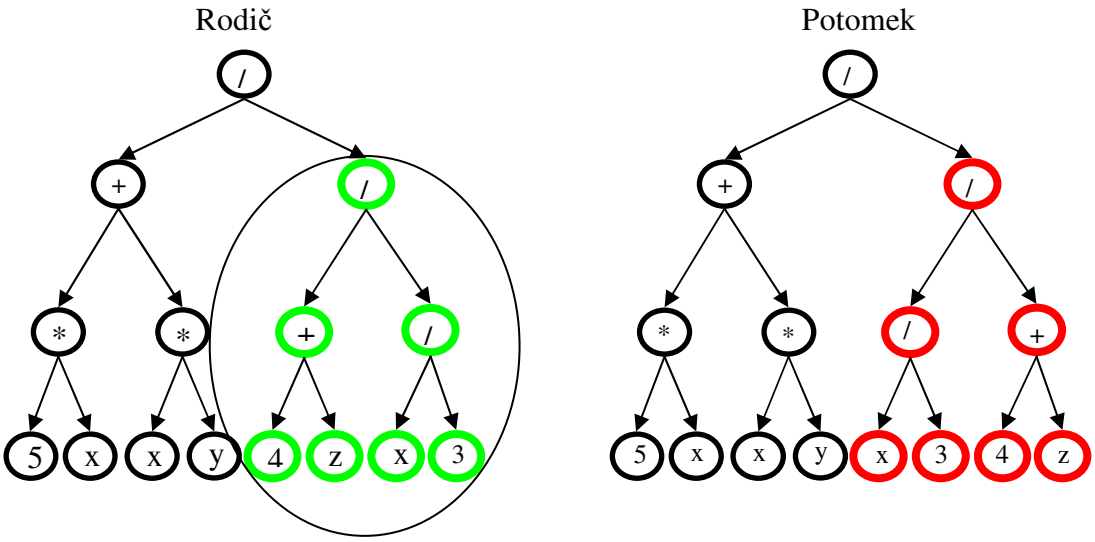
Dalším typem mutace je tzv. smršťující mutace (*hoist mutation*). Při této mutaci vzniká nový jedinec, který je kopií náhodně vybraného podstromu svého rodiče. Potomek bude tudíž menší a bude mít menší počet uzlů. [7]



Obr. 18.:Smršťující mutace

2.7.3.8 Permutace

Posledním typem mutace, o kterém bude zmínka, je tzv. permutace. Při permutaci je náhodně vybrán uzel, který obsahuje funkci a u této funkce dojde k permutaci argumentů. Je logické, že vybraná funkce nesmí být komutativní, jinak by tato mutace neměla žádný efekt a potomek by byl čistou kopií rodiče. [7]



Obr. 19.:Permutace

2.8 Bloat efekt

Z podstaty GP jakožto metody, která na rozdíl od klasických genetických algoritmů (GA) pracuje s jedinci proměnné délky a struktury, vyplívá, že může docházet k neúměrnému

nárůstu velikosti zpracovávaných stromových struktur. Tento jev se nazývá *bloat efekt* (bobtnání). Bloat efekt se projevuje vzrůstem komplexity vyvíjených programů. Generovaná řešení mohou obsahovat části kódů, které nic nekonají. Kód s větší komplexitou většinou dostatečně nezobecňuje (projevuje se patologické chování). Bloat efekt jde proti snaze získat efektivní řešení. Velikost nadbytečného kódu roste s počtem generací. [11], [12]

2.8.1 Odstranění blat efektu

Existuje několik způsobů, jak udržet rozumnou velikost vyvíjených programů po celou dobu výpočtu a částečně tak zamezit vzniku bloat efektu. Mezi využívané způsoby patří omezení na počet uzlů nebo hloubku generovaných řešení. Jiným způsobem, jak se bránit nežádoucímu bobtnání kódu, je zavedení vícekriteriální fitness, kdy se při ohodnocení řešení bere v potaz nejen kvalita daného řešení, ale i jeho komplexita. Dalším způsobem je archivace vhodných jedinců za každou generaci, kaskádování (reinicializace-udržuje diverzitu) nebo používání vyvážených (inteligentních) operátorů. [4], [5]

2.9 Parametry algoritmu genetického programování

Stejně jako jiné evoluční algoritmy, tak i algoritmus GP má několik řídících parametrů, jejichž nastavení významně ovlivňuje běh algoritmu a dosažený výsledek. Parametry algoritmu GP můžeme rozdělit na hlavní a vedlejší.

2.9.1 Hlavní parametry

Hlavní parametry mají vliv na rychlost běhu algoritmu a na kvalitu výsledného řešení. Hlavní parametry jsou:

- **Generations** (počet generací). Tento parametr určuje, po kolik generací algoritmus běží. Čím větší je počet generací, tím větší je i pravděpodobnost dosažení vyhovujícího výsledku.
- **PopSize** (velikost populace). PopSize značí počet jedinců v populaci. Velikost populace se stanovuje na základě složitosti řešeného problému. Čím větší je složitost problému, tím větší populace se volí. Velký počet jedinců v populaci ovšem sebou

nese nevýhodu v podobě delšího běhu algoritmu. Koza doporučuje volit velikost populace v rozmezí $PopSize \in \langle 300;10000 \rangle$ jedinců.

- **Depth** (hloubka). Tento parametr značí maximální hloubku syntaktického stromu, kterým je reprezentován každý jedinec v počáteční populaci. Koza doporučuje hloubku $Depth = 6$.
- **Functions** (množina funkcí). Tento parametr obsahuje funkce s jedním a více parametry, přičemž typy funkcí se volí v závislosti na řešeném problému
- **Terminals** (množina terminálů). Je to parametr obsahující proměnné, konstanty nebo funkce bez argumentů. Stejně jako u množiny funkcí, tak i zde se terminály volí v závislosti na řešeném problému
- **FrMutace** (frakce mutace). Parametr frakce mutace značí, kolik procent jedinců v nové generaci má vzniknout pomocí genetického operátoru mutace. Koza doporučuje volit tento parametr $FrMutace \in \langle 0;0,1 \rangle$.
- **FrKrizeni** (frakce křížení). Parametr frakce křížení značí, kolik procent jedinců v nové generaci má vzniknout pomocí genetického operátoru křížení. Koza doporučuje volit tento parametr $FrKrizeni \in \langle 0,8;0,95 \rangle$.
- **FrReprodukce** (frakce reprodukce). Parametr frakce reprodukce značí, kolik procent jedinců v nové generaci má vzniknout pomocí genetického operátoru reprodukce. Koza doporučuje volit tento parametr $FrRepodukce \in \langle 0;0,05 \rangle$.

2.9.2 Vedlejší parametry

Vedlejší parametry slouží především k tomu, aby nedocházelo k tzv.bloat efektu a tudíž ke zvýšení výpočetního času a vytváření jedinců extrémní velikosti. Vedlejšími parametry jsou:

DepthAfterCrossover (hloubka jedince po křížení). Tímto parametrem se ovlivňuje hloubka syntaktického stromu jedince vzniklého křížením.

LeafCountAfterCrossover (počet listů jedince po křížení). Nastavením tohoto parametru se ovlivňuje počet listů jedince vzniklého křížením.

DepthAfterMutation (hloubka jedince po mutaci). Tímto parametrem se ovlivňuje hloubka syntaktického stromu jedince vzniklého mutací.

LeafCountAfterMutation (počet listů jedince po mutaci). Nastavením tohoto parametru se ovlivňuje počet listů jedince vzniklého mutací.

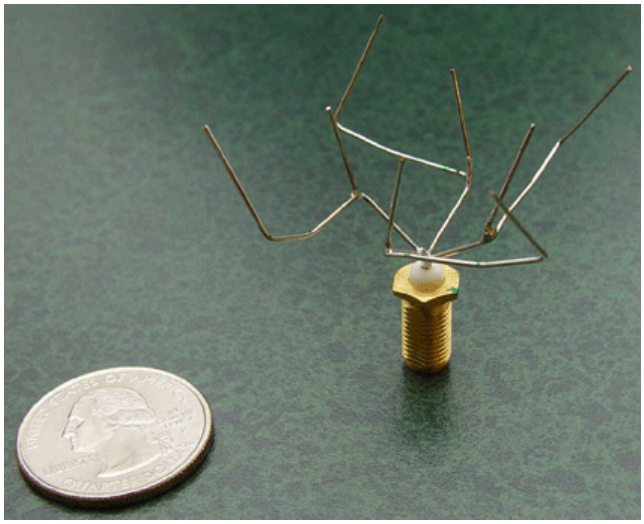
LeafPst (pravděpodobnost výběru vnitřního uzlu). Koza svými pokusy zjistil, že pokud je jako bod křížení volen oproti vnějšímu uzlu vnitřní uzel s vyšší pravděpodobností, má to za následek rychlejší průběh evoluce. Zároveň častější výběr vnitřního uzlu snižuje riziko vzniku tzv.bloat efektu. Koza doporučuje volit vnitřní uzel jako bod křížení s pravděpodobností $LeafPst = 0{,}9$.

2.10 Využití

Genetického programování nachází uplatnění v mnoha odvětvích lidské a technické činnosti. Jako příklad lze uvést:[7]

- symbolická regrese,
- fitování a modelování dat
- návrh elektrických obvodů,
- zpracování obrazu a signálu,
- predikce časových řad a ekonomických modelů,
- řízení průmyslových procesů,
- lékařství, biologie a bioinformatika,
- umělá inteligence v počítačových hrách,
- komprese obrazu, zvuku nebo dat,
- fraktály a teorie chaosu

Za zmínku stojí, že pomocí GP byly navrženy např. elektrické obvody pro časově optimální řízení robota, PID a PID-D2 regulátory, třídící síť pro 7 prvků během 16 kroků, elektronický teploměr, dolno-propustný filtr, anténa vyvíjená pro NASA (Obr.20) nebo kompresní algoritmy. Dále se GP často používá pro návrh optimální struktury neuronových sítí. [7]



Obr. 20.:Anténa navržená pomocí GP

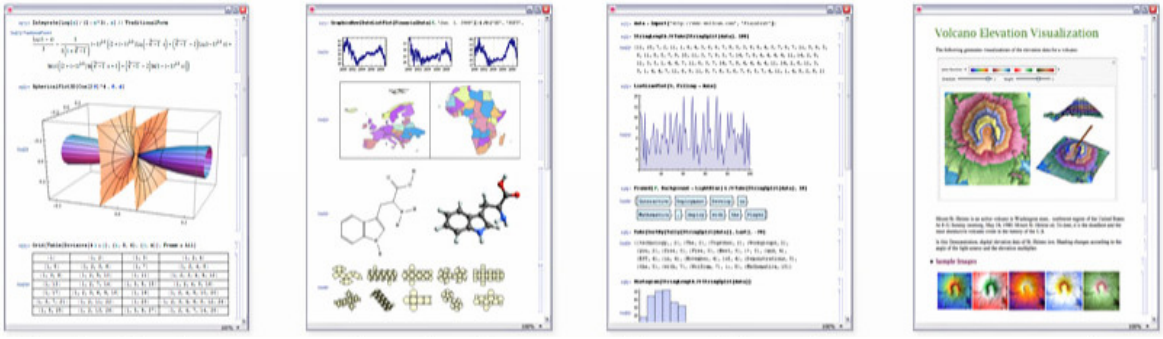
II. PRAKTICKÁ ČÁST

3 PROSTŘEDÍ MATHEMATICA

Mathematica je světově nejproslulejší a v současnosti s největší pravděpodobností také nejlepší výpočetní systém vytvořený společností Wolfram Research. Poprvé byl uveden 23. června 1988. Autorem softwaru Mathematica je britský matematik Stephen Wolfram. Hlavní myšlenkou, která stála za vznikem Mathematici, bylo vytvořit jeden samostatný systém, který by dokázal zvládnout všechny aspekty technického programování. To se podařilo díky vytvoření nového typu symbolického počítačového jazyku využívajícího prvky umělé inteligence. V dnešní době můžeme pojem Mathematica chápat jako programovací jazyk, vývojové prostředí nebo jako obrovskou znalostní databázi. [10]

V dnešní době je Mathematica používána v matematice, fyzice, biologii, chemii, geografii, ekonomii nebo informatice. Je používaná při výuce na středních školách a univerzitách po celém světě nebo na všech 15 ministerstvech americké vlády. [10]

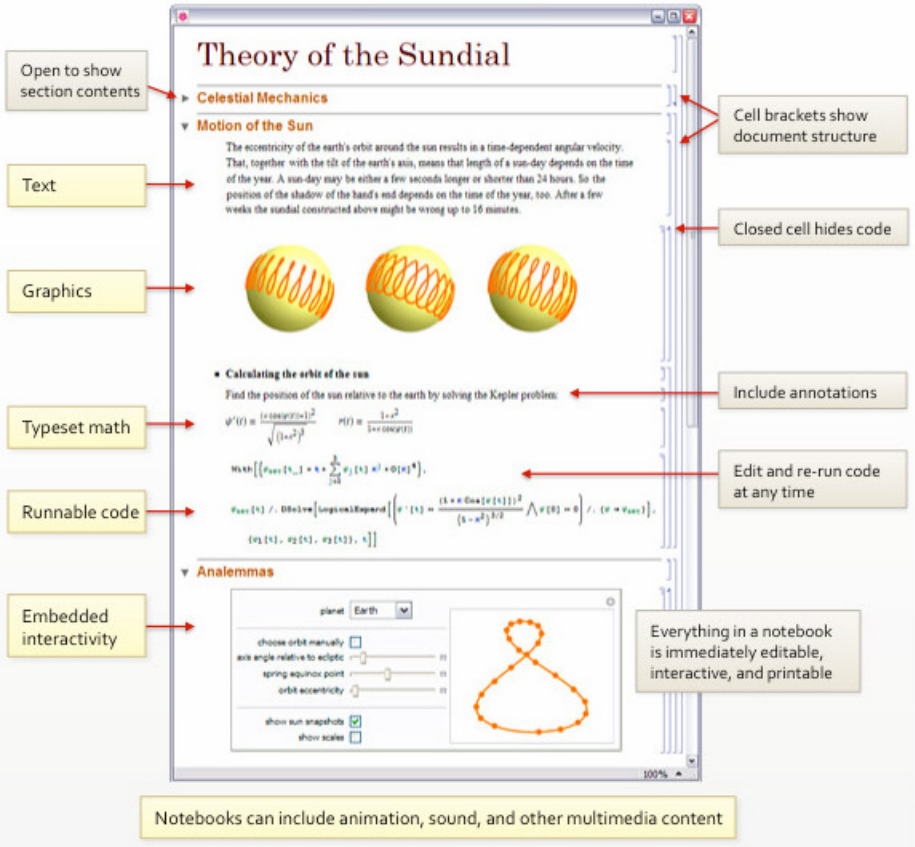
V Matemathice lze provádět okamžité výpočty a vizualizace, přistupovat přímo do hlavní databáze nebo prostřednictvím Internetu ke vzdáleným databázím, vytvářet záznamy práce, vytvářet interaktivní dokumenty a prezentace, psát programy v jazyce Mathematica nebo vytvářet rozsáhlé aplikace. Mathematica pracuje s daty jakéhokoliv typu, např. s chemickými daty, geografickými, socioekonomickými, finančními nebo astronomickými. Dále je možné automaticky importovat nebo exportovat data v jednom z více než 200 podporovaných formátů, připojovat se k externím programům, databázím nebo webovým stránkám, případně odesílat data přímo z programu na e-mail. [10]



Obr. 21.:Prostředí Mathematica [10]

Při práci v Matematice se používá soubor, který se nazývá notebook. Tento soubor automaticky organizuje všechno do spustitelného, interaktivního dokumentu. Základním objektem notebooku je buňka, do níž se píší příkazy. Notebook obsahuje buňky, které ukazují strukturu dokumentu. Každý výpočet má svou vstupní a výstupní buňku.

Požadavek na výpočet buňky se předává jádru, tzv. Kernelu, které požadavek vyhodnotí a vykoná. Notebook může obsahovat text, grafiku, matematický vzorec, spustitelný kód, animaci, zvuk nebo video. [10]



Obr. 22.:Notebook [10]

Mathematica podporuje paralelní výpočty a obsahuje rozhraní pro komunikaci s jinými technickými programy jako MATLAB, LabView a jinými. S využitím speciálního protokolu *MathLink* umožňuje Mathematica volat své funkce i z jiných programovacích jazyků, a to C, .NET, Java, Haskell nebo Visual Basic. Mathematica je dostupná pro 32 a 64bitové architektury procesorů a platformy Windows, Linux, MAC OS X a Solaris. [10]

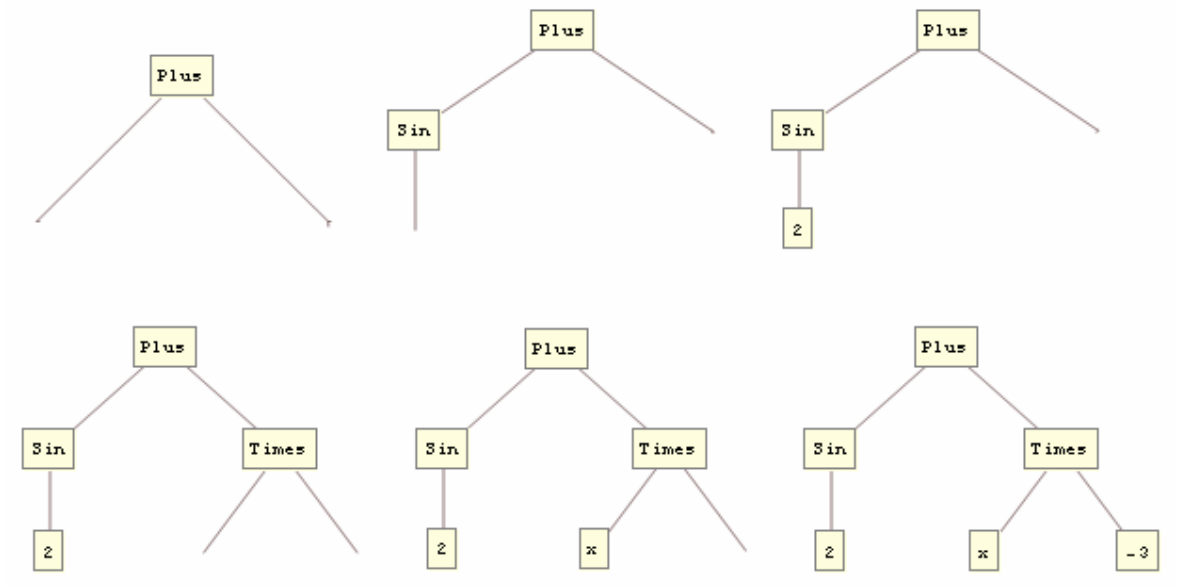
4 POPIS PROGRAMU

Pro velkou obsáhlost celého programu zde uvádím výpis a popis jen některých funkcí. Kompletní zdrojový kód včetně detailního popisu všech funkcí je uveden v přiloženém souboru.

4.1 Generování jedinců

```
PopInit[depth_,method_String,op0_,preop0_,var0_]:=
  Block[{d=depth,op=op0,preop=preop0,var=var0,index,
    Composite,Arity},
   If[d>0,Composite=Join[op,preop,var];
    Arity=Join[ConstantArray[2,Length[op]],
      ConstantArray[1,Length[preop]],ConstantArray[0,Length[var]]],
    Composite=var;
    Arity=Table[0,{Length[var]}]];
   If[method=="Full",
    If[d>0,index=RandomInteger[{1,Length[op]+Length[preop]}];
     Switch[Arity[[index]],
      2,Composite[[index]][PopInit[d-1,method,op,preop,var],
      PopInit[d-1,method,op,preop,var]],
      1,Composite[[index]][PopInit[d-1,method,op,preop,var]]],
     index = RandomInteger[{1, Length[var]}];
     If[index==2,RandomReal[{-6,6}],Composite[[index]]]],
    If[method=="Grow",index=RandomInteger[{1,Length[Composite]}];
     Switch[Arity[[index]],
      0,If[index==Length[Composite],RandomReal[{-6, 6}],
     Composite[[index]]],
      1,Composite[[index]][PopInit[d-1,method,op,preop,var]],
      2,Composite[[index]][PopInit[d-1,method,op,preop,var],
    PopInit[d-1,method,op,preop,var]]]]]];
```

Funkce *PopInit[]* vytváří na základě zvolené množiny funkcí, množiny terminálů, metody generování a hloubky náhodného jedince. Vytváření syntaktického stromu probíhá zleva nahoře směrem vpravo dolů (Obr.23, Tab.1).



Obr. 23.:Vytváření jedince

Krok	Jedinec
1.	+
2.	Sin[]+
3.	Sin[2]+
4.	Sin[2]+ *
5.	Sin[2]+x*
6.	Sin[2]+x*(-3)

Tab.1.:Vytváření jedince

4.2 Vhodnost

Účelová funkce, která počítá ohodnocení daného jedince na základě zadaných dat, je definována následovně:

```
1 RawFitness[jedinec0_, data0_] :=
2   Quiet[With[{jedinec = jedinec0 /. DisTransform, data = data0},
3     If[Head[jedinec] === Real, suma = 9999,
4      TimeConstrained[
5       suma=Sum[Abs[N[(data[[i,2]]-(jedinec/.x->data[[i,1]])),8]],
6       {i, 1, Length[data]}], 2, suma = 9999]];
7     If[(Head[suma] === Real && suma =!= Overflow[]) ||
8       Head[suma] === Integer, suma,
9      9999]]];
```

Na řádku *5* je výpočet samotné účelové funkce, která je definována jako součet absolutních hodnot rozdílu skutečných a aproximovaných hodnot.

4.3 Výběr jedinců

Výběr jedinců může probíhat na základě turnajového výběru nebo na základě vhodnosti.

```
1 TournamentSelection[population_, NormFit_, nindividual_] :=
2   Module[{pop = population, nf = NormFit, ni = nindividual, ind},
3    ind = RandomSample[Range[Length[pop]], ni];
4    inds = Table[{ind[[i]], nf[[ind[[i]]]]}, {i, 1, Length[ind]}];
5    bestind =
6     pop[[inds[[Position[inds[[All,2]],Max[inds[[All,2]]],1,1]//
7      Flatten,1]][[1]]]];
8    bestind];
```

U turnajové selekce lze pomocí parametru *nindividual* zvolit počet jedinců, kteří se budou účastnit turnajové selekce.

```
1 FitnessProportional[population_, NormFit_] :=
2   Block[{pop = population, nf = NormFit, nof, suma},
3    nof = RandomReal[];
4    i = 1;
5    suma = 0;
6    While[suma < nof, suma += nf[[i]]; i++];
7    pop[[i - 1]]];
```

Obě funkce nevrací index, kde se v populaci nachází vybraný jedinec, ale vrací samotného jedince.

4.4 Genetické operace

4.4.1 Křížení

Funkce *Crossover[]* má implementované všechny tři typy křížení, a to křížení podstromů, jednobodové křížení a arita-2 křížení. Vstupem funkce jsou dva jedinci, reprezentující rodiče a výstupem jsou dva potomci. Typ křížení je vybírán náhodně. Níže uvádím zkrácený obsah funkce *Crossover[]* obsahující kód pro arita-2 křížení.

```
1 CrossOver[individual1_, individual2_, type_, lcm0_] :=
2   Block[{ind1=individual1, ind2 = individual2, t = type, lcm = lcm0,
3      off1 = Null, off2 = Null, cesty = {}},
4    If[t == 0(*nahodne vybrany typ*), t = RandomInteger[{1, 3}]];
5    If[t == 1,
6     off1=op[[RandomInteger[{1, Length[op]}]]][##] &[ind1, ind2], ]]
```

Tvorba potomka probíhá tak, že se náhodně zvolí funkce s aritou 2. Jako první argument vybrané funkce bude první rodič, druhým argument analogicky druhý rodič.

4.4.2 Mutace

Ve funkci *Mutation[]* jsem implementoval typy mutace - mutace podstromu, mutace podstromu se zachováním velikosti, bodová mutace, mutace konstanty, vyzvedávající mutace, smršťující mutace a permutace. Níže uvádím zkrácený obsah funkce *Mutation[]* obsahující kód pro mutaci konstanty.

```
1 Mutation[individual_, type0_, lcm0_] :=
2   Block[{ind = individual, t = type0, lcm = lcm0, off = Null},
3    If[t == 0(*nahodne vybrany typ*), type = RandomInteger[{1, 1}]];
4    Switch[t,
5     1,
6     uzly = Position[ind, _, Infinity, Heads -> False];
7     fuzly = {};
8     fuzly = Select[uzly, NumberQ[Extract[ind, #]] &];
9     If[Length[fuzly] > 0,
10     cesta = fuzly[[RandomInteger[{1, Length[fuzly]}]]];
11     number = RandomReal[{-4, 4}];
12     off = ReplacePart[ind, cesta -> number],
13     off = ind]];
14   off];
```

Mutace konstanty probíhá tak, že se nejprve zjistí, ve kterých uzlech jsou čísla. Následně je náhodně vybrán jeden z uzlů, obsahující konstantu a ta je nahrazena novou náhodně vygenerovanou konstantou.

4.4.3 Reprodukce

Funkce reprodukce *Reproduction[]* vytváří kopii vybraného jedince.

```
1 Reproduction[individual1_] := Block[{}, individual1];
```

5 TESTOVACÍ PŘÍKLADY

Testování slouží k ověření toho, zda je vytvořený algoritmus správný. Pro testování algoritmů GP jsou všeobecně uznávané a používané příklady z oblasti symbolické regrese, které jsou použity i v této práci pro otestování funkčnosti vytvořeného algoritmu. Mezi další vhodné testovací příklady patří tzv. umělý mravenec nebo predikce řad.

5.1 Symbolická regrese

Ve statistice pomocí regresní analýzy vyšetřujeme neznámou relaci $\varphi \subseteq \Re^n \mapsto \in \Re$ jedné závislé proměnné $y \in \Re$ na nezávislé proměnné $x \in \Re^m$. Jelikož neznáme φ, je cílem najít vhodnou aproximaci ψ^*. [9]

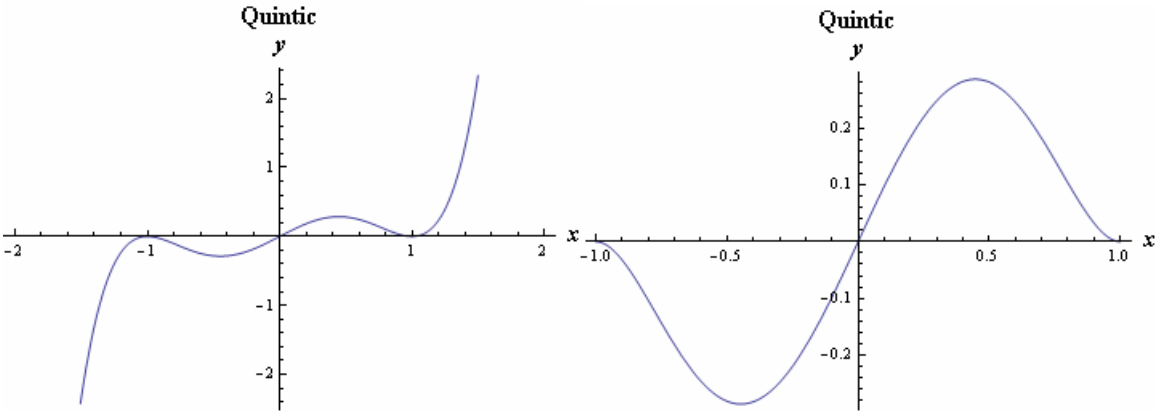
Regrese je statistická metoda používaná k předpovídání hodnoty nějaké proměnné, která je závislá na jedné nebo více nezávislých proměnných. [9]

Výsledkem regrese je funkce $\psi^* : \Re^m \mapsto \Re$. Funkce ψ^* je nejlepší odhad vybraný z množiny Ψ možných funkcí $\psi : \Re^m \mapsto \psi$. [9]

Symbolická regrese je jeden z možných způsobu, jak najít vhodnou regresní funkci pro zadaná data. Symbolická regrese není limitována určováním optimálních hodnot parametrů. Místo toho může být regresní funkce zkonstruována kombinací matematických výrazů, proměnných a konstant. V GP je cílová regresní funkce konstruována a upřesňována během evolučního procesu. Na začátku se v závislosti na velikosti populace a matematických výrazech, proměnných, respektive konstantách vytvoří počáteční populace, kde každý jedinec představuje jednu z možných regresních funkcí. Během evolučního procesu se vybírají nejvhodnější jedinci, kteří se vzájemně kříží, případně dochází k mutacím jedinců, dokud nenalezneme nejvhodnějšího jedince, tudíž nejvhodnější regresní funkci. [9]

5.1.1 Quintic

V tomto případě je hledaná funkce $f_1(x)$ ve tvaru $f_1(x) = x^5 - 2x^3 + x$ na intervalu $x \in \langle -1;1 \rangle$. Graf funkce $f_1(x)$ je na obrázku (Obr.24).



Obr. 24.:Graf fce $f_1(x)$ a fce $f_1(x)$ na intervalu <-1,1>

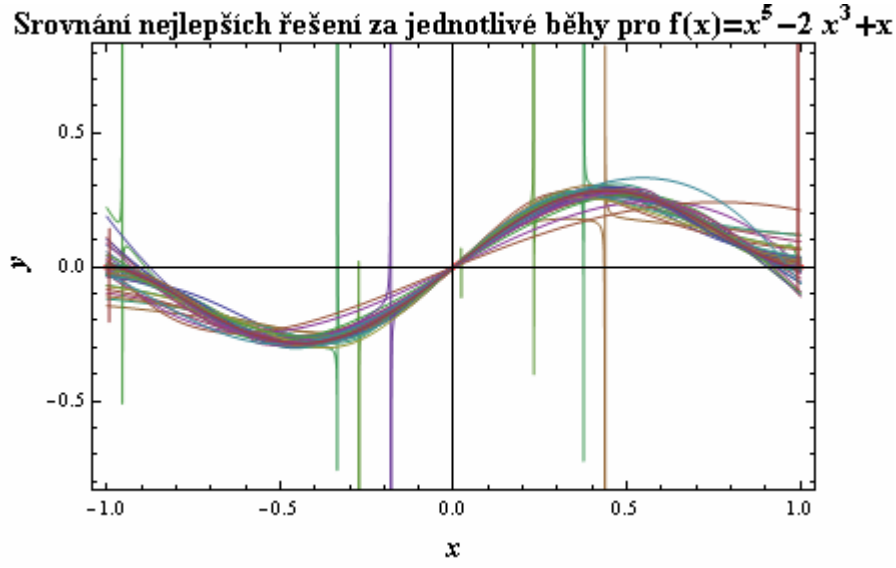
5.1.1.1 Nastavení parametrů

Parametry GP byly nastaveny následovně:

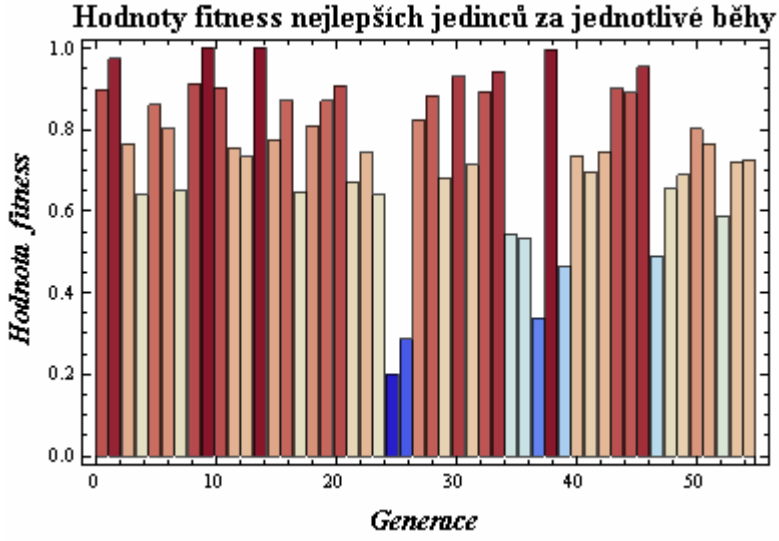
Parametr	Hodnota
PopSize	700
Generations	500
Runs	50
Functions	{Plus[], Subtract[], Times[], Divide[], Minus[]}
Terminals	{x, Random[-4,4]}
Method	HalfAndHalf
Depth	6
Leafcountcm	100
FrMutace	0.1
FrKrizeni	0.9
FrReprodukce	0

Tab.2.:Nastavení parametrů pro 1.testovací příklad

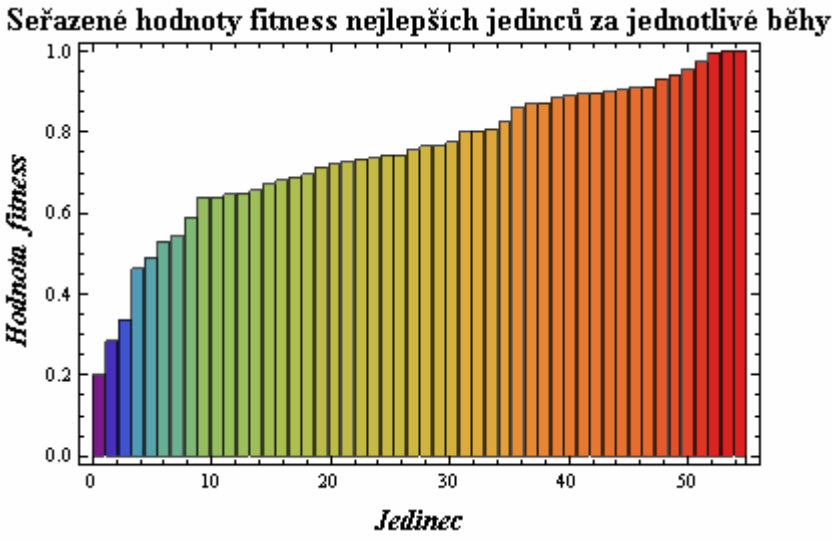
5.1.1.2 Vyhodnocení výsledků



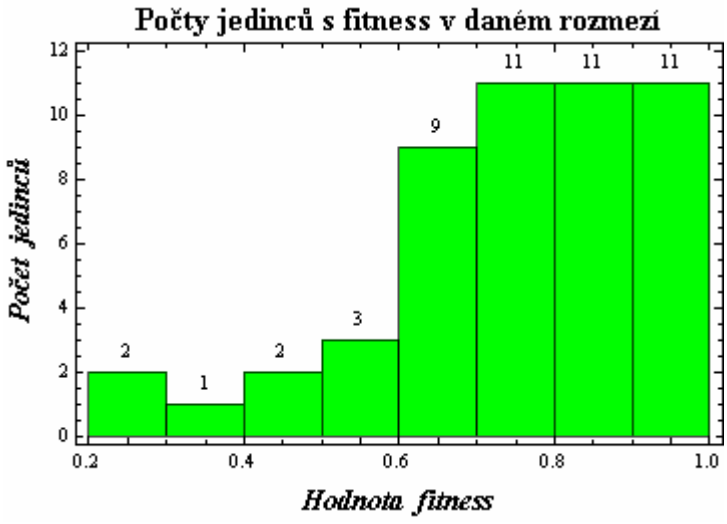
Obr. 25.:Nejlepší řešení za jednotlivé běhy, 1.testovací příklad



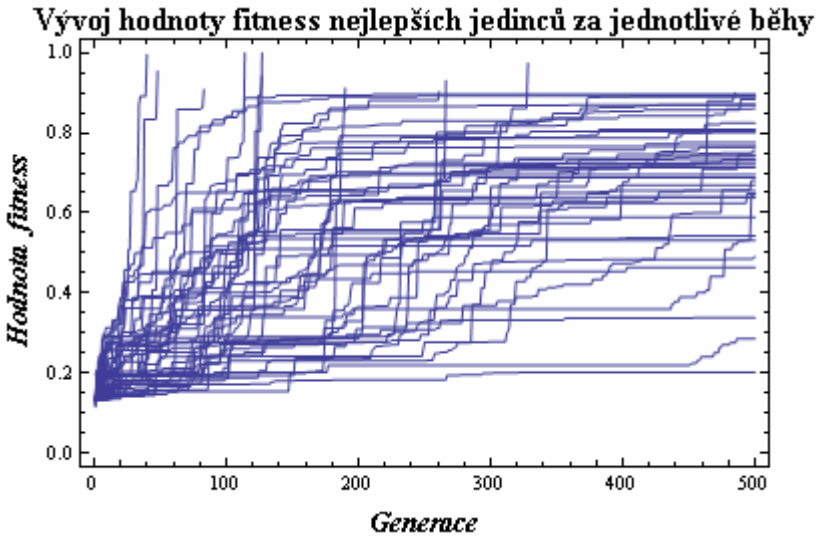
Obr. 26.:Nejlepší jedinci a fitness, 1.testovací příklad



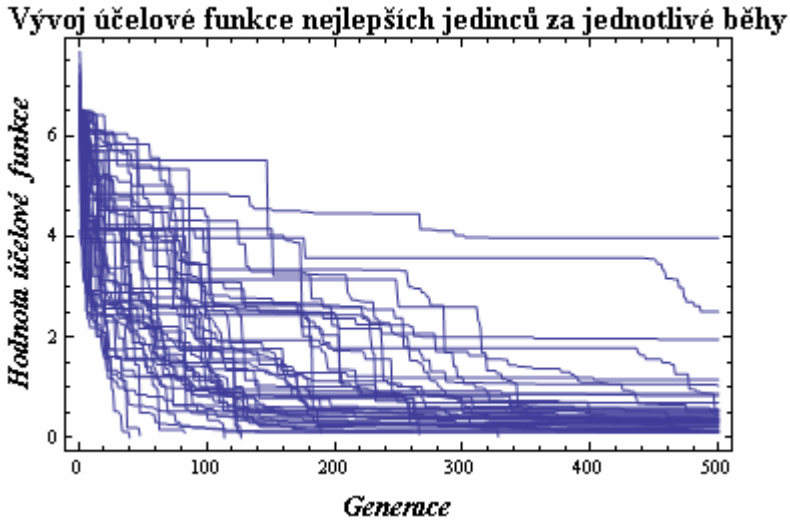
Obr. 27.:Seřazení nejlepších jedinců, 1.testovací příklad



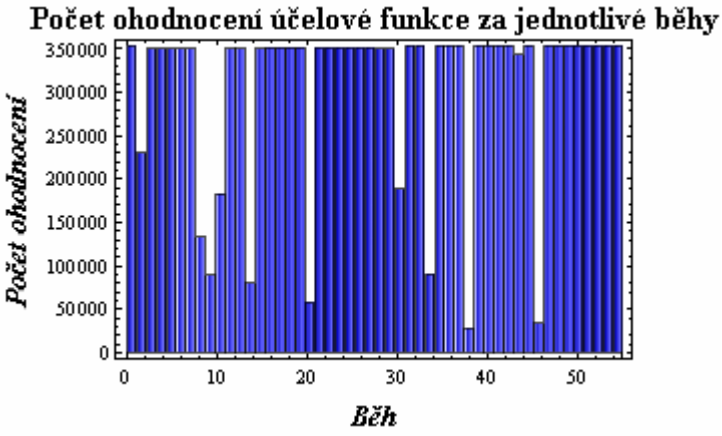
Obr. 28.:Porovnání hodnoty fitness a počtu jedinců, 1.testovací příklad



Obr. 29.:Vývoj hodnoty fitness nejlepších jedinců, 1.testovací příklad



Obr. 30.:Vývoj hodnoty účelové funkce nejlepších jedinců, 1.testovací příklad

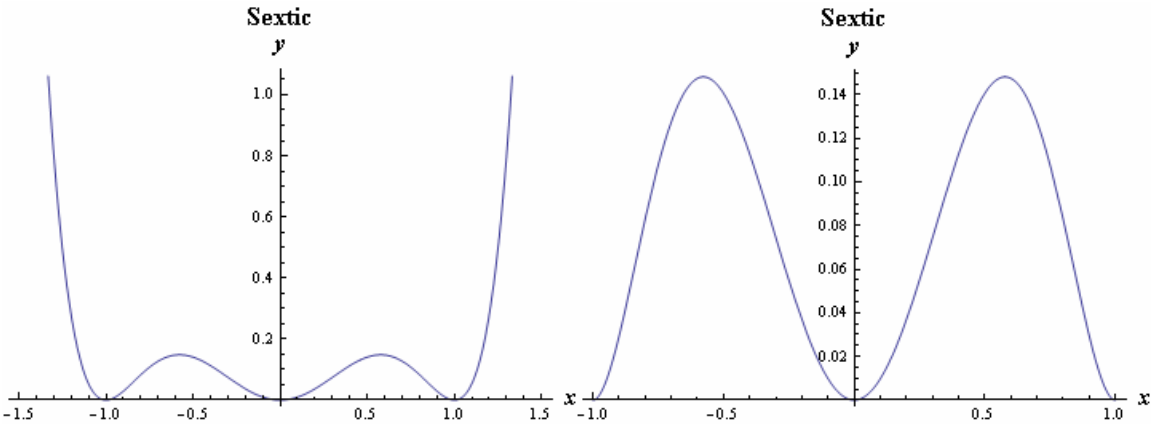


Obr. 31.:Počet ohodnocení účelové funkce, 1.testovací příklad

Jak je patrné z výsledků GP dokázalo ve většině případů najít dostatečně vyhovující regresní funkci. Ve dvou případech našlo samotnou zadanou funkci $f_1(x)$ a to ve tvaru: $f_1^1(x) = x - x^2 \cdot \left(2x - x^3\right)$ a $f_1^2(x) = x \cdot \left(-1 + x^2\right) \cdot \left(-1 + x^2\right)$. Přibližně v pěti případech GP při hledání uvízlo v lokálních minimech, které byly z hlediska hodnoty účelové funkce zcela nevyhovující. Celkový počet ohodnocení účelové funkce za 50 běhů byl 15212400.

5.1.2 Sextic

Hledaná funkce $f_2(x)$ má tvar $f_2(x) = x^6 - 2x^4 + x^2$, interval hledání je stejný jako v předchozím příkladě, tedy $x \in \langle -1;1 \rangle$. Graf funkce $f_2(x)$ je na obrázku (Obr.32).



Obr. 32.:Graf fce $f_2(x)$ a $f_2(x)$ na intervalu <-1,1>

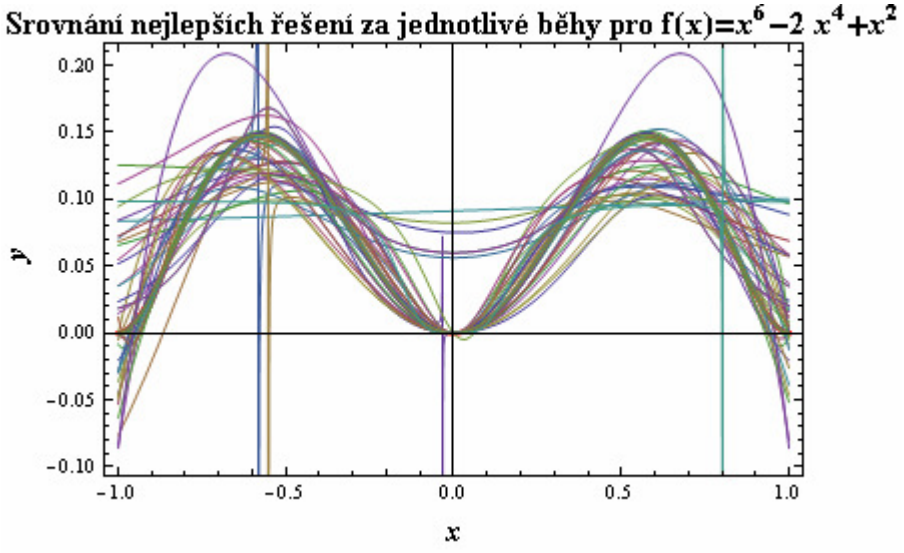
5.1.2.1 Nastavení parametrů

Parametry GP byly nastaveny následovně:

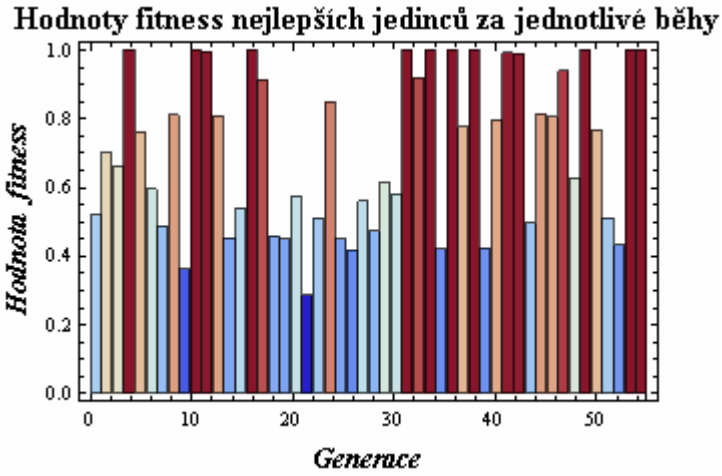
Parametr	Hodnota
PopSize	600
Generations	500
Runs	50
Functions	{Plus[], Subtract[], Times[], Divide[], Minus[]}
Terminals	{x, Random[-4,4]}
Method	HalfAndHalf
Depth	6
Leafcountcm	200
FrMutace	0.1
FrKrizeni	0.9
FrReprodukce	0

Tab.3.:Nastavení parametrů pro 2.testovací příklad

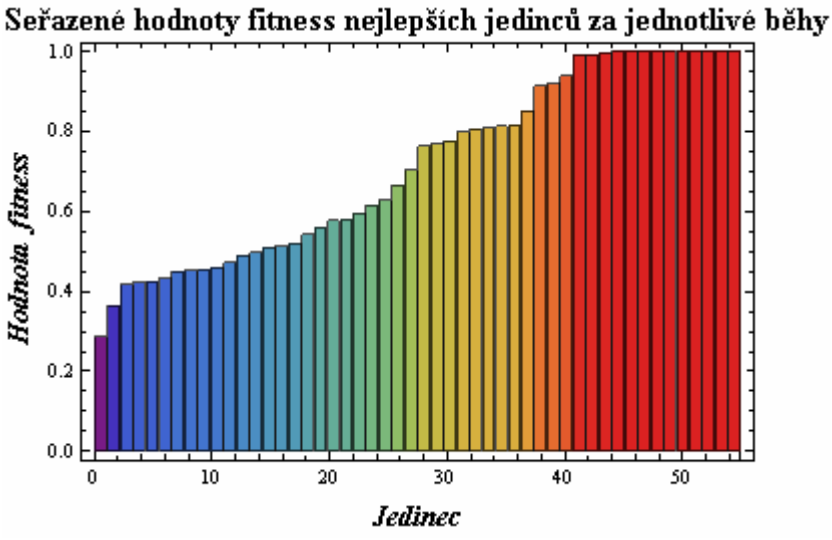
5.1.2.2 Vyhodnocení výsledků



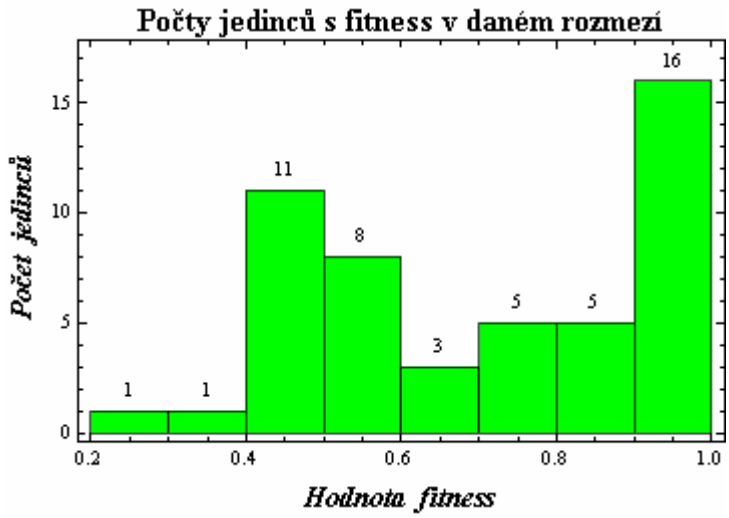
Obr. 33.:Nejlepší řešení za jednotlivé běhy, 2.testovací příklad



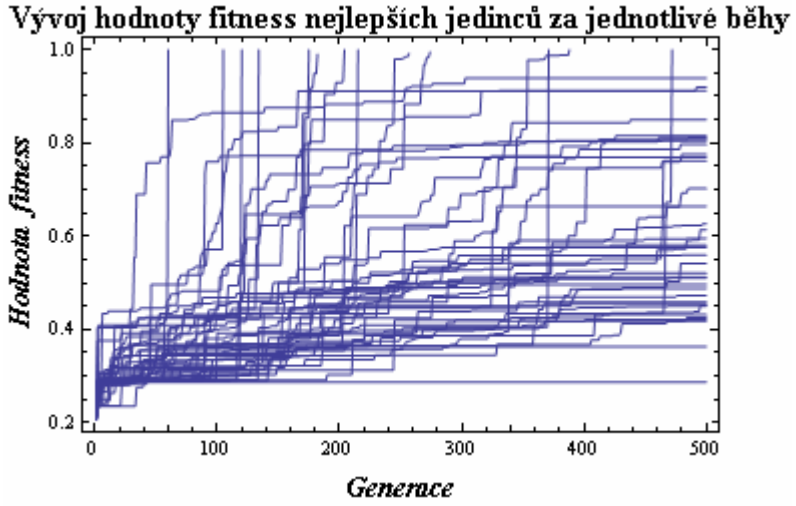
Obr. 34.:Nejlepší jedinci a fitness, 2.testovací příklad



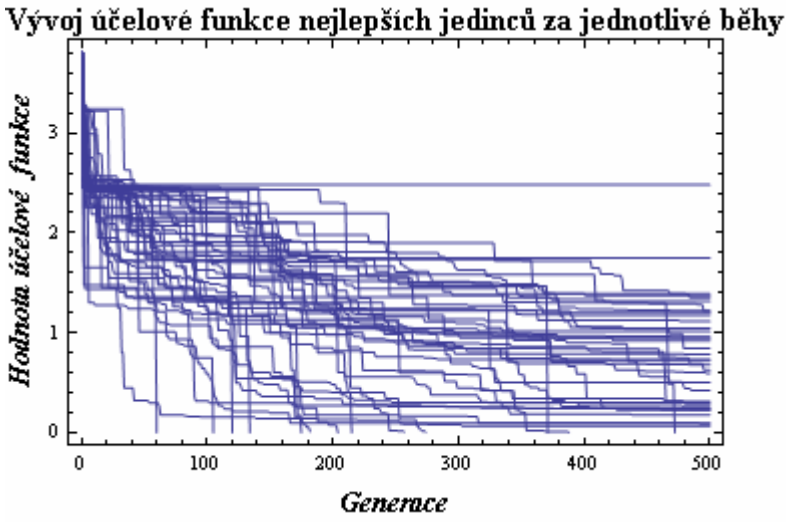
Obr. 35.:Seřazení nejlepších jedinců, 2.testovací příklad



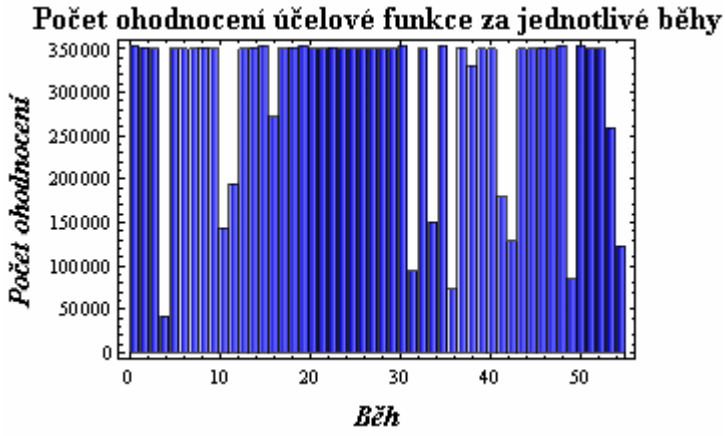
Obr. 36.:Porovnání hodnoty fitness a počtu jedinců, 2.testovací příklad



Obr. 37.:Vývoj fitness nejlepších jedinců, 2.testovací příklad



Obr. 38.:Vývoj hodnoty účelové funkce nejlepších jedinců, 2.testovací příklad

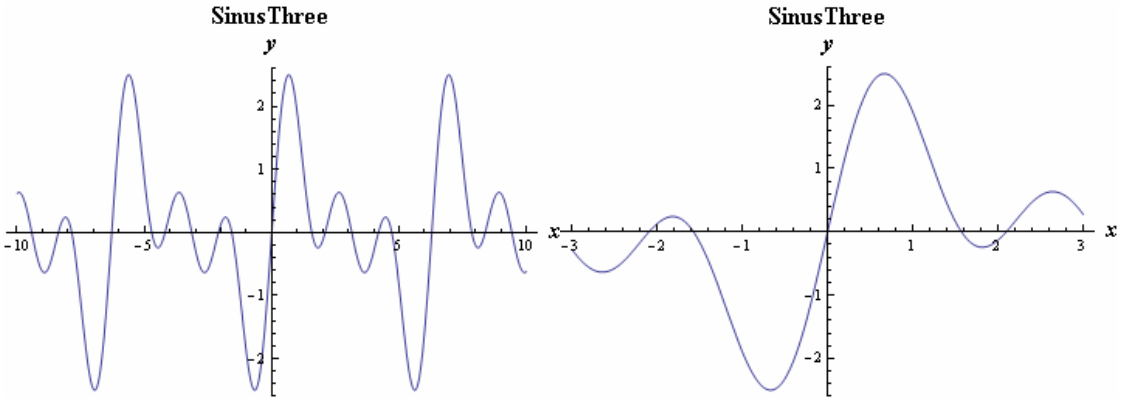


Obr. 39.:Počet ohodnocení účelové funkce, 2.testovací příklad

Ve druhém testovacím příkladu bylo pomocí GP během 50 běhů nalezeno 26 dostatečně vyhovujících funkcí. V desíti případech našlo GP samotnou zadanou funkci $f_2(x)$ a to v jednom z následujících tvarů: $f_2^1(x) = \left(x - x^3\right) \cdot \left(x - x^3\right)$, $f_2^2(x) = \left(-x + x^3\right) \cdot \left(-x + x^3\right)$, $f_2^3(x) = x \cdot \left(1 - x^2\right) \cdot \left(x - x^3\right)$, $f_2^4(x) = x^2 \cdot \left(-1 + x^2\right) \cdot \left(-1 + x^2\right)$ a $f_2^5(x) = -1 \cdot \left(x - x^3\right) \cdot \left(-x + x^3\right)$. Přibližně ve 13 případech GP při hledání uvízlo v lokálních minimech, které byly z hlediska hodnoty účelové funkce zcela nevyhovující. Celkový počet ohodnocení účelové funkce za 50 běhů byl 15075600.

5.1.3 SinusThree

Dalším příkladem je funkce $f_3(x)$ ve tvaru $f_3(x) = \sin(x) + \sin(2x) + \sin(3x)$, interval $x \in \langle -3;3 \rangle$. Graf funkce $f_3(x)$ je na obrázku (Obr.40).



Obr. 40.:Graf fce $f_3(x)$ a fce $f_3(x)$ na intervalu <-3,3>

5.1.3.1 Nastavení parametrů

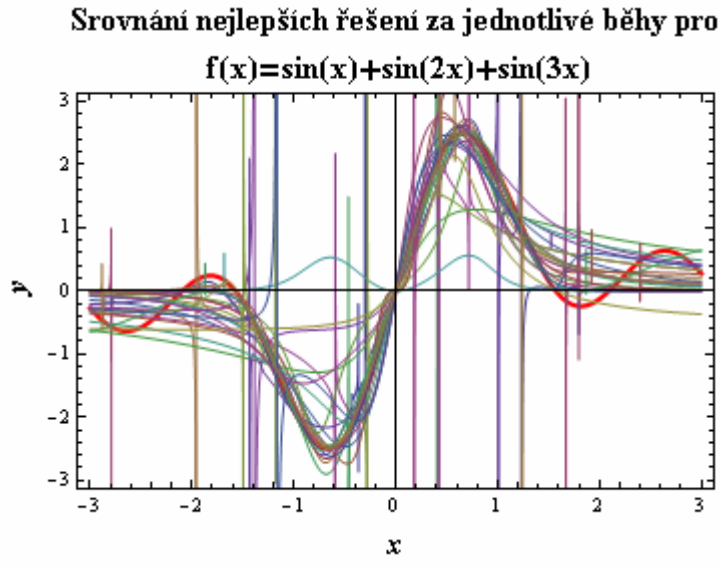
Parametry GP byly nastaveny následovně:

Parametr	Hodnota
PopSize	700
Generations	500
Runs	50
Functions	{Plus[], Subtract[], Times[], Divide[], Minus[]}
Terminals	{x, Random[-4,4]}
Method	HalfAndHalf
Depth	6
Leafcountcm	250
FrMutace	0.1
FrKrizeni	0.9
FrReprodukce	0

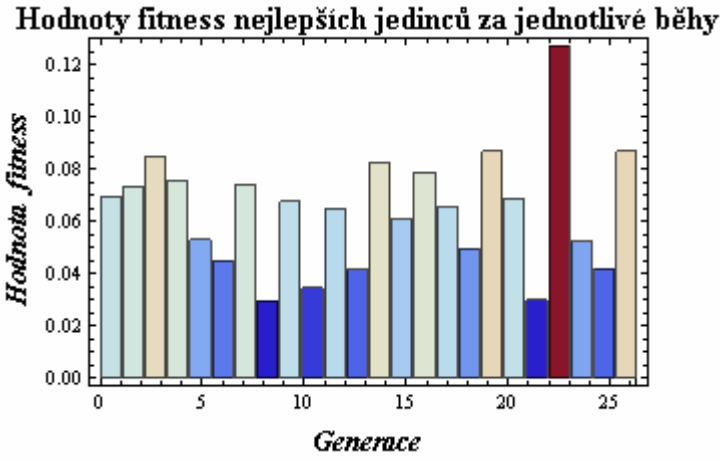
Tab.4.:Nastavení parametrů pro 3.testovací příklad

5.1.3.2 Vyhodnocení výsledků

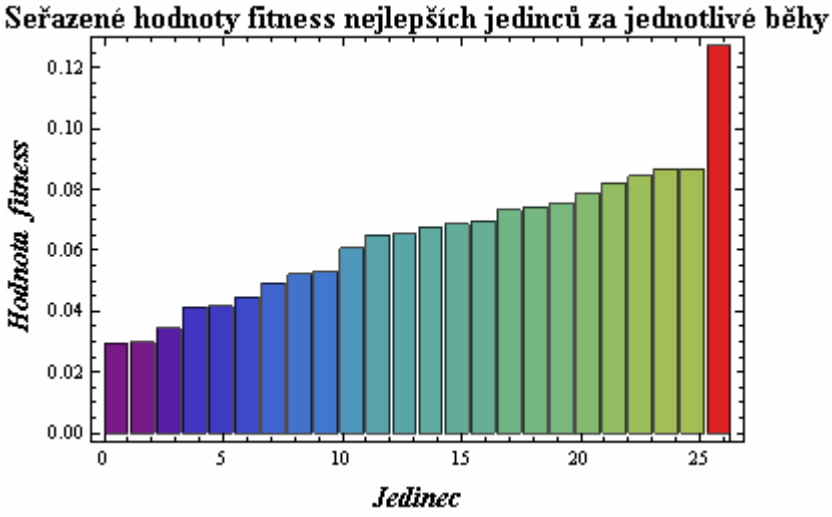
Z technických důvodů bylo u tohoto testovacího příkladu provedeno jen 24 místo původních 50 zamýšlených běhů



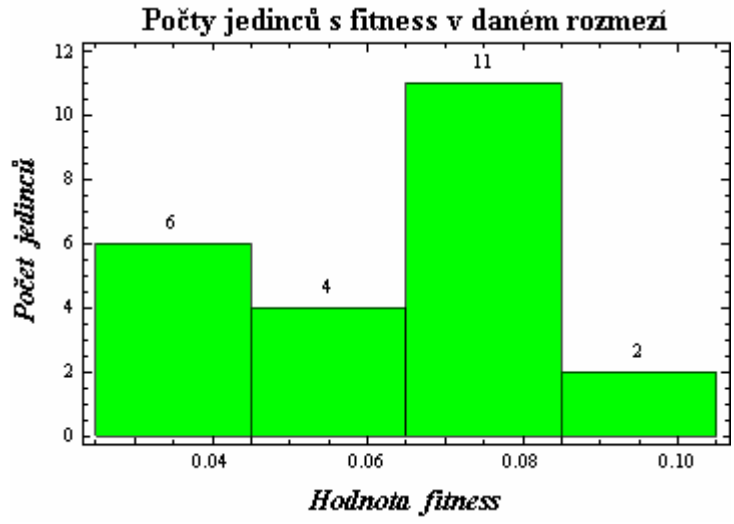
Obr. 41.:Nejlepší řešení za jednotlivé běhy, 3.testovací příklad



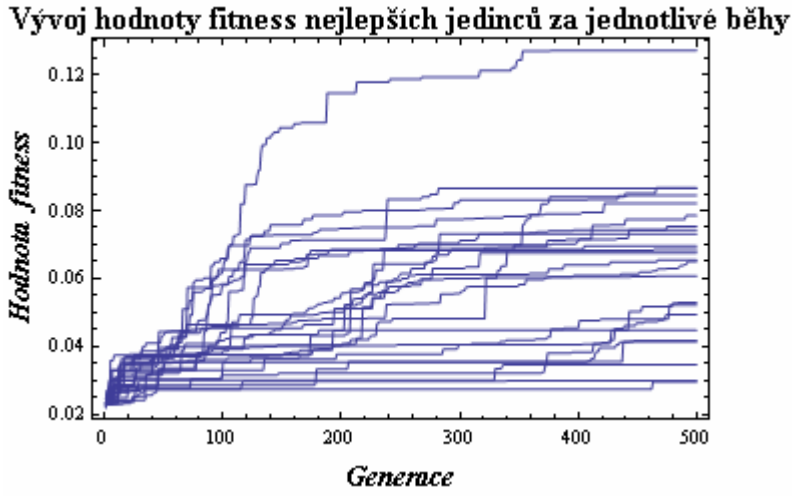
Obr. 42.:Nejlepší jedinci a fitness, 3.testovací příklad



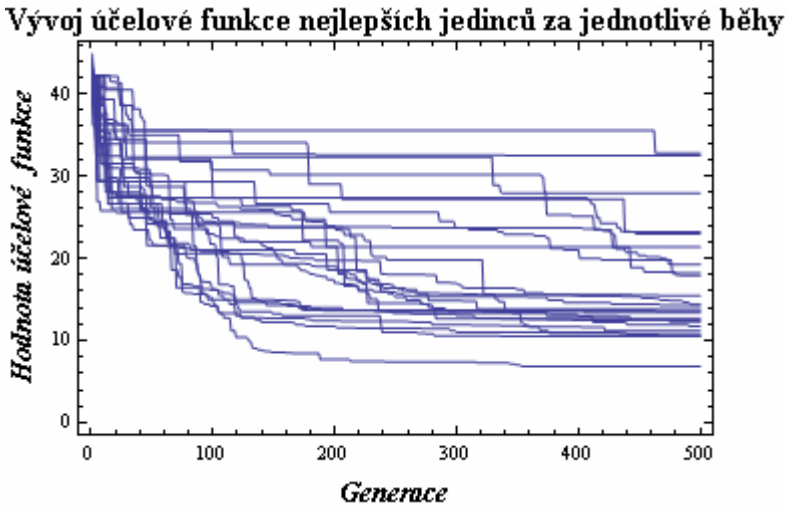
Obr. 43.:Seřazení nejlepších jedinců, 3.testovací příklad



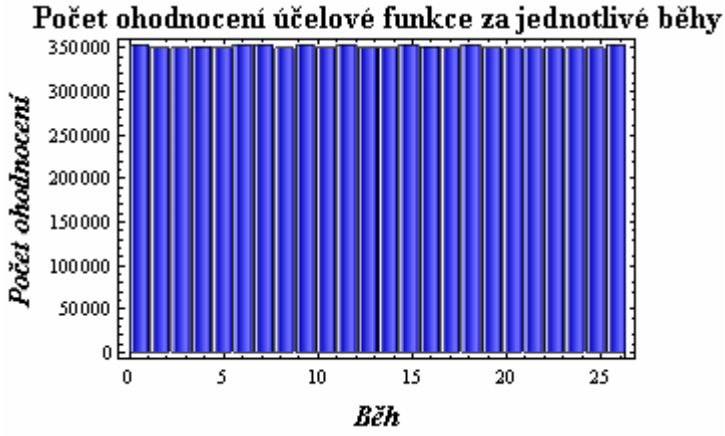
Obr. 44.:Porovnání hodnoty fitness a počtu jedinců, 3.testovací příklad



Obr. 45.:Vývoj fitness nejlepších jedinců, 3.testovací příklad



Obr. 46.:Vývoj hodnoty účelové funkce nejlepších jedinců, 3.testovací příklad

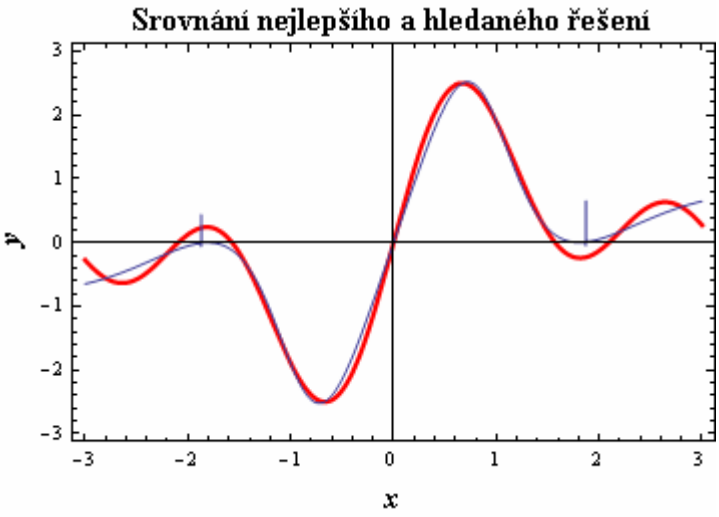


Obr. 47.:Počet ohodnocení účelové funkce, 3.testovací příklad

Z výsledků je patrné, že při snaze najít odpovídající aproximaci funkce $f_3(x)$ pomocí polynomů je konvergence k hledanému optimálnímu řešení velmi pomalá. Nejlepší nalezené řešení mělo tvar

$$f_3^1(x) = -\frac{x\left(-3{,}4183 + x^2\right)\cdot\left(-3{,}29723 + x^2\right)^2}{7{,}88657 - 5{,}41623x^2 + 5{,}62549x^4 - 1{,}42794x^6 + 0{,}0571361x^8 - 0{,}00996203x^{10}}$$ a

hodnotu fitness $F_A^1 = 0{,}127261263625244$. Graf porovnání nejlepšího jedince a hledaného řešení je na obrázku (Obr.48).

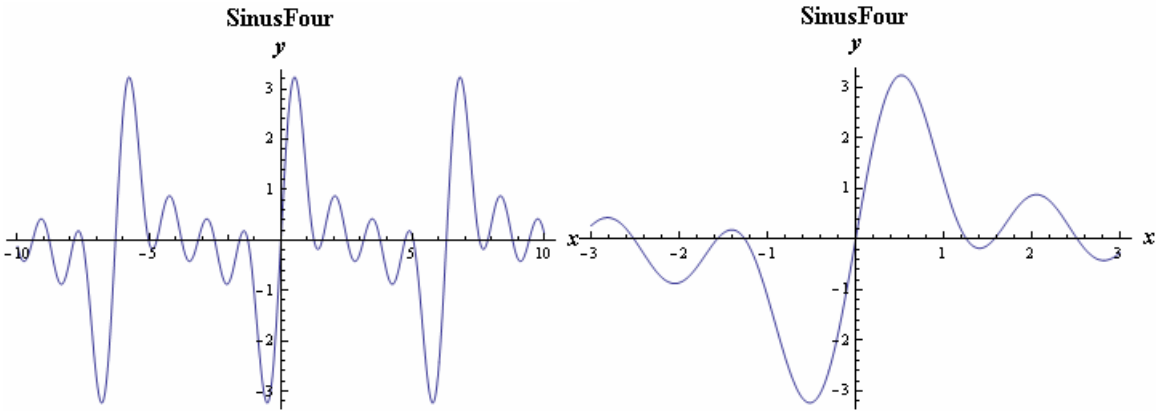


Obr. 48.:Porovnání hledaného a nejlepšího řešení, 3.testovací příklad

Celkový počet ohodnocení účelové funkce za 24 běhů byl 8440000. Konvergenci ke globálnímu optimu by bylo možné zrychlit zvětšením populace, zvětšením počtu evolučních cyklů a změnou omezení velikosti nově vznikajících potomků.

5.1.4 SinusFour

Posledním příkladem je funkce $f_4(x)$ ve tvaru $f_4(x) = \sin(x) + \sin(2x) + \sin(3x) + \sin(4x)$ na intervalu $x \in \langle -3;3 \rangle$. Graf funkce $f_4(x)$ je na obrázku (Obr.49).



Obr. 49.:Graf fce $f_4(x)$ a fce $f_4(x)$ na intervalu <-3,3>

5.1.4.1 Nastavení parametrů

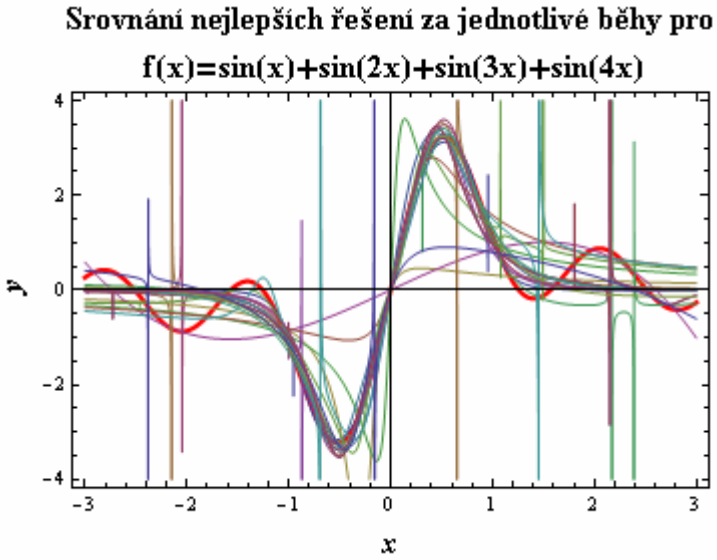
Parametry GP byly nastaveny následovně:

Parametr	Hodnota
PopSize	700
Generations	500
Runs	50
Functions	{Plus[], Subtract[], Times[], Divide[], Minus[]}
Terminals	{x, Random[-4,4]}
Method	HalfAndHalf
Depth	6
Leafcountcm	250
FrMutace	0.1
FrKrizeni	0.9
FrReprodukce	0

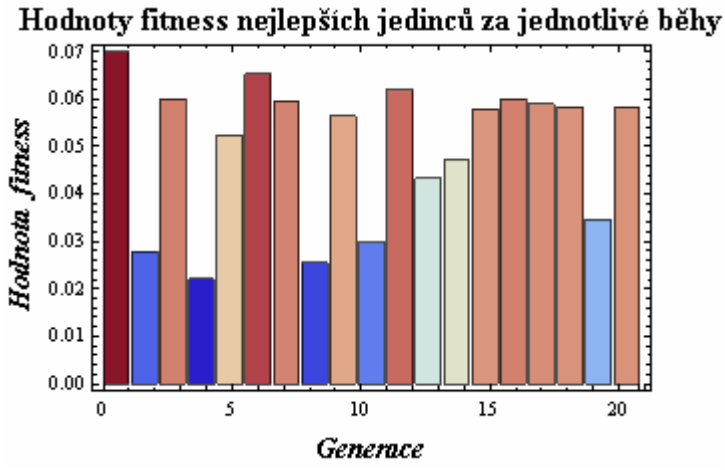
Tab.5.:Nastavení parametrů pro 4.testovací příklad

5.1.4.2 Vyhodnocení výsledků

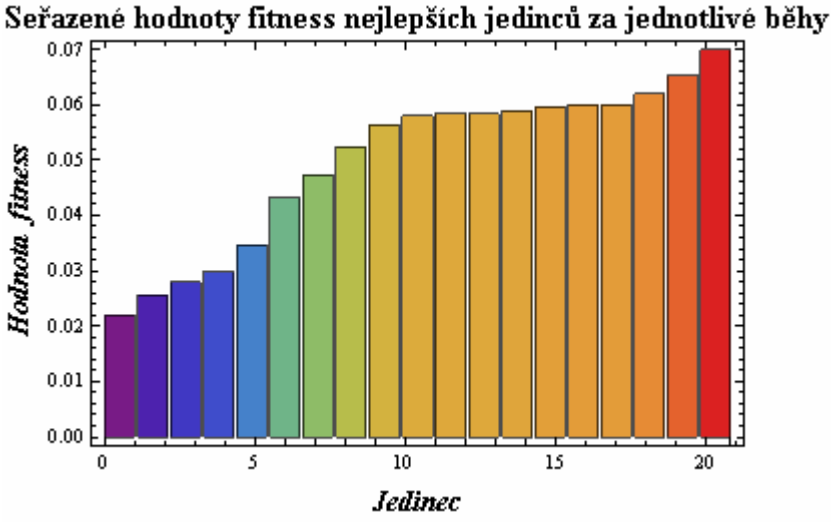
Z technických důvodů bylo u tohoto testovacího příkladu provedeno jen 19 z původně plánovaných 50 běhů.



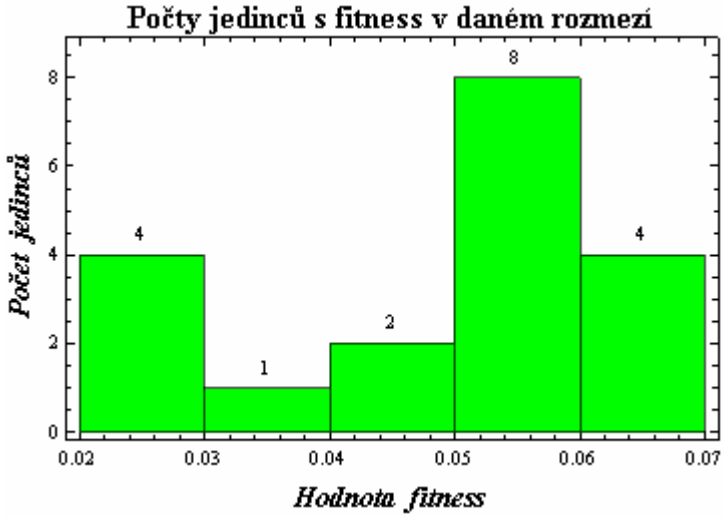
Obr. 50.:Nejlepší řešení za jednotlivé běhy, 4.testovací příklad



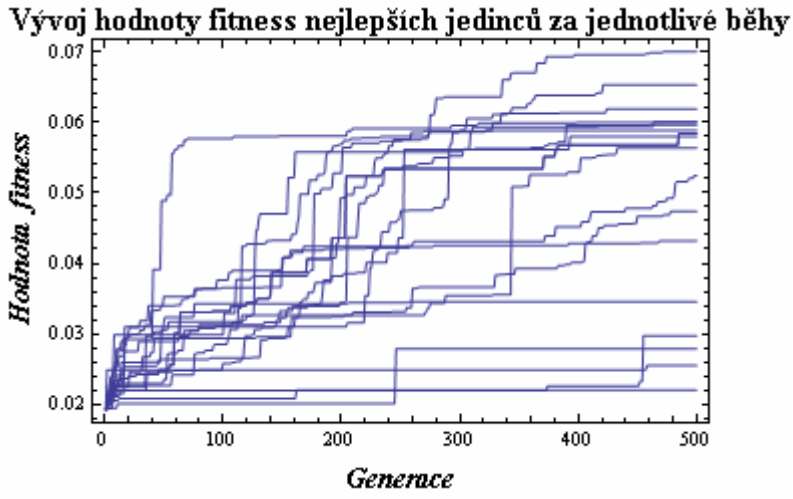
Obr. 51.:Nejlepší jedinci a fitness, 4.testovací příklad



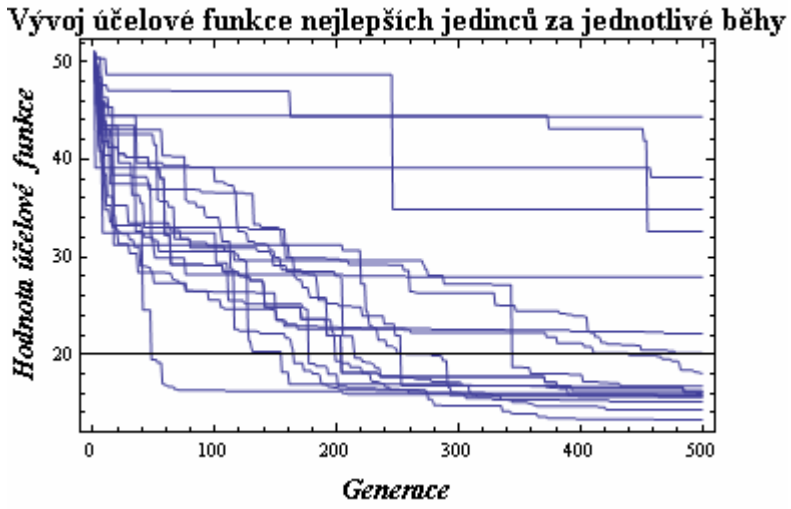
Obr. 52.:Seřazení nejlepších jedinců, 4.testovací příklad



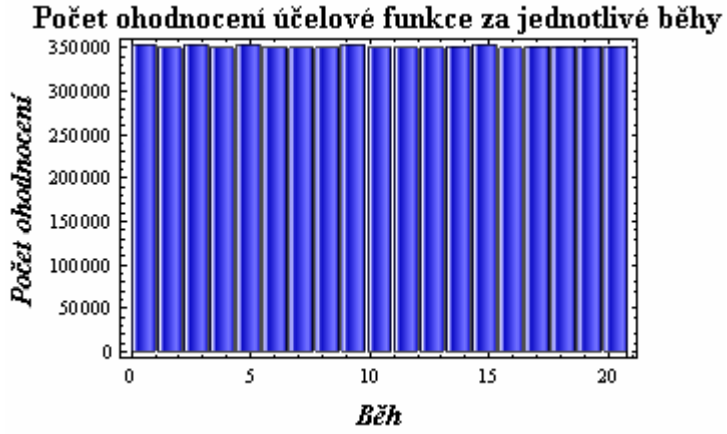
Obr. 53.:Porovnání hodnoty fitness a počtu jedinců, 4.testovací příklad



Obr. 54.:Vývoj fitness nejlepších jedinců, 4.testovací příklad



Obr. 55.:Vývoj hodnoty účelové funkce nejlepších jedinců, 4.testovací příklad

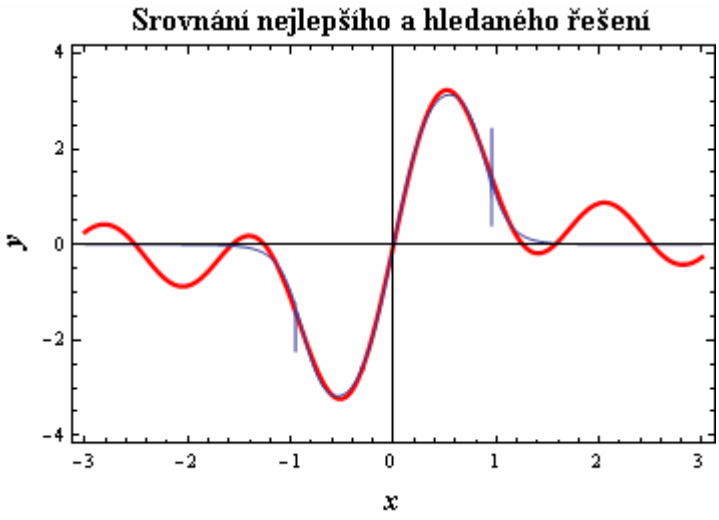


Obr. 56.:Počet ohodnocení účelové funkce, 4.testovací příklad

Stejně jako v předchozím testovacím příkladu, tak i zde je vidět velmi pomalá konvergence k hledanému optimálnímu řešení. Nejlepší nalezené řešení mělo tvar $f_4^1(x) =$

$$= -1.50838 + \frac{0.0408288\,x^2\,(1.64833 + x^2)}{(-0.953846 + x)\,(0.953586 + x)\,(1.02751 - 1.76044\,x + x^2)\,(0.408037 - 0.64697\,x + x^2)\,(0.850343 + 0.216499\,x + x^2)\,(0.36714 + 0.806902\,x + x^2)}$$

a hodnotu fitness $F_A^2 = 0{,}06996761407151633$. Graf porovnání nejlepšího jedince a hledaného řešení je na obrázku (Obr.57).



Obr. 57.:Porovnání hledaného a nejlepšího řešení, 4.testovací příklad

Celkový počet ohodnocení účelové funkce za 19 běhů byl 6681000. Konvergenci ke globálnímu optimu by bylo možné zrychlit zvětšením populace, zvětšením počtu evolučních cyklů a změnou omezení velikosti nově vznikajících potomků.

ZÁVĚR

Práce je ve své teoretické části zaměřena na popis genetického programování, objasnění základních pojmů a vysvětlení algoritmu genetického programování. Je zde popsán způsob reprezentace a tvorby jedinců, metody jejich ohodnocení a způsoby vytváření nových jedinců prostřednictvím genetických operátorů křížení, mutace a reprodukce.

Nicméně, hlavním cílem této práce bylo vytvořit algoritmus genetického programování v prostředí Mathematica a následně jej otestovat na všeobecně uznávaných testovacích příkladech.

Úspěšně si mi podařilo vytvořit funkce realizující generování a inicializaci jedinců podle zvolené metody (úplná metoda, růstová metoda a lineární půl na půl metoda), funkce jejich ohodnocení, funkce výběru rodičů (turnajová selekce, resp. výběr podle vhodnosti). Dále funkce pro křížení (křížení podstromů, jednobodové a arita-2 křížení), mutace (mutace podstromů, se zachováním velikosti, bodová mutace, mutace konstanty, vyzvedávající, smršťující mutace a permutace) a reprodukci.

Algoritmus byl otestován na čtyřech testovacích příkladech z oblasti symbolické regrese. Cílem bylo najít k zadaným křivkám odpovídající funkce sestavené pouze ze zadaných funkcí a terminálů. Pro polynomy 5. a 6.stupně byly výsledky pozitivní. Při hledání trigonometrických řad se projevilo nevhodné nastavení řídících parametrů pomalou konvergencí k hledaným řešením. U polynomů se projevila jedna z výhod GP, a to netradiční tvar řešení v porovnání s tvarem zadaných funkcí. To může být výhoda zejména při řešení složitých problémů, kdy může dojít ke zjednodušení výsledků.

Na základě prováděných pokusů během tvorby a testování algoritmu GP mohu konstatovat, že velmi dobré výsledky v rámci testovacích příkladů použitých pro otestování funkčnosti algoritmu lze získat i volbou menší populace a většího počtu prováděných evolučních cyklů. Výhodou je kratší doba běhu v porovnání s volbou velké populace.

Díky univerzálnosti jednotlivých funkcí, lze algoritmus GP do budoucna s pomocí menších úprav použít pro řešení široké škály různorodých problému z oblasti evolučního návrhu algoritmů, regulátorů, regulačních a elektrických obvodů, plánování nebo řízení procesů. GP také nachází využití v oblastech, jako jsou paralelní výpočty, buněčné automaty nebo kvantové počítače, kde je pro inženýry a techniky těžké najít řešení nebo vytvořit odpovídající program.

CONCLUSION

In theoretical part this work is aimed at description of genetic programming, explanation main concepts and algorithm of genetic programming. There is discribing representation and creation of individuals, fitness methods and creation methods of new individuals via genetic operators like crossover, mutation and reproduction.

But the main aim of this works was to create the genetic programming algorithm in computation system Mathematica and tested its functionality on classical examples.

I successfully created the functions which implement generating and initialising of individuals by chosen method (full method, grow method and ramped half and half method), fitness functions, selection functions (tournament selection and fitness-proportionate selection). Next the crossover functions (subtree crossover, one-point crossover and arity-2 crossover), the mutation function (subtree, size-fair subtree, point, constant, hoist, shrink and permutation mutation) a reproduction function.

GP algorithm was tested on four classical examples from symbolic regression‘s area. The aim was to find for given curves the corresponding functions builded up only with given functions and terminals. For so called Quintic and Sextic problem the results were positive. By searching the trigonometric series was proved the unsuitable adjustment of control parameters by slow convergence to searched solutions. At polynomials it was proved one of GP’s advantages and this is the unusual solution-form in contrast given functions. It can be advantage during solving the difficult problems when it can lead to simplification of solutions.

Based on the realized experiments during the creation and testing GP algorithm I can claim that we can get very good results for the testing examples by chosing smaller population and bigger evolutionary loops. The advantage is shorted running-time in comparing with choice of big population.

Considering the universality of individual functions the algorithm GP can be simply modified to solving broad range of different problems like evolutionary design of algorithms, controllers, control and electrical circuits, planning and process control. GP is also used in areas like parallel computing, cellular automata or quantum computing, where is difficult to find or create the appropriate program.

SEZNAM POUŽITÉ LITERATURY

[1] O´REILLY, Una-May, et al. *Genetic Programming Theory and Practice II.* USA: Springer Science + Business Media, Inc., 2005. 337 s. ISBN 0-387-23253-2.

[2] ZELINKA, Ivan, OPLATKOVÁ Zuzana, ŠENKEŘÍK Roman. *Aplikace umělé inteligence: aneb vybrané statě z evolučních algoritmů.* 1.vyd. Zlín: Univerzita Tomáše Bati ve Zlíně, 2010. 151 s. ISBN 978-80-7318-898-6.

[3] KVASNIČKA, Vladimír, POSPÍCHAL, Jiří, TIŇO, Peter. *Evolučné algoritmy.* 1.vyd. Bratislava: Vydaveteľstvo STU, 2000. 223 s. ISBN 80-227-1377-5.

[4] MAŘÍK Vladimír, et al. *Umělá inteligence(4).* 1.vyd., Praha: Academia 2003, ISBN 80-200-1044-0, Kapitola 5, Genetické programování a vybrané problémy evolučních výpočtů, s. 128-170.

[5] HYNEK, Josef. *Genetické algoritmy a genetické programování.* 1.vyd. Praha: Grada Publishing, a.s., 2008. ISBN 978-80-247-2695-3, Kapitola 13, Genetické programování, s. 123-134.

[6] KOZA, John R. *Genetic Programming : On the Programming of Computers by Means of Natural Selection.* Sixth. [s.l.] : Massachusetts Institute of Technology, 1998. 813 s. ISBN 0-262-11170-5.

[7] POLI, Riccardo, LANGDON, William B., MCPHEE, Nicolas Freitag. *A Field Guide to Genetic Programming.* 1.vyd. [s.l.] : University of Essex, 2008. 250 s. ISBN 978-1-4092-0073-4.

[8] KOZA, John R. *Introduction to Genetic Programming: Tutorial.* Přednáška v rámci GECCO 2007 London July 7-11, 2007

[9] WEISE, Thomas. *Global Optimization Algorithms : Theory and Application* [online]. 2.vyd. [s.l.] : [s.n.], 2009, Version: 2009-06-26 [cit. 2010-05-14]. Dostupné z WWW : <http://www.it-weise.de/projects/book.pdf>.

[10] Wolfram Research, Inc. *Wolfram Reasearch: Mathematica, Technical and Scientific Software* [online]. c2010 [cit. 2010-05-22]. Dostupné z WWW: <http://www.wolfram.com>.

[11] SOULE, Terence. *Removal Bias: A new cause of code growth in tree based evolutionary programming.* In ICEC 98: IEEE International Conference on Evolutionary Computation 1998, IEEE Press, 1998, s. 781-786.

[12] MCPHEE, N. *A Schema Theory Analysis of the Evolution of Size in Genetic Programming with Linear Representations.*

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CGGP	Grammar-based Genetic Programming
CGP	Cartesian Genetic Programming
EA	Evoluční algoritmy
EVT	Evoluční výpočetní techniky
F	Množina funkcí
F_A	Upravená vhodnost
F_N	Normalizovaná vhodnost
F_R	Účelová funkce
F_S	Standardizovaná vhodnost
GA	Genetické algoritmy
GP	Genetické programování
p_C	Frakce křížení
p_M	Frakce mutace
p_R	Frakce reprodukce
PADO	Paralel Algorithm Discovery and Orchestration
T	Množina terminálů

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Příloha P I: CD se zdrojovým kódem algoritmu GP (obsah CD je v souboru *obsah.txt*)