VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

GEOMETRICKÉ SÉMANTICKÉ GENETICKÉ PROGRAMOVÁNÍ

GEOMETRIC SEMANTIC GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ONDŘEJ KONČAL**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/16990/2017/xkonca00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů　　　　　　　　　　　　　　　　　　　　Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Končal Ondřej, Bc.**

Obor: Bioinformatika a biocomputing

Téma: **Geometrické sémantické genetické programování**
Geometric Semantic Genetic Programming

Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s problematikou genetického programování a jeho variantami, zejména se zaměřte na geometrické sémantické genetické programování (GSGP) a kartézské genetické programování (CGP).
2. Seznamte se s existujícími implementacemi GSGP a CGP.
3. Navrhněte metodu pro automatickou redukci kandidátních stromů v GSGP.
4. Navrženou metodu implementujte ve zvoleném programovacím prostředí.
5. Experimentálně vyhodnoťte navrženou metodu srovnáním s původní verzí GSGP na zadaných úlohách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

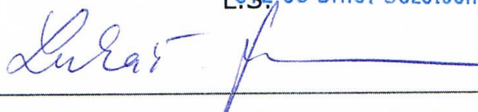
Vedoucí: **Sekanina Lukáš, prof. Ing., Ph.D.**, UPSY FIT VUT

Datum zadání: 1. listopadu 2017

Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.



prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato práce se zabývá převodem řešení získaného geometrickým sémantickým genetickým programováním (GSGP) na instanci kartézského genetického programování (CGP). GSGP se ukázalo jakožto kvalitní při tvorbě složitých matematických modelů, ale problémem je výsledná velikost řešení. CGP zase dokáže dobře redukovat velikost již vzniklých řešení. Tato práce dala pomocí kombinací těchto dvou metod vzniknout podstromovému CGP (SCGP), které jako vstup používá výstup GSGP a evoluci pak provádí pomocí CGP. Experimenty provedené na čtyřech úlohách z oblasti farmakokinetiky ukázaly, že SCGP dokáže vždy zmenšit řešení a ve třech ze čtyř případů navíc úspěšně bez přetrénování.

Abstract

This thesis examines a conversion of a solution produced by geometric semantic genetic programming (GSGP) to an instantion of cartesian genetic programming (CGP). GSGP has proven its quality to create complex mathematical models; however, the size of these models can get problematically large. CGP, on the other hand, is able to reduce the size of given models. This thesis combinated these methods to create a subtree CGP (SCGP). The SCGP uses an output of GSGP as an input and the evolution is performed using the CGP. Experiments performed on four pharmacokinetic tasks have shown that the SCGP is able to reduce the solution size in every case. Overfitting was detected in one out of four test problems.

Klíčová slova

geometrické sémantické genetické programování, kartézské genetické programování, symbolická regrese, evoluční algoritmus, farmakokinetika

Keywords

geometric semantic genetic programming, cartesian genetic programming, symbolic regression, evolutionary algorithm, pharmacokinetics

Citace

KONČAL, Ondřej. *Geometrické sémantické genetické programování.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Geometrické sémantické genetické programování

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Ondřej Končal
21. května 2018

Poděkování

Tímto bych rád poděkoval svému vedoucímu prof. Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení, vstřícnost při konzultacích a cenné rady, které mi pomohly tuto práci vyhotovit. Tato práce byla podporována Ministerstvem školství, mládeže a tělovýchovy z projektu „IT4Innovations National Supercomputing Center – LM2015070“ pro podporu velkých výzkumných infrastruktur, experimentální vývoj a inovativní projekty.

Obsah

1 Úvod **3**

2 Evoluční algoritmy **4**
2.1 Základní pojmy . . . 4
2.2 Princip evolučního algoritmu . . . 5
2.3 Genetické programování . . . 6
2.3.1 Formy GP . . . 9
2.3.2 Symbolická regrese . . . 10
2.4 Pokročilé genetické operátory v genetickém programování . . . 10
2.4.1 Syntaktické . . . 10
2.4.2 Kontextové . . . 11
2.4.3 Sémantické . . . 11
2.5 Geometrické sémantické genetické programování . . . 13
2.5.1 Open source knihovna . . . 15
2.5.2 Výhody a nevýhody GSGP . . . 16
2.6 Kartézské genetické programování . . . 16
2.6.1 Reprezentace . . . 16
2.6.2 Mutace . . . 17
2.6.3 Prohledávací algoritmus . . . 17
2.6.4 Výhody a nevýhody CGP . . . 17

3 Návrh **19**
3.1 Rozbor GSGP . . . 19
3.2 Genotyp a funkce CGP . . . 19
3.3 Převod výstupu GSGP na instanci CGP . . . 20
3.4 Ohodnocení jedince v CGP . . . 21
3.5 Mutace, selekce a ukončení . . . 21
3.6 Podstromové CGP . . . 21

4 Implementace **27**
4.1 Funkce vsazené do programu GSGP . . . 27
4.1.1 Načtení parametrů a případné spuštění vlastní evoluce a evaluace . . 27
4.1.2 Převod výsledku jedince GSGP na instanci CGP . . . 27
4.2 Třída CGP . . . 28
4.3 Vstup a výstup . . . 28
4.3.1 Argumenty . . . 28
4.3.2 Parametry . . . 28
4.3.3 Trénovací a testovací data . . . 30
4.3.4 Výstup . . . 30

4.4 Překlad a spuštění 31

5 Experimentální vyhodnocení metody **32**
5.1 Trénovací a testovací data 32
5.2 Nastavení parametrů 33
5.3 Experimenty 33
5.3.1 Výstup geometrického sémantického programování 34
5.3.2 Změna počtu generací v SCGP 35
5.3.3 Změna metody selekce podstromů v SCGP 39
5.3.4 Nejlepší nalezené nastavení SCGP 41
5.3.5 Optimalizace výstupu SCGP pomocí CGP 43
5.3.6 Shrnutí 45

6 Závěr **46**

Literatura **47**

Kapitola 1

Úvod

Tvoření a inovování jsou činnosti především lidské. S nástupem automatizace a počítačů se ale čím dál více práce přenáší na stroje. Ty jsou dnes schopné nejen tvorby předem dané, ale umí i automaticky tvořit a inovovat dle požadovaných vlastností. K automatizované tvorbě programů a matematických modelů slouží genetické programování s mnohými modifikacemi. Jedná se o algoritmus ze skupiny přírodou inspirovaných algoritmů, algoritmů evolučních. Tyto algoritmy původně sloužily, a většina dnes stále slouží, k optimalizaci.

Základní forma genetického programování tvoří a mění jedince pouze na úrovni syntaktické. Malá syntaktická změna však může vyvolat velkou změnu na úrovni sémantiky a obráceně. A jelikož je sémantika to, co je u vytvářených systémů hlavní, bylo by výhodné ji měnit cíleně. Přímé změny na sémantické úrovni dokáže provádět geometrické sémantické genetické programování (GSGP), které však vytváří příliš velké matematické modely. Zde může pomoci kartézské genetické programování (CGP), které se ukázalo jakožto vhodné pro optimalizaci již existujících řešení z oblasti návrhu obvodů a symbolické regrese. Tato práce se snaží spojit tyto dvě vylepšení genetického programování a vytvořit tak co nejkvalitnější řešení, které bude co nejmenší.

Tato práce je strukturována následovně: Kapitola 2 se zabývá teoretickými předpoklady genetického programování, pokusy o jeho sémantické rozšíření a v neposlední řadě GSGP a CGP. Kapitola 3 rozebírá návrh převodu řešení získaného GSGP na instanci CGP a jeho následné použití v metodě podstromového CGP, které využívá CGP k evoluci řešení. Návrh počítá s použitím knihovny pro GSGP představené v 2.5.1. Následující kapitola 4 pak popisuje implementaci tohoto návrhu a v kapitole 5 jsou s touto implementací prováděny experimenty. Kapitola 6 shrnuje výsledky a poznatky obsažené v této práci a obsahuje návrh budoucí práce.

Kapitola 2

Evoluční algoritmy

Informatika, tak jako mnohé vědy, čerpá inspiraci v biologii. Evoluční algoritmy čerpají inspiraci konkrétně z Darwinovy teorie evoluce a teorií z ní vzniklých. Jádrem teorie evoluce a také evolučního algoritmu je fakt, že organismy se v průběhů generací přizpůsobují podmínkám prostředí, ve kterém žijí. Jedinec s vhodnějšími vlastnostmi pro dané prostředí má větší šanci na přežití a následné zplození potomků, kteří budou nést jeho geny.

Algoritmy s myšlenkou postupného vývoje optimálního modelu v průběhu generací vznikaly nezávisle na sobě už od 60. let 20. století. Koncem tisíciletí byly sjednoceny pod pojmem evoluční algoritmy (EA). Nejprve byly využívány pouze k optimalizaci parametrů, ale v poslední době dokáží navrhnout celé modely dle zadaných dat a/nebo parametrů. Společnou vlastností všech EA je existence multimnožiny kandidátních řešení, nad kterou mohou být prováděny operace upravující její prvky a zajišťující vyhodnocování těchto prvků [9].

2.1 Základní pojmy

V oblasti EA existuje několik ustálených a často používaných pojmů. Definice zde uvedených pojmů jsou vytvořeny podle [9]. Mnoho pojmů v oblasti EA je analogií ke stejnojmenným pojmům z biologie a mezi hlavní patří následující:

Gen – Základní jednotka informace jak v molekulární genetice, tak v EA. Obvykle je to jeden bit nebo celé číslo, ale může jím být i číslo desetinné.

Alela – Hodnota genu.

Chromozom – Uspořádané pole genů.

Genotyp – Soubor všech chromozomů, které kódují kandidátní řešení. Pokud je kandidátní řešení tvořeno pouze jedním chromozomem, jsou pojmy chromozom a genotyp totožné.

Fenotyp – Vyjádření genotypu. Vzniká přímou interpretací genotypu nebo algoritmem využívající hodnoty genotypu.

Fitness – Hodnota určující kvalitu kandidátního řešení. Obvykle větší hodnota fitness značí kvalitnější kandidátní řešení a tím pádem i větší šanci na předání genů potomkům. Existuje

více druhů vyjádření hodnoty fitness, které jsou používány v závislosti na problému, který EA řeší:

Hrubá fitness – hodnota z množiny hodnot přirozených řešenému problému.

Standardizovaná hodnota fitness – převedená hodnota hrubé fitness do podoby, kde hodnota 0 je nejlepší možné ohodnocení jedince.

Přizpůsobená hodnota fitness – výpočet je proveden pomocí vzorce

$$\frac{1}{\text{standardizovaná hodnota fitness} + 1}\,. \tag{2.1}$$

Výsledné hodnoty se pak nachází v intervalu $\langle 0,\, 1 \rangle$, kde hodnota 1 je ohodnocením nejlepším.

Normalizovaná hodnota fitness – vypočítá se jako

$$\frac{\text{hrubá hodnota fitness}}{\sum_{i=1}^{N} \text{hrubá hodnota fitness i-tého jedince}}\,, \tag{2.2}$$

kde N je velikost populace. Hodnoty následně leží v intervalu $\langle 0,\, 1 \rangle$, lepší jedinec má vyšší ohodnocení a suma hodnot normalizované fitness je 1.

Populace – Multimnožina kandidátních řešení. Většinou má pevně danou velikost, ale někdy se může velikost v průběhu výpočtu měnit.

Generace – Jedna iterace EA.

2.2 Princip evolučního algoritmu

EA jsou metody stochastického prohledávání, které mají společné následující části (které se však u jednotlivých algoritmů mohou významně lišit):

Inicializace – Vytvoření počáteční populace (generace 0). Většinou je vytvořena z náhodně vygenerovaných jedinců, ale při znalostech o problému je možné s jejich pomocí vytvořit populaci cíleně.

Ohodnocovací funkce – Funkce přiřazující hodnotu fitness na základě fenotypu.

Ukončovací kritérium – Podmínka, při jejímž splnění běh algoritmu skončí. Většinou je to nalezení dostatečně kvalitního řešení a/nebo dosažení určitého počtu generací.

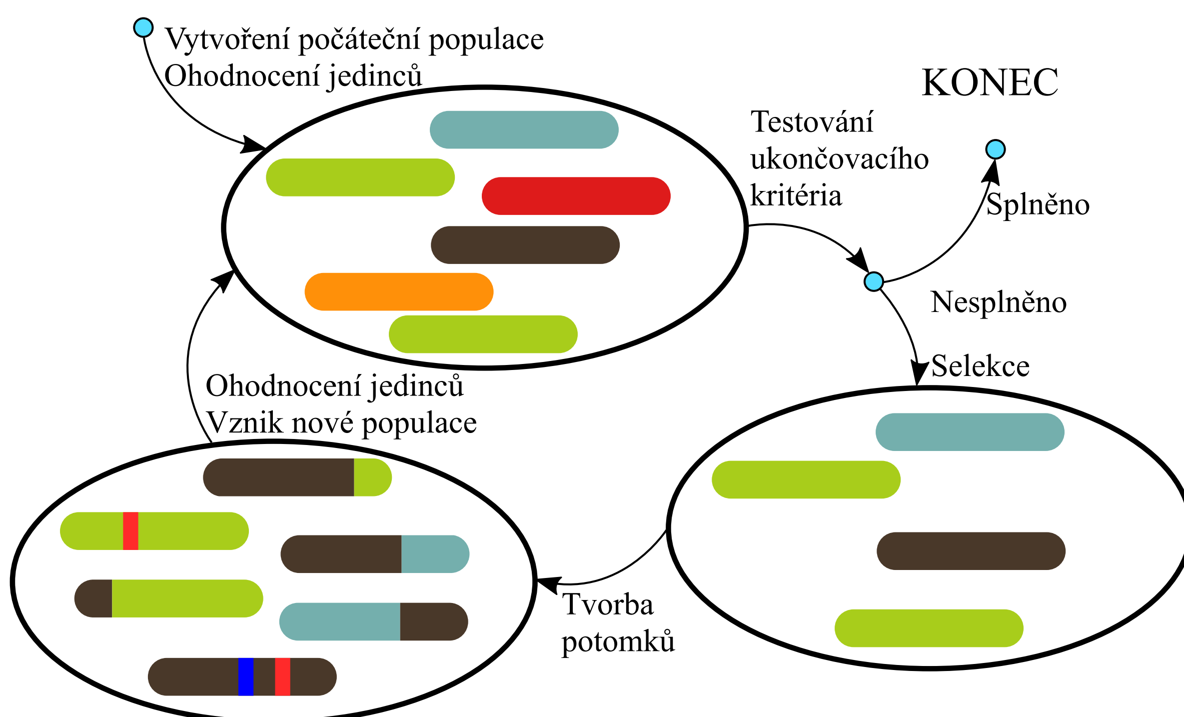
Selekce – Výběr jedinců, kteří se stanou rodiči pro další generaci. Na ně pak mohou být aplikovány genetické operátory nebo přechází do další generace beze změny.

Reprodukce – Z vybraných jedinců (rodičů) vznikají za pomoci genetických operátorů potomci. Genetický operátor má vždy pouze určitou šanci na aplikaci. Mezi dva hlavní operátory patří křížení a mutace. Křížením vznikne jedinec stvořený kombinací rodičů a

mutace mění hodnoty náhodně vybraných genů rodiče na jiné.

Vytvoření nové generace – Nová generace obsahuje určitý poměr nově vytvořených jedinců a jedinců již existujících v aktuální generaci. Pokud je vytvořena pouze z potomků, jedná se o model *generační* (jako u jednoletých rostlin). Druhá možnost je vytvoření populace jak z potomků, tak z jedinců z aktuální generace. Potom se jedná o tzv. *překrývající se* model (jako u zvířat).

Samotný běh EA (obr. 2.1) začíná inicializací následovanou ohodnocením vzniklé populace. Následně je populace testována na ukončovací kritérium. Dokud nejsou jeho podmínky splněny, probíhá následující cyklus: Selekce rodičů, jejich reprodukce, ohodnocení potomků a nakonec vytvoření nové generace [9].



Obrázek 2.1: Průběh evolučního algoritmu

2.3 Genetické programování

Genetické programování (GP) je druh EA, který umožňuje automatizovaně vytvářet počítačové programy. Kandidátní programy jsou obvykle reprezentovány syntaktickými stromy proměnné délky. GP vzniklo koncem 80. let 20. století a k jeho rozvoji nejvíce přispěl John Koza. Hlavní rozdíl oproti předchozím druhům EA je v tom, že GP může optimalizovat nejen parametry, ale dokáže vytvářet celé spustitelné struktury (programy).

Strom je sestaven z prvků množiny terminálů T a množiny neterminálů N. Uzly stromu představují funkce f_i s nenulovým počtem argumentů, kde $f_i \in N$, listy zase konstanty, proměnné a funkce bez argumentů náležící do množiny T. Funkce by měly být *uzavřené* –

každý výstup funkce lze použít jako vstup jakékoli jiné funkce. Funkce, které mohou mít nedefinovanou hodnotu (dělení nulou nebo odmocnina a logaritmus ze záporných čísel) by měly používat tzv. *chráněnou variantu*, tj. vždy vracet nějakou hodnotu [9].

Počáteční populace je obvykle generována náhodně z množiny terminálů T a množiny neterminálů N. Čím větší diverzita počáteční populace, tím lépe. Jako základní způsoby generování jedinců byly vytvořeny následující metody [5]:

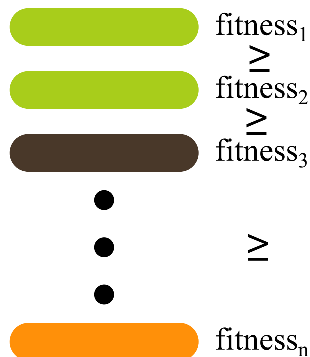
> *Grow* – Uzly jsou náhodně vybírány jak z množiny T, tak z množiny N tak dlouho, dokud neobsahují všechny koncové uzly terminály, nebo není dosaženo maximální hloubky stromu. Při maximální hloubce stromu jsou uzly vybírány jen z množiny T. Populace obsahuje náhodně velké stromy od hloubky 1 až po maximální možnou.
>
> *Full* – Dokud není dosaženo maximální hloubky stromu, jsou uzly vybírány pouze z množiny N. Při jejím dosažení jsou vybírány naopak uzly z množiny T. To zajistí maximální možný nárůst stromu, ale celá populace je tvořena stejně hlubokými stromy.
>
> *Ramped half-and-half* – Nejprve je určen rozsah hloubek stromů, mezi které je následně rovnoměrně rozdělena populace. Polovina stromů z každé skupiny hloubek je pak vytvořena metodou Grow, druhá polovina metodou Full. Tato metoda vytvoří populaci s největší syntaktickou diverzitou.

Existuje několik selekčních mechanismů, kdy každý vede na jiný selekční tlak a tím pádem i jinou rychlost konvergence k extrému účelové funkce. Pokud je selekční tlak příliš vysoký, může být konvergence předčasná a dosažený extrém bude pouze lokální. Nejběžnější algoritmy selekce jsou následující:

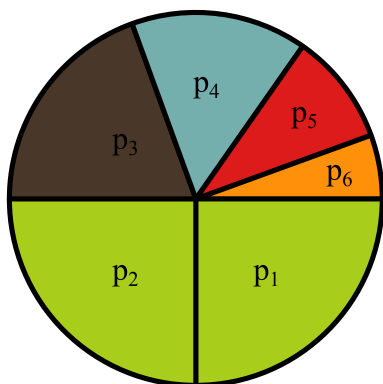
> *Deterministická selekce* (obr. 2.2) – Deterministicky jsou vybrání jedinci s co největší hodnotou fitness.
>
> *Proporcionální selekce (Ruleta)* (obr. 2.3) – Jedinec p_i má pravděpodobnost na výběr určenou jako
>
> $$p_i = \frac{f_i}{\sum_{j=1}^{G} f_j}, \tag{2.3}$$
>
> kde G je počet jedinců v populaci. Tento výpočet zachová poměr mezi hodnotami fitness, ale převede je na hodnoty z intervalu $\langle 0, 1 \rangle$ a jejich suma je rovna 1. Při seřazení pravděpodobností za sebe je pokryt celý interval $\langle 0, 1 \rangle$ a vygenerováním náhodného čísla z tohoto intervalu je vybrán právě jeden jedinec (jako při zatočení kolem štěstí/ruletou). Vygenerování čísla proběhne tolikrát, kolik jedinců je potřeba vybrat. Každý jedinec má jistou nenulovou šanci být vybrán a může být vybrán i vícekrát.
>
> *Podle pořadí (Ranked)* (obr. 2.4) – Jedinci se seřadí sestupně podle hodnoty fitness a pravděpodobnost výběru je nepřímo úměrná pořadí jedince. Vyřeší se tak potenciální problém rulety, kdy někteří jedinci mohou mít řádově lepší fitness hodnotu a jedinci s horším ohodnocením se pak prakticky nikdy nemohou

dostat do další generace.

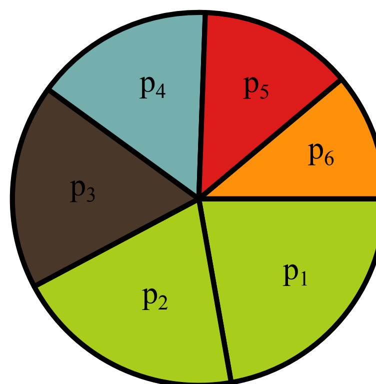
Turnaj (obr. 2.5) – Náhodně je vybráno k jedinců ($k \geq 2$) z populace, kteří se porovnají, a ten s nejvyšší hodnotou je vybrán pro reprodukci. Turnaj proběhne tolikrát, kolik jedinců je zapotřebí vybrat. Vždy bude vybrán nejlepší ze skupiny k jedinců, ale tato skupina se může skládat i z jedinců s nejhorším ohodnocením.



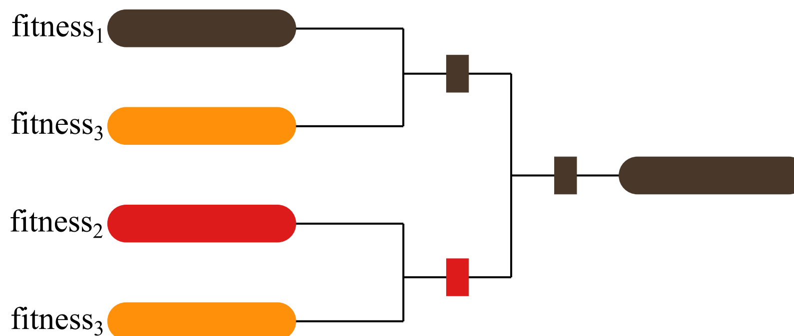
Obrázek 2.2: Deterministická selekce



Obrázek 2.3: Proporcionální selekce (Ruleta)



Obrázek 2.4: Podle pořadí (Ranked)



Obrázek 2.5: Turnaj

Dále je možno zohlednit volbu elitismu. Pokud je elitismus v algoritmu povolen, zajišťuje, že se jedinec s nejlepším ohodnocením v aktuální generaci vždy dostane do generace násle-

dující.

Ve stromově orientovaném GP je křížení realizováno jako výměna podstromů mezi dvěma stromy. V každém z rodičů je náhodně vybrán jeden podstrom, jenž si s druhým rodičem vymění. Tímto se oba přímo stanou potomky, takže pokud je potřeba zachovat rodiče, musí se jedinec před křížením zduplikovat.

Typická implementace mutace náhodně vybere uzel ze stromu vybraného jedince. Následně se vygeneruje nový strom, který se vloží na dané místo v rodičovském stromu. Opět je zde potřeba prvně zkopírovat strom, pokud se má rodič zachovat.

Se změnou stromu se ale nemusí vždy změnit hodnota fitness, pokud dojde například ke vzniku tzv. *intronu* (např. $n + 0$ nebo $\frac{n}{1}$). Strom tak může aplikací těchto, pro hodnotu fitness neutrálních, operací začít narůstat a prodlužovat dobu ohodnocení jedince. Tomuto jevu se říká *bloat*. Lze jej regulovat, pokud je do hodnoty započítaná výška stromu nebo je celkově výška stromu omezena.

Výpočet fitness je většinou prováděn na základě zadané trénovací množiny. Hodnota fitness pak může představovat sumu všech odchylek od trénovací množiny (viz vzorec 2.4), průměrnou odchylku (viz vzorec 2.5) nebo počet hodnot, které se liší více než o povolenou odchylku ϵ od požadované hodnoty (viz vzorec 2.6) [9]. V následujících vztazích značí y_j^{GP} výstupy kandidátního programu, které jsou porovnávané s výstupy trénovací množiny y_j o velikosti N.

$$f = \sum_{j=1}^{N} (y_j^{GP} - y_j)^2 \tag{2.4}$$

$$f = \frac{\sum_{j=1}^{N} |y_j^{GP} - y_j|}{N} \tag{2.5}$$

$$f = \frac{\sum_{j=1}^{N} h_j}{N}; \quad \text{kde} \quad h_j = \begin{cases} 1 & |y_j^{GP} - y_j| < \epsilon \\ 0 & \text{jinak} \end{cases} \tag{2.6}$$

2.3.1 Formy GP

Pro základní formu GP, jak už bylo řečeno, je typické, že kandidátní řešení jsou reprezentována syntaktickými stromy. Tím se liší od klasických EA, jakými jsou například genetický algoritmus nebo evoluční strategie, které pracují s vektory hodnot z příslušné problémové domény přímo. Algoritmy odvozené nebo inspirované GP tvoří taktéž spustitelné struktury a řadí se mezi ně kartézské GP, lineární GP nebo gramatická evoluce [7].

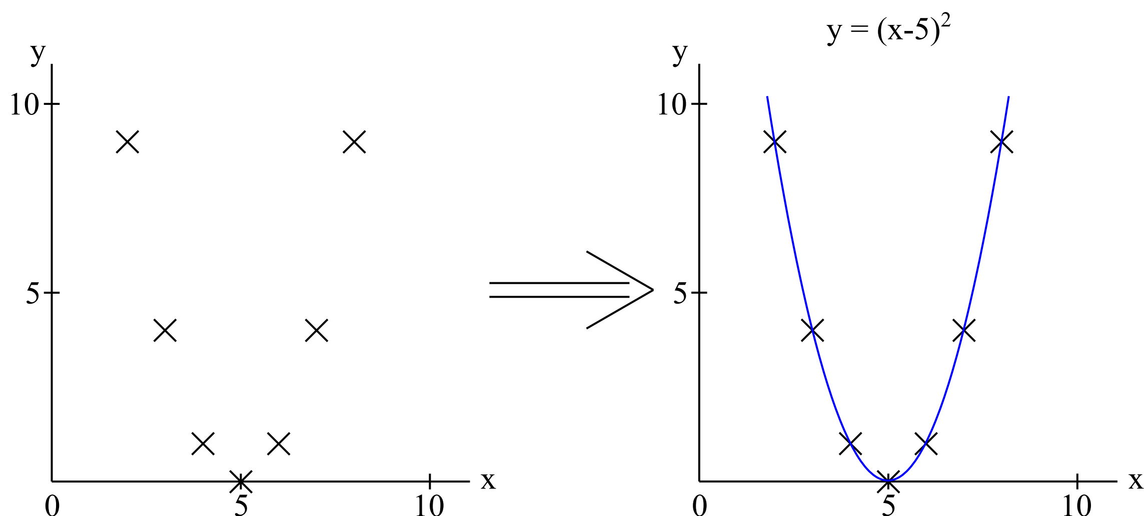
Kartézské GP používá pro reprezentaci orientované acyklické grafy a bude podrobně rozebráno v podkapitole 2.6.1.

V Lineárním GP jsou kandidátní řešení sice reprezentována za pomoci binárního řetězce, ale ten je chápán spíše jako strojový nebo tříadresný kód. Skupiny bitů určené velikosti tak kódují operace, proměnné a konstanty.

Gramatická evoluce pracuje s jedinci tvořenými vektorem celých čísel a gramatikou v Backus-Naurově formě (BNF), pomocí které jsou vektory překládány na kandidátní program. Vychází se ze startovního symbolu gramatiky, který je postupně, spolu s ostatními neterminály, přepsán dle pravidla vybraného hodnotou aktuálního genu *modulo* počet přepisovacích pravidel neterminálu. Pokud jsou všechny geny vyčerpány, začíná se opět od začátku chromozomu a řetězec je upravován tak dlouho, dokud neobsahuje pouze terminály.

2.3.2 Symbolická regrese

Klasickou úlohou na otestování funkčnosti GP je symbolická regrese. Jedná se o hledání matematického modelu, který co nejlépe popisuje zadaná data. Výstupem je funkce $E : \mathbb{R}^n \to \mathbb{R}$ popisující vstupní data tvořená dvojicemi (v_i, y), kde $v_i \in \mathbb{R}^n$ a $y \in \mathbb{R}$, jejíž přesnost závisí na velikosti a rozmanitosti trénovacích dat a na funkcích, ze kterých může být vytvořena [5]. Příklad takto zadaných dat a k nim nalezené odpovídající funkce je uveden na obr. 2.6.



Obrázek 2.6: Symbolická regrese

Konvenční řešení předpokládají výsledný tvar matematické formule, případně funkce, ze kterých je tato formule složena. Následně se hledají pouze koeficienty, se kterými výsledná funkce co nejpřesněji popisuje data.

U GP je fitness funkce volena podle aplikace. V případě tolerance občasné, výrazně velké, odchylky od trénovacích dat, je výhodné výpočet hodnoty fitness provádět za pomoci vzorce 2.6. Výpočet tímto způsobem změní hodnotu fitness i při značném rozdílu hodnot v jednom bodu (funkce zde nemusí být definovaná, nebo se zde vyskytuje lokální extrém) pouze o 1. V opačném případě, kdy velké výchylky nemohou být tolerovány, je lépe výpočet hodnoty fitness provádět dle vzorce 2.4 nebo 2.5.

2.4 Pokročilé genetické operátory v genetickém programování

Od vzniku GP bylo představeno mnoho vylepšení genetických operátorů, ale většina pracovala na úrovni genotypu. Není to ale genotyp, který je potřeba cíleně změnit, nýbrž fenotyp. V následující části podkapitoly je proto uveden přehled a základní rozdělení genetických operací, které se snaží nějakým způsobem upravovat fenotyp.

2.4.1 Syntaktické

Základní formy křížení i mutace se zaměřují pouze na syntaxi. Patří sem mimo jiné i jednobodové a uniformní křížení (nad zarovnanými stromy) a všechny formy mutace pro zmenšení stromu [8].

2.4.2 Kontextové

Kontext podstromů je využit jako dodatečná informace pro výběr uzlů ke křížení. Jako ohodnocení se používá hodnota fitness těchto podstromů. Bylo vytvořeno několik kontextových genetických operátorů [8]:

> *Soft Brood Selection (SBS)* – Rodiče se kříží N-krát a vytvoří $2N$ potomků, ze kterých se vyberou dva nejlepší. Tento postup je inspirován živočichy, kteří se častěji páří a porodí více potomků, než kolik jich přežije.
>
> *Selective Crossover* – Každý možný podstrom je ohodnocen dle toho, jak velký má dopad na celý strom. Křížení je pak provedeno výměnou nejlépe ohodnoceného podstromu prvního rodiče za nejhůře ohodnocený podstrom druhého rodiče.
>
> *Context Aware Crossover (CAC)* – Z prvního rodiče se náhodně vybere podstrom, který je křížen se všemi možnými podstromy druhého rodiče. Ze všech vzniklých potomků je vybrán nejlepší a proces je zopakován i pro druhého rodiče.
>
> *Context Aware Mutation (CAM)* – Vygeneruje se multimnožina potencionálně užitečných podstromů, ze kterých je jeden náhodně vybrán. Ten je vložen na všechna možná místa rodiče a ze vzniklých potomků je vybrán ten s nejlepší hodnotou fitness.

2.4.3 Sémantické

Sémantika je topologický prostor o n dimenzích, kde n je velikost trénovací množiny. V klasickém GP je zanedbána a všechny operátory pracují na úrovni syntaxe. Pro účely GP ji lze popsat jako vektor výstupů programu pro každý vstup. Sémantika je tak úzce svázána s hodnotou fitness. Jedná se o poměrně novou disciplínu, která má dva významné mezníky. Těmi je vznik nepřímých, sémantiku zohledňujících, metod kolem roku 2009 a vznik přímých, sémantiku si uvědomujících, metod roku 2012. Před nimi existovaly pouze pokusy o zvýšení syntaktické diverzity, která sémantiku ovlivňovala bez zaručených výsledků [10].

Diverzita

Diverzita je důležitou vlastností v GP. Pokud je moc malá, začne se GP ubírat do extrému účelové funkce, a pokud se tak stane moc brzy, je tento extrém pouze lokální a může se z něj dostat pouze za pomoci dobré mutace. O zvýšení diverzity při inicializaci se snažil už Koza s metodou Ramped Half-and-Half, ale ta zajišťovala pouze syntaktickou diverzitu. Úplnou diverzitu při inicializaci zajišťuje až *Semantically Driven Initialization* (SDI), která vytvoří jedince s takovým genotypem, že všechny listy a uzly jsou nezbytné pro výpočet hodnoty fitness.

Diverzitu za běhu programu lze kontrolovat rozložením hodnot fitness nebo různými metrikami pracujícími jak na úrovni genotypu, tak na úrovni fenotypu [10].

Nepřímé sémantické metody

Nepřímo lze sémantiku upravovat tak, že se upravuje syntaxe a podle dopadu na sémantiku jsou vybíráni jedinci. Metody jsou děleny na metody sémantické diverzity, lokality a geometrie povrchu fitness [10]:

Sémantická diverzita – Vychází z myšlenky diverzity populace, ale zde je aplikována na jedince a jeho potomky. Pro problémy nad boolovskou doménou lze použít *Reduced Ordered Binary Decision Diagrams* (ROBDDs). Ty redukují stromy do minimálních tvarů, které si odpovídají sémanticky právě tehdy, když jsou stromy identické. Toto lze využít po křížení a provádět ho tak dlouho, dokud nejsou potomci sémanticky různí od rodičů [10].

Obecnou metodou je *No Same Mate* (NSM) selekce, kdy je zakázáno křížení jedinců se stejnou hodnotou fitness. Dalšími genetickými operátory podporujícími sémantickou diverzitou jsou *Semantic Aware Crossover* (SAC) a *Semantic Aware Mutation* (SAM). Oba operátory kontrolují sémantickou ekvivalenci vybraných podstromů (u mutace vybraného a vygenerovaného podstromu) na náhodné množině hodnot z domény problému. Pokud nejsou ekvivalentní, operace je provedena. V opačném případě jsou vybrány nové podstromy (u mutace je vybráno nové místo a vygenerován nový podstrom) [8].

Lokalita – Další z vlastností důležitých pro všechny EA je lokalita a EA se snaží mít lokalitu co nejmenší. Definice malé lokality říká, že malá změna v genotypu vede na malou změnu fenotypu. Převedeno na případ syntaxe a sémantiky to znamená, že malá syntaktická změna vyvolá malou sémantickou změnu. Nad boolovskou doménou byl opět použit ROBDDs a obecnou metodou se stalo *Approximating Geometric Crossover* (AGC), které vytvoří několik potomků z rodičů a vybere nejpodobnějšího, ale sémanticky odlišného, potomka (podobné jako SBS). Nebylo ale lepší něž klasické křížení.

Vylepšení přineslo až *Semantic Similarity based Crossover* (SSC). To má určitý předem daný počet pokusů, po který vybírá náhodné podstromy z rodičů a porovná je na základě sémantické vzdálenosti. Pokud je dost malá, ale nenulová, je křížení provedeno. Pokud není, procedura se opakuje až do vyčerpání povoleného počtu pokusů. V případě nenalezení žádných sémanticky podobných podstromů jsou prohozeny podstromy náhodné. Dalším vylepšením bylo *The Most SSC* (MSSC), kde je vždy proveden předem daný počet náhodných výběrů podstromů a nakonec je realizováno křížení v uzlech se sémanticky si nejpodobnějšími podstromy.

Na stejném principu funguje i *Semantic Similatiry based Mutation* (SSM), která generuje nový podstrom a vybírá náhodný uzel v rodiči tak dlouho, dokud není splněna podmínka podobnosti nebo není dosažen předem daný počet pokusů [8].

Geometrie povrchu fitness – Po vynalezení geometrického křížení, které se snaží vytvořit jedince co nejblíže lineární kombinaci obou rodičů a leží tak mezi nimi [2], byla tato metoda aplikována i na sémantiku. Vzniklo tak *Approximating Geometric Crossover* (AGC), které vytvoří z rodičů několik potomků a vybere toho sémanticky nejbližšího svým rodičům. Jeho úspěšným vylepšením byla změna křížení, které nově potomka umístí do stejné sémantické vzdá-

lenosti od obou rodičů. Tento operátor při testech překonal všechny ostatní. Tato forma křížení začíná vykazovat přímou změnu sémantiky, neboť potomka umístí na určené místo – někam mezi rodiče [10].

Přímé sémantické metody

Do této kategorie spadají metody, které pracují přímo na úrovni sémantiky. Jako první sem lze zařadit vylepšení dříve popsané AGC s křížením umisťujícím potomka do stejné sémantické vzdálenosti od rodičů. Dále byly představeny *Geometric Semantic Crossover* (GSC), jehož výsledek nebude nikdy horší než je horší z rodičů, a *Geometric Semantic Mutation* (GSM), která vždy vytváří unimodální geometrii povrchu fitness pro úlohy přiřazení vstupních dat ke známým výstupům (např. regrese a klasifikace) [10]. Geometrické sémantické genetické operátory jsou podrobněji rozebrány v následující podkapitole 2.5.

2.5 Geometrické sémantické genetické programování

Geometrické sémantické genetické programování (GSGP) se snaží upravovat jedince přímo na úrovní sémantiky místo syntaxe za pomocí geometrických sémantických operátorů (GSO). Představené GSO vypadají následujícím způsobem:

$$\text{GSC: } T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2) \tag{2.7}$$

$$\text{GSM: } T_M = T + ms \cdot (T_{R1} - T_{R2}) \tag{2.8}$$

kde $T, T_1, T_2 : \mathbb{R}^n \to \mathbb{R}$ jsou rodiče, $T_R, T_{R1}, T_{R2} : \mathbb{R}^n \to \mathbb{R}$ jsou náhodné reálné funkce s oborem hodnot $\langle 0,\, 1\rangle$ a ms je reálná hodnota označovaná jako *mutation step*, která udává velikost mutace. Pro upravení náhodně vygenerovaných funkcí tak, aby vracely hodnoty v rozsahu $\langle 0,\, 1\rangle$, je použita speciální logistická funkce sigmoida:

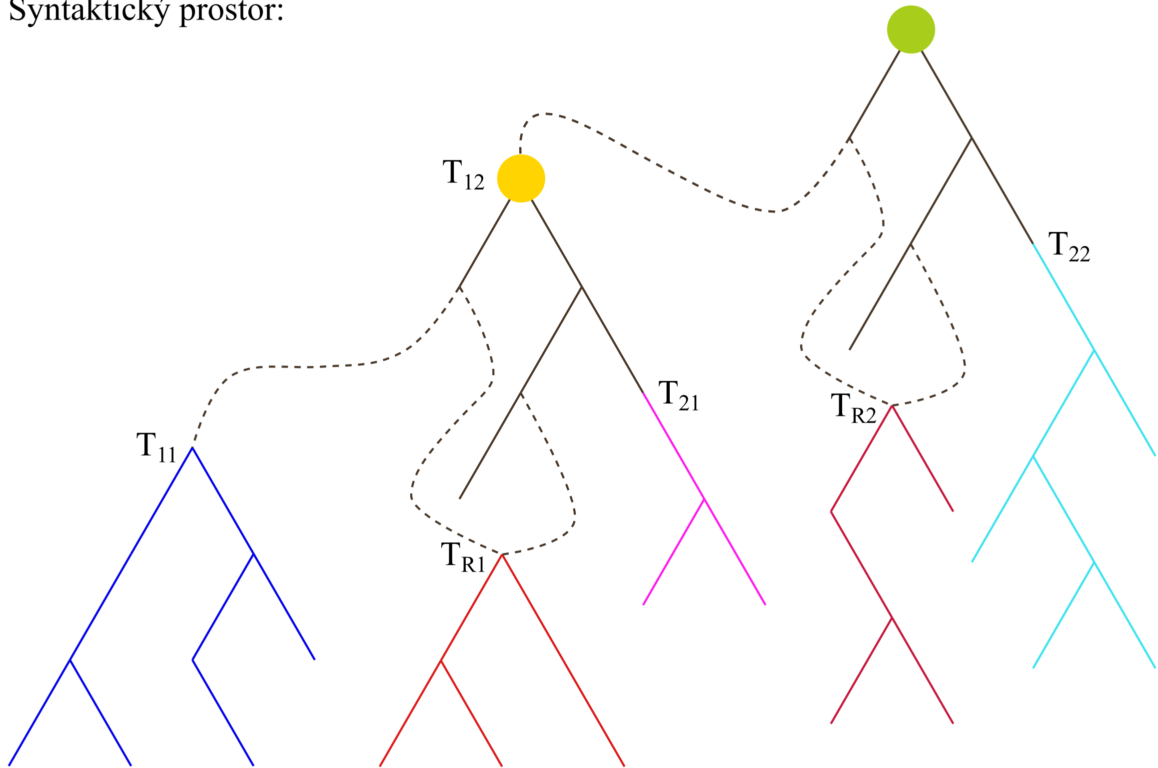
$$T_R = (1 + e^{-T_{rand}})^{-1} \tag{2.9}$$

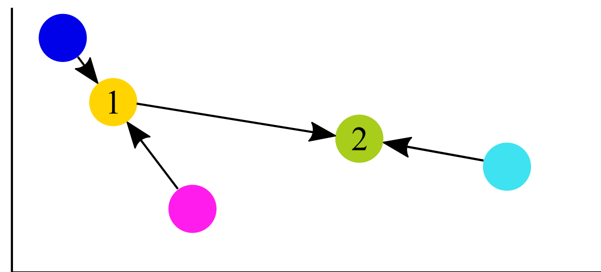
kde T_{rand} je funkce vracející neomezenou hodnotu.

Potomek vzniklý těmito operacemi je ale vždy větší než rodič, a to dokonce větší než součet velikostí všech stromů, které vstupují do operace. Tento nárůst lze vidět na obrázcích 2.7 a 2.8, kdy jsou připojovány stále nové stromy k původnímu stromu vstupujícímu do operace. Na obrázku 2.7 s GSC je vidět spojení nejprve rodičů (modrý a purpurový strom) a náhodného stromu dávající vzniknout novému jedinci (žlutý), který je v následující generaci křížen s dalším jedincem (azurový strom) s pomocí jiného náhodného stromu a vzniká zelený jedinec. Jedinec vzniklý GSC spadá v sémantickém prostoru mezi své rodiče a vzdálenost od nich je ovlivněna přidaným náhodným stromem (viz sémantický prostor). Obrázek 2.8 s GSM ukazuje připojení náhodně vytvořených stromů (červený a oranžový strom) v jedné generaci k rodiči (modrý strom) a vzniká tak nový jedinec (žlutý), který je v následující generaci mutován pomocí přidání dalších dvou náhodných stromů a vzniká zelený jedinec. GSM posune jedince v jeho sémantickém okolí o hodnotu úměrnou kroku mutace. Aplikací těchto operací naroste velikost jedince velmi rychle v rozsahu několika málo generací a to způsobí i nárůst doby potřebné pro výpočet hodnoty fitness [2]. Byly navrženy dvě možnosti, jak se s touto skutečností vypořádat. První je provedení redukce jedinců na sémanticky ekvivalentní po každém vytvoření nové generace. Druhou, která je použita v této práci, je vytvoření počáteční populace spolu s multimnožinou náhodně vygenerovaných stromů a noví jedinci jsou tvořeni pouze odkazy [10]. Záznam jedince po křížení

pak vypadá ($crossover, \&T_1, \&T_2, \&T_R$) a po mutaci ($mutation, \&T, \&T_{R1}, \&T_{R2}, ms$).

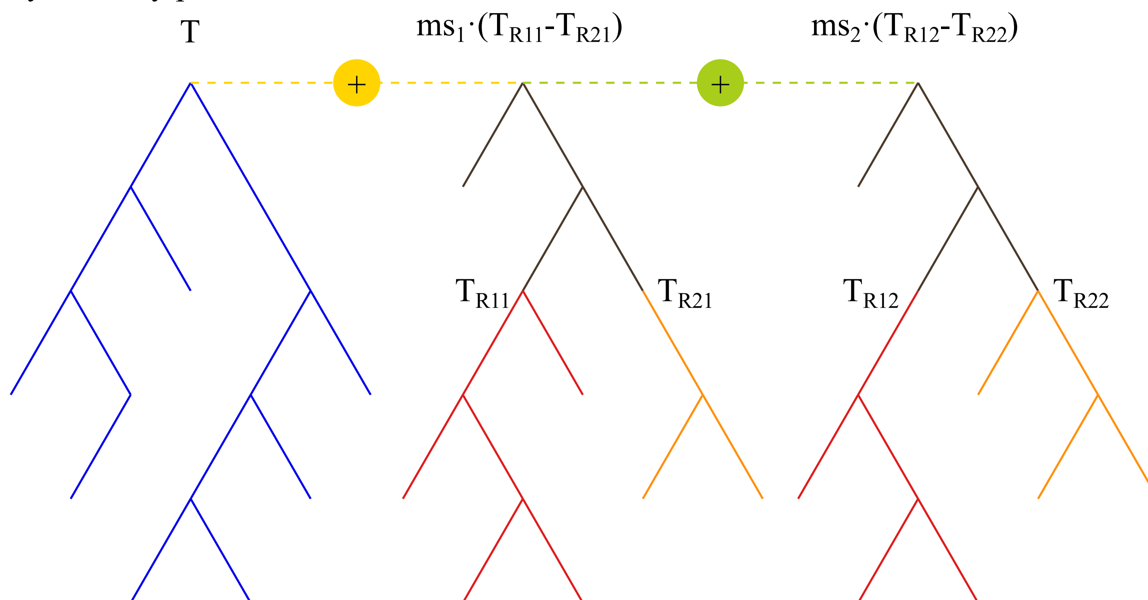
Díky čistě aritmetickému přepočtu nad již ohodnocenými stromy není potřeba hodnotu

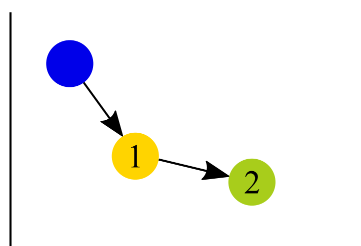




Obrázek 2.7: Aplikace GSC po dvě generace. V první generaci je vytvořen žlutý jedinec z modrého a purpurového stromu; v druhé generaci vzniká zelený jedinec ze žlutého a azurového stromu. Ve spodní části se nachází abstrakce sémantického prostoru, ve kterém potomek leží mezi svými rodiči.

fitness vždy počítat celou znova, ale pouze ji přepočítat dle vzorců GSO z hodnot fitness jednotlivých podstromů. To má za následek výrazné zrychlení ohodnocování jedinců. V případě zobrazení celkové struktury jedince je potřeba provést rekonstrukci, která je ale časově i prostorově náročná [2].





Obrázek 2.8: Aplikace GSM po dvě generace. V první generaci je vytvořen žlutý jedinec z modrého stromu; v druhé generaci vzniká zelený jedinec ze žlutého stromu. Ve spodní části se nachází abstrakce sémantického prostoru, ve kterém potomek leží v okolí rodiče.

2.5.1 Open source knihovna

Pro výzkumné účely GSGP byla vytvořena open source knihovna v jazyce C++. Je ke stažení na stránkách http://gsgp.sourceforge.net/ a jejím autorem je Mauro Castelli.

Při práci s knihovnou byly zjištěny dvě chyby v implementaci. První chyba způsobovala především vyřazení poslední funkce používané v uzlech vytvářených stromů a jednalo se o chráněné dělení. Chybou bylo zmenšení hodnoty, která násobila výsledek funkce *frand()*, o 1. Výsledkem tohoto odečtení a automatického zaokrouhlení dolů při převodu desetinného čísla na celé není nikdy vygenerována nejvyšší hodnota z požadovaného rozsahu. Odstranění této chyby bylo provedeno odstraněním -1 z řádků 1064, 1071, 1075, 1077 a 1404 v souboru *GP.h*. Změny byly provedeny 21.7.2017, reálné datum úprav ve zdrojových kódech bohužel chybí (je zjistitelné pouze z data poslední modifikace souborů).

Druhou nalezenou chybou je přebytečný cyklus *for(int k=0;k<30;k++)* na řádku 720 v souboru *GP.h*. Existence tohoto řádku má za následek pouze zpomalení běhu programu a funkčnost neovlivňuje. Ve zdrojovém kódu se tato chyba nachází stále.

2.5.2 Výhody a nevýhody GSGP

Největší výhodou GSGP je jeho schopnost používat genetické operátory na úrovni sémantiky a ovlivňovat kandidátní řešení přímo. Děje se tak za pomoci přidávání stromů, které s každou generací zvětšují jedince. U GSM se jedná o nárůst pouze o konstantu $c \approx 2 \cdot |T_R|$. Zvětšení jedince při GSC však závisí na počtu generací a $|T_N| \approx 2 \cdot |T_{N-1}| + |T_R|$. Největší nevýhodou je tedy stále narůstající velikost kandidátního řešení při aplikaci těchto operací. Pokusy o redukci kandidátních řešení byly neúspěšné a v praxi se používá dříve popsaná metoda uložení stromů a tvoření jedinců za pomoci odkazů na ně a na odkazy z minulých generací.

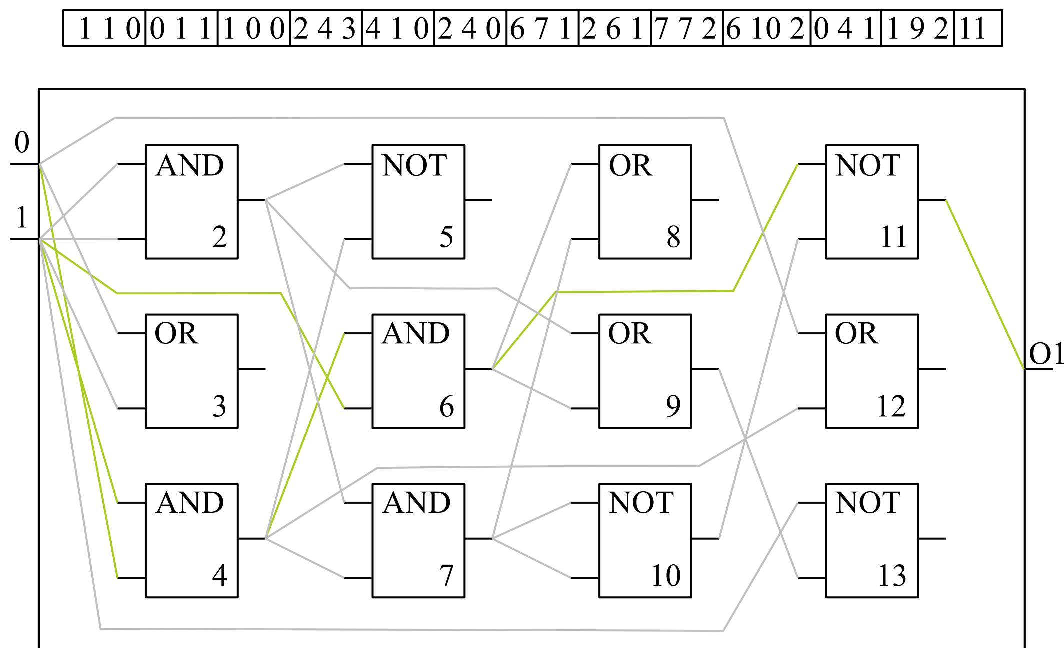
2.6 Kartézské genetické programování

Kartézské genetické programování (CGP) vzniklo jakožto metoda evoluce pro číslicové obvody. Vytvořil ji Julian F. Miller roku 1997, svůj název však dostala až později a mezi formy GP byla zařazena roku 2000 [6]. Jak ukázal Thomsonův experiment [9], evoluční design dokáže překonat návrh člověka díky pojmutí všech možných fyzikálních vlastností, které člověk zanedbává. Existují také polymorfní hradla, která mohou měnit svoji funkce v závislosti na fyzikálních vlastnostech (teplota, napětí) a činí ruční návrh takřka neproveditelným. Kromě návrhu číslicových obvodů se CGP uplatnilo při úlohách optimalizace obvodů, symbolické regrese, návrhu antén a filtrů, nebo při tvorbě umění [9].

2.6.1 Reprezentace

Kandidátní řešení jsou reprezentována v podobě orientovaných acyklických grafů, které mají formu dvojrozměrné mřížky uzlů. Kvůli tomu je nazýváno "kartézské". Chromozom je tvořen celými čísly, která značí zapojení vstupů, výstupů a funkce. Na jeho konci se nachází také geny, které označují zapojený výstupů (obr. 2.9). Po dekódování genotypu mohou některé uzly zůstat nepropojeny s výstupem a nepřispívat tak k výpočtu výstupní hodnoty. Takovým uzlům se říká *neaktivní* a lze je určit za pomoci rekurzivního sestupu počínajícího ve výstupech. Velikost fenotypu se tak v průběhu generací mění, ale genotyp má vždy konstantní velikost. Tímto je zabráněno tvorbě *bloatu*, který je jednou z nežádoucích vlastností GP. Místo něj zde vzniká redundance na několika úrovních, která je pro CGP přínosná. Těmito úrovněmi jsou: vynechání uzlů z fenotypu nebo jejich vícenásobné použití, neaktivita některého ze vstupů uzlu (v případě že jej funkce uzlu nepotřebuje), nepoužití primárního vstupu ve fenotypu [6].

Mřížka má předem danou velikost $n_c \times n_r$ (počet sloupců $\times$ počet řádků). Stejně je tomu i u počtu vstupů (n_i) a výstupů (n_o). Funkce uzlů jsou vybrány z množiny funkcí Γ a počet vstupů uzlů mřížky n_n je dán dle maximální arity funkcí z Γ. Celková velikost genotypu n_g je rovna $n_c \cdot n_r \cdot (n_n + 1) + n_o$. Vstupy jsou indexovány od 0 do $n_i - 1$ a uzly mřížky od n_i do $n_i + n_c \cdot n_r - 1$. Dalším parametrem je *L–back/levels–back* (L) určující, o kolik sloupců zpětně může být vstup uzlu napojen. Uzel není nikdy možné napojit do stejného sloupce. V případě, že se $L = n_c$, je možné připojit vstupy uzlu na libovolný uzel v předchozích sloupcích nebo na vstup. Pokud je k tomu ještě $n_r = 1$, je možné vstup připojit na libovolný předcházející uzel nebo vstup. Výstupy lze připojit na libovolný uzel obvodu nebo na vstup [6, 9].



Obrázek 2.9: Genotyp (nahoře) a fenotyp (dole) CGP

2.6.2 Mutace

Mutace je jediným genetickým operátorem v základní formě CGP. Náhodně vybraný gen změní svoji hodnotu na nově vygenerované číslo z povoleného rozsahu (dle parametru L u vstupů nebo dle $|\Gamma|$ u funkcí). Počet zmutovaných genů je udán jako celé číslo nebo jako procentuální část z celkové velikosti chromozomu. V CGP existují tzv. *neutrální mutace*, které změní genotyp, ale nezmění hodnotu fitness. Takové mutace mohou být: změna nezapojeného uzlu, změna nepoužitého vstupu u zapojeného uzlu, změny neovlivňující hodnotu fitness. Mutace, které vylepšují fitness, se nazývají *adaptivní*. Pokud nastane jedna adaptivní mutace po několika neadaptivních, má to obvykle za následek velkou změnu fenotypu. Zatímco jedna mutace genu má exploitativní funkci, tato řada mutací projevující se najednou má zase funkci explorativní (částečná náhrada za křížení) [9].

2.6.3 Prohledávací algoritmus

Mechanismus evoluce je podobný evoluční strategii $(1 + \lambda)$, kde populace obsahuje $(1 + \lambda)$ jedinců, ze kterých je vybrán 1 nejlepší jakožto rodič a z něj je vytvořeno genetickými operátory (zde pouze mutací) λ potomků (většinou $\lambda \in \langle 4, 9 \rangle$). V případě, že existuje více jedinců se stejně dobrou hodnotou fitness a tato fitness hodnota je nejlepší v populaci, je vybrán jedinec, který nebyl v minulé generaci rodičem. Zvýší se tak diverzita a dá se možnost projevu neutrálních mutací provedených na rodiči této generace. Zbylé kroky jsou stejné jako u GP (viz podkapitola 2.3) [9].

2.6.4 Výhody a nevýhody CGP

CGP si dokáže kromě návrhu obvodů a jiných spustitelných struktur také velmi dobře poradit s optimalizací již vzniklé struktury. Optimalizace může zmenšovat fenotyp nebo zrychlovat číslicové obvody hledáním co nejparalelnějšího zapojení. CGP také automaticky

zabraňuje vzniku *bloatu* a používá několik druhů redundance. V základní verzi používá jako genetický operátor pouze mutaci (díky neutrálním mutacím má však kromě exploitativní funkce i funkci explorativní) a to jen na syntaktické úrovni.

Kapitola 3

Návrh

Tato kapitola se zabývá návrhem programu, který zpracuje syntaktický strom vytvořený pomocí GSGP a zmenšuje jeho velikost pomocí CGP. Jedinec vytvořený GSGP je uložen jako několik stromů, mezi kterými jsou v jednotlivých generacích definována propojení. To vše je potřeba převést na uzly a jejich propojení v CGP. Takto vytvořený chromozom dosahuje značné velikosti a úlohou CGP je tak jeho redukce.

3.1 Rozbor GSGP

Knihovna GSGP (viz 2.5.1) používá při tvorbě matematického modelu pouze operace z množiny {+, -, *, /}, kde dělení nulou vrací 1. Generovat náhodné stromy a jedince počáteční populace lze metodami Grow, Full, Ramp half-and-half a selekce probíhá turnajem o dané velikosti. Záznamy o GSM, GSC a replikaci jsou uloženy do struktury s indexy tří stromů z minulé generace nebo z náhodně vygenerovaných stromů. U každého ze záznamů je také indikace, o kterou z těchto tří operací se jedná, a zda je použita ve výsledném řešení.

Ke knihovně je dodán také soubor s funkcí *main*, ve které je naimplementován běh GSGP za pomocí funkcí z knihovny. Jeho prvním důležitým výstupem je soubor obsahující funkce tvořené jedinci počáteční populace a také funkce náhodné, které byly použity při GSC a GSM. Druhým jsou záznamy GSC a GSM (případně replikace) v každé generaci, ale jen těch, které se podílely na vytvoření výsledku. Z těchto dvou datových záznamů lze vytvořit funkční výpočetní model a nebo předpis funkce. Dále jsou vytvořeny soubory obsahující data o běhu programu. Kromě klasické evoluce umožňuje program vzít model vytvořený evolucí a aplikovat jej na nová data.

Při převodu modelu vytvořeného GSGP na model zpracovatelný za pomoci CGP je potřeba myslet na výslednou velikost chromozomu CGP. Aplikace GSM má za následek konstantní nárůst velikosti stromu, GSC jej ale zvětšuje exponenciálně (viz 2.5.2). Z tohoto důvodu nebude GSC použito při tvorbě modelu popisujícího vstupní data.

Funkce *main* dodaná ke knihovně (soubor GP.cc) je ponechána v původním stavu, pouze je doplněna o tři volání funkcí. Jedná se o funkce umožňující nastavení CGP, spuštění CGP a převod výstupu GSGP na CGP.

3.2 Genotyp a funkce CGP

Počet vstupů n_i je dán počtem proměnných ve vstupních datech, $n_r = 1$, $n_o = 1$ a n_c má velikost dánu podle výsledku GSGP, případně zvětšenou o několik procent, aby se vy-

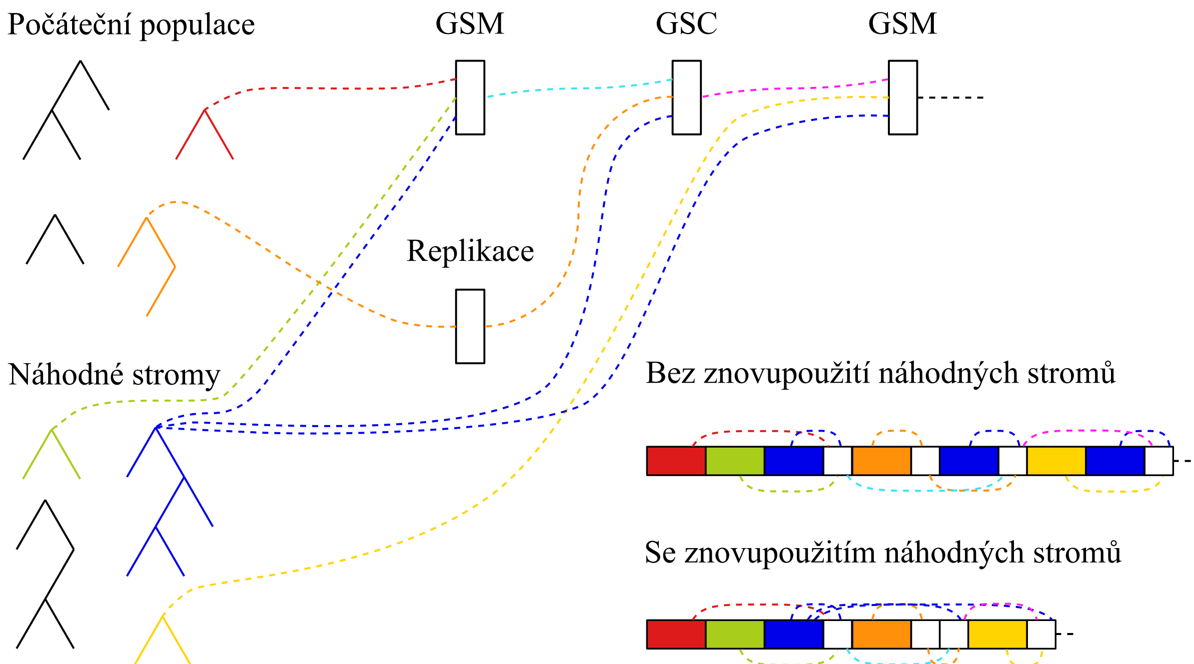
tvořily i neaktivní uzly. Každý uzel CGP obsahuje dva vstupy, svoji funkci, svoji hodnotu a informaci o tom, zda je uzel aktivní. Vstupy mohou být napojeny na jakýkoli předchozí uzel ($L = n_c$), hodnota uzlu je určena pro uchování hodnoty konstanty nebo pro uložení mezivýpočtu hodnoty fitness. Funkce uzlů jsou následující:

1. Konstanta (rozsah ⟨-10,00; 10,00⟩)
2. $+$
3. $-$
4. $*$
5. / (chráněné, $x/0 = 1$)
6. Sigmoida $(1 + e^{-x})^{-1}$
7. e^x

3.3 Převod výstupu GSGP na instanci CGP

Při převodu modelu GSGP na CGP je potřeba uložit do chromozomu jak použité stromy, tak operace nad nimi. Použití náhodných stromů se může začít opakovat a pak je potřeba je vložit do chromozomu znova, nebo se odkázat na již uložené stromy (obr. 3.1).

Převod nastává po uložení všech statistik. Prochází přes všechny generace a každý



Obrázek 3.1: Ilustrace převodu jedince v GSGP na jedince v CGP dvěma způsoby

záznam, který je označen jakožto použitý pro tvorbu výsledku, se přidá do chromozomu. Pokud se v záznamu objeví odkaz na náhodný strom nebo jedince z počáteční populace, je jeho první výskyt vždy převeden na postfixovou notaci a následně na odpovídající uzly CGP. V případě opakovaného výskytu stromu je možno vytvořit strom znovu, nebo se připojit na uzel CGP se stromem již vytvořeným. Při použití GSM, GSC a replikace je proveden odkaz vždy na uzly vytvořené v minulé generaci, případně na stromy/uzly náhodné, se kterými jsou spojeny (u GSM a GSC dodány další uzly propojující tři podstromy).

Jednodušší algoritmus (alg. 2) vždy znovu vytvoří náhodný strom – nemusí si pamatovat

a testovat, zda a který strom již použil. Složitější algoritmus (alg. 3) se odkazuje na uzly s vrcholy již uložených stromů v chromozomu CGP. Oba používají pro převod stromů do CGP algoritmus 1, který strom převede nejprve na vektor s posfixovou notací a následně jej zleva zjednodušuje a ukládá jako uzly CGP.

Algoritmus 1 Převod syntaktického stromu do chromozomu CGP

Input: Pointer to *tree*; //GSGP
Pointer to *chromosome* : vector of nodes; //CGP

Create postfix notation vector v from *tree*;

while count of items i in vector v is greater than 1 **do**
 if i is operator or constant **then**
  Get index from N items before i in v, where N is arity of operator;
  Remove N items before i in v;
 Create node with information from operator and inputs;
 Change i to temporal value with actual size of *chromosome*;
 Add node to end of *chromosome*;

3.4 Ohodnocení jedince v CGP

Při ohodnocení je nejprve proveden rekurzivní sestup, který určí aktivní uzly. Díky nim se zamezí zbytečným výpočtům – stačí uvažovat pouze hodnoty aktivních uzlů. Kvalita jedince je určena na dvou úrovních, přičemž druhá úroveň není vždy potřebná k určení kvality. První úrovní je hodnota fitness vypočítaná dle vzorce 2.5 z odezev na vstupní vektory. V případě, že je hodnota fitness splňuje požadavky selekční metody, přistupuje se k druhé úrovni, pokud ji selekce vyžaduje. Na druhé úrovni je určena velikost fenotypu jedince, která nepřímo úměrně určuje kvalitu jedince.

3.5 Mutace, selekce a ukončení

Jako genetický operátor je u CGP použita tzv. *Skip mutation* [4]. Ta pracuje tak, že kontroluje, zda byl zmutován aktivní uzel. Pokud se tak nestane, nemění se ani hodnota fitness a její výpočet tak není prováděn zbytečně znova.

Selekce se drží teoretického předpokladu a z $1 + \lambda$ jedinců vybere jako rodiče jednoho nejlepšího.

Algoritmus je ukončen po daném počtu generací, neboť nelze zjistit, jaká je velikost nejmenšího fenotypu a ani nelze předpokládat úplnou shodu s trénovacími daty.

3.6 Podstromové CGP

Speciální variantu CGP, kterou jsem navrhl speciálně pro problém redukce počtu aktivních uzlů řešení získaného GSGP, je tzv. *podstromové (Subtree) CGP* (SCGP). Jedinec získaný pomocí GSGP, který je použit jako vstup SCGP, musí splňovat následující:

1. Nepoužívá GSC (pouze GSM a replikaci)
2. Převod do CGP je realizován bez znovupoužití náhodných stromů (algoritmus 2)

Algoritmus 2 Převod GSGP na CGP bez znovupoužití náhodných stromů

//p means the parent of individual, r is a random tree used to create an individual
Input: Generations of populations gp; //a 2D structure to store all populations in //all generations

$cgpIndexRecords$: vector of records
$chromosome$: vector of nodes

for all input variables **do**
 Create an input node and add it to *chromosome*;

for all generations g in gp **do**
 for all individuals i in g **do**
  if i is used to compute output **then**
   if $g == 0$ **then** //init population
    if i was created by GSC **then**
     Convert p_1, p_2 and r to CGP, add to *chromosome* and remember index of p_1, p_2 and r in *chromosome*;
     Add connection nodes to *chromosome*, which connects p_1, p_2, r as GSC;
    if i was created by GSM **then**
     Convert p, r_1 and r_2 to CGP, add to *chromosome* and remember index of p, r_1 and r_2 in *chromosome*;
     Add connection nodes to *chromosome*, which connects p, r_1, r_2 and *mutation_step* as GSM;
    if i was created by replication **then**
     Convert p to CGP, add to *chromosome*;
   else //next generations
    if i was created by GSC **then**
     Get p_1, p_2 indexes from *cgpIndexRecords*;
     Convert r to CGP, add to *chromosome* and remember index of r in *chromosome*;
     Add connection nodes to *chromosome*, which connects p_1, p_2, r as GSC;
    if i was created by GSM **then**
     Get p index from *cgpIndexRecords*;
     Convert r_1, r_2 to CGP, add to *chromosome* and remember index of r_1, r_2 in *chromosome*;
     Add connection nodes to *chromosome*, which connects p, r_1, r_2 and *mutation_step* as GSM;
    if i was created by replication **then**
     Get index from *cgpIndexRecords*;
   Make record to *cgpIndexRecords* with index of last node added to *chromosome*;
Creat an output node conected to the last node and add it to *chromosome*;

Algoritmus 3 Převod GSGP na CGP se znovupoužitím náhodných stromů (část 1/2)

//p means the parent of individual, r is a random tree used to create an individual
Input: Generations of populations gp; //a 2D structure to store all populations in //all generations

$cgpIndexRecords$: vector of records
$chromosome$: vector of nodes
$used$: int[tree count]

for all input variables **do**
    Create an input node and add it to *chromosome*;

for all generations g in gp **do**
    for all individuals i in g **do**
        if i is used to compute output **then**
            if $g == 0$ **then** //init population
                if i was created by GSC **then**
                    Convert p_1, p_2 and r to CGP, add to *chromosome* and remember index of p_1, p_2 and r in *chromosome*;
                    Add connection nodes to *chromosome*, which connects p_1, p_2, r as GSC;
                    if random tree reuse is enabled **then**
                        Save index of r to *used*;
                if i was created by GSM **then**
                    Convert p, r_1 and r_2 to CGP, add to *chromosome* and remember index of p, r_1 and r_2 in *chromosome*;
                    Add connection nodes to *chromosome*, which connects p, r_1, r_2 and *mutation_step* as GSM;
                    if random tree reuse is enabled **then**
                        Save index of r_1, r_2 to *used*;
                if i was created by replication **then**
                    Convert p to CGP, add to *chromosome*;
            else //next generations
                if i was created by GSC **then**
                    Get p_1, p_2 indexes from *cgpIndexRecords*;
                    if random tree reuse is enabled **then**
                        if r index is not saved in *used* **then**
                            Convert r to CGP, add to *chromosome* and remember + save index of r in *chromosome* to *used*;
                        else
                            Get r index from *used*;
                    else
                        Convert r to CGP, add to *chromosome* and remember index of r in *chromosome*;
                    Add connection nodes to *chromosome*, which connects p_1, p_2, r as GSC;
                if i was created by GSM **then**
                    Get p index from *cgpIndexRecords*;

Algoritmus 4 Převod GSGP na CGP se znovupoužitím náhodných stromů (část 2/2)

if random tree reuse is enabled **then**
  if r_1 index is not saved in *used* **then**
    Convert r_1 to CGP, add to *chromosome* and remember + save index of r_1 in *chromosome* to *used*;
  else
    Get r_1 index from *used*;
  if r_2 index is not saved in *used* **then**
    Convert r_2 to CGP, add to *chromosome* and remember + save index of r_2 in *chromosome* to *used*;
  else
    Get r_2 index from *used*;
else
  Convert r_1, r_2 to CGP, add to *chromosome* and remember index of r_1, r_2 in *chromosome*;
Add connection nodes to *chromosome*, which connects p, r_1, r_2 and *mutation_step* as GSM;
if i was created by replication **then**
  Get index from *cgpIndexRecords*;
Make record to *cgpIndexRecords* with index of last node added to *chromosome*;
Create an output node conected to the last node and add it to *chromosome*;

Hlavní myšlenka je v rozdělení chromozomu na menší, sémanticky spjaté, chromozomy obsahujících jednotlivé GSM a původní strom. Díky asociativitě a komutativitě sčítání lze upravovat a vyhodnocovat jednotlivé chromozomy nezávisle, rychleji a lokálněji. Nad každým je tak možné spustit samostatný běh CGP.

Nejprve je výsledek GSGP nepoužívajícího GSC převeden do chromozomu CGP bez znovupoužití náhodných stromů. Následně je chromozom rozdělen na jednotlivé GSM chromozomy tak, jak byly v generacích přidávány. Pro každý chromozom je vytvořen vektor odezev na vstupní vektory, což způsobí značné zrychlení výpočtu hodnoty fitness v pozdější fázi algoritmu. Následuje cyklus, ve kterém se postupně spouští CGP nad každým chromozomem. Pro výpočet hodnoty fitness daného chromozomu je jakožto požadovaný výstup použit výstup z trénovacích dat snížený o odezvy všech ostatních chromozomů. Dále probíhá klasické CGP s parametry (*generations*, *population_size*, *mutation_rate*, *selection_method*). V posledním kroku jsou odstraněny uzly, které nejsou aktivní, a je přepočítán vektor odezev na vstupy pro evolvovaného jedince. Po evoluci chromozomů následuje sjednocení chromozomů po dvou uzlem s operací součtu. Chromozomy jsou vybírány dle zadané strategie *subtree_select_method*. V poslední fázi dojde opět k výpočtu odezev všech chromozomů na vstupní vektory. Cyklus evoluce chromozomů za pomoci CGP a následné spojování probíhá tak dlouho, dokud nezbude jediný chromozom, který prošel evolucí (alg. 5).

Algoritmus 5 SCGP (část 1/2)

Input: $trainData$;
$generations_of_populations$;
$generations$;
$population_size$;
$mutation_rate$;
$selection_method$;
$subtree_select_method$;

Known: $inputCount$; //from trainData

Create chromosome $c\,(c_0, \ldots, c_{m-1})$ from GSGP without GSC and without subtree reuse from $generations_of_populations$ (algorithm 2);
Set the actual node index act as an input to the output node, $act \Leftarrow c_{m-1}.in_1$;

//Get indexes of mutation subtrees and index of generation 0 tree
Create empty index vector $indexes()$;
while TRUE **do**
  if operation in c_{act} is Addition AND $c_{act}.in_2$ is root of mutation subtree **then**
    Add $c_{act}.in_2$ into $indexes$;
    $act \Leftarrow c_{act}.in_1$;
  else
    Add act to $indexes$;
    break;
//$indexes\,(index_0, \ldots, index_{k-1})$ created

//Create partial chromosomes - subchromosomes with $indexes$ as outputs
Vector of empty partial chromosome vectors $pc\,(pc_0, \ldots, pc_{k-1})$; //the same size as //$indexes$
for all $index_a$ in $indexes$ **do**
  Add inputs $c_0, \ldots, c_{inputCount-1}$ to pc_a;
  Get active nodes for subtree with the root in c_{index_a} AND index them from $inputCount$ into $NewIndex$;
  for all active node N in c_{index_a} **do**
    $N.in_1 \Leftarrow newIndex$ for $N.in_1$; //if $newIndex$ is not specified, can be used random
    $N.in_2 \Leftarrow newIndex$ for $N.in_2$; //in active nodes range
    Add N to pc_a;
  Add output node to pc_a and connect it;

//Create partial fitnesses - precomputed trainData values for every partial chromosome
Vector of empty partial fitness vectors $pf\,(pf_0, \ldots, pf_{k-1})$; //the same size as $indexes$ //and pc
for all pc_a in pc **do**
  for all data row dr in $trainData$ **do**
    Compute the output value of pc_a for dr and add it to pf_a;

Algoritmus 6 SCGP (část 2/2)

$fitness_{start} \Leftarrow$ compute fitness of c for $trainData$;
Create difference vector $diff\,(d_0, \ldots, d_{l-1})$, where l is the data row count in trainData;
repeat
  for all pc_a in pc **do**
    for all d_b in $diff$ **do**
      $d_b \Leftarrow trainData[b].output - \sum_{i=0, i \neq a}^{k-1} pf_{i\,b}$;

    Compute fitness of pc_a for $trainData$, but for output comparsion use $diff$ vector;
    $pc_a \Leftarrow$ Run standart CGP for pc_a with parameters: *generations*, *population_size*, *mutation_rate*, *selection_method* and using $diff$ vector for fitness;
    Remove inactive nodes in pc_a and restore connections (similar to partial chromosome creation);
    for all data row dr from $trainData$ **do**
      Compute output of pc_a on dr and save it to pf_a on place (on dr index);

  //Join of partial chromosomes
  $pc_{size} \Leftarrow k$;
  Empty vector of empty partial chromosome vectors $pc'\,()$;
  while $pc_{size} > 1$ **do**
    Select partial chromosomes S_a, S_b from pc, depending on *subtree_select_method*;
    Chromosome $S \Leftarrow S_a$ joined by Addition node with S_b;
    $pc_{size} \Leftarrow pc_{size} - 2$;
    Remove S_a, S_b from pc;
    Add S to pc';
  if pc_{size}==1 **then**
    Add pc_0 to pc';
  $pc \Leftarrow pc'$; $//pc'(pc'_0, \ldots, pc'_{k'-1}), k'$ is $\lceil \frac{k}{2} \rceil$

  Vector of empty partial fitness vectors $pf'\,(pf'_0, \ldots, pf'_{k-1})$;
  for all pc_a in pc **do**
    for all data row dr in $trainData$ **do**
      Compute the output value of pc_a for dr and add it to pf'_a;
  $pf \Leftarrow pf'$;
until k changed;

Kapitola 4

Implementace

V této kapitole jsou popsány hlavní části implementace spolu s nastavením a parametry programu. Implementace je provedena v jazyce C++ se standardem C++11 a základní funkce jsou zakomponovány přímo do programu GSGP.

4.1 Funkce vsazené do programu GSGP

Využití původního kódu GSGP je základním stavebním kamenem výsledného programu. Pro minimalizaci zásahu do původního programu byly vytvořeny pouze dvě funkce, které jsou volány v těle hlavního programu (plus vložení souboru s těmito funkcemi). První je funkce načtení parametrů a případného spuštění CGP/SCGP/evaluace, druhá je převod výsledku jedince GSGP na instanci CGP.

4.1.1 Načtení parametrů a případné spuštění vlastní evoluce a evaluace

Po načtení konfiguračního souboru GSGP je volána funkce *void cgp_settings(int argc, const char **argv, char *train_file, char *test_file)* (soubor *CGP.h*). Nejprve je načteno nastavení ze souboru *cgp_config.ini* a následně jsou prohledány vstupní argumenty programu *argv*. Indikuje se v nich první výskyt jednoho z přepínačů *-cgp*, *-scgp*, *-use*. V případě, kdy se některý z nich v argumentech nachází, se vytvoří instance třídy CGP, předají se jí názvy souborů *train_file* a *test_file* a zašle se zpráva pro vyvolání příslušné metody. Po skončení vyvolané metody je celý program ukončen.

4.1.2 Převod výsledku jedince GSGP na instanci CGP

Po uložení výsledku GSGP nastává volání funkce *void cgp_upgrade(population **p)* (soubor *gsgpToCGP.h*). Tato funkce převede vstupní generace populací GSGP dle algoritmu 2 na instanci CGP bez znovupoužití náhodných stromů a dle algoritmu 3 na instanci CGP se znovupoužitím náhodných stromů. Implementace těchto dvou algoritmů je spojena v jednu funkci *createChromosomeFromGSGP(population **p, vector<cgp_node> *chromosome, bool isRandomTreeReused)*, kde volba algoritmu převodu závisí na parametru *isRandomTreeReused*. Nakonec jsou oba výsledky se statistikami uloženy pomocí metod třídy CGP *saveCGPAfterGSGP(std::string suffix)* a *saveStatisticsAfterGSGP(std::string suffix)*.

4.2 Třída CGP

Hlavní částí vlastní implementace je třída CGP, jejíž deklarace se nachází v souboru *CGP.h*. Obsahuje dva konstruktory (základní bezparametrový nebo s názvy souborů s trénovacími a testovacími daty, které načítá), tři hlavní funkční metody (*runCGP()*, *runSubtreeCGP()*, *runEvaluation()*) a několik vstupních, výstupních a pomocných metod. Důležitou součástí je i využití třídy Chromosome, která se stará o úpravu chromozomů CGP, SCGP a o matematické a genetické operace nad nimi (soubor *Chromosome.h*). Použité typy uzlů jsou stejné jako u GSGP (konstanta, $+$, $-$, $*$, $/$, $sig(x)$) doplněné o e^x pro ulehčení náhrady sigmoidy.

Metoda *runCGP()* obsahuje klasickou implementaci CGP dle [9] doplněnou o sběr statistických dat v průběhu evoluce, zkracování chromozomu na určený násobek velikosti aktivní části chromozomu a možnost změny pravděpodobnosti mutace. Metoda *runSubtreeCGP()* je implementací algoritmu 5 s uložením statistik po každém sjednocení chromozomů. Poslední funkční metoda, *runEvaluation()*, pouze načte již vytvořenou instanci CGP a provede její evaluaci.

4.3 Vstup a výstup

Původní chování programu provádějícího GSGP je nezměněno i zde. Jedná se o konfigurační soubor *configuration.ini* a argumenty *-train_file* a *-test_file*. Přidán je soubor *cgp_config.ini* s nastavením CGP a GSGP a možné argumenty *-cgp* (pro CGP), *-scgp* (pro SCGP), *-use* (pro evaluaci).

4.3.1 Argumenty

Program lze spustit s následujícími argumenty:
`-train_file` *filename* – Cesta k souboru *filename* s trénovacími daty. Formát obsahu souboru je popsán v podkapitole 4.3.3.
`-test_file` *filename* – Cesta k souboru *filename* s testovacími daty. Formát obsahu souboru je popsán v podkapitole 4.3.3.
`-cgp` – Spuštění klasického CGP, které upravuje již vytvořenou instanci ve formátu CGP (např. výsledek GSGP nebo SCGP).
`-scgp` – Spuštění SCGP.
`-use` – Spuštění evaluace zadané instance ve formátu CGP.
Argumenty *-train_file* a *-test_file* jsou povinné. Při nepoužití žádného dalšího argumentu je spuštěno GSGP. Při použití více nepovinných argumentů je vybrán první (první vlevo).

4.3.2 Parametry

Původní soubor s parametry pro GSGP, *configuration.ini*, vypadá následovně (popis převzat z `http://gsgp.sourceforge.net/wp-content/uploads/2013/07/docv2.rar`):
`population_size = 2000` – Velikost populace. Spolu s `random_tree` udává počet vygenerovaných stromů při inicializaci.

`max_number_generations = 1000` – Počet generací, po kolika bude evoluce ukončena.

`init_type = 2` – Typ inicializace. 0 značí metodu Grow, 1 Full a 2 je Ramped half-and-half.

`p_crossover = 0` – Pravděpodobnost použití GSC. Hodnota musí být kladná a zároveň `p_crossover` + `p_mutation` ≤ 1.

`p_mutation = 0.9` – Pravděpodobnost použití GSM. Hodnota musí být kladná a zároveň `p_mutation` + `p_crossover` ≤ 1.

`max_depth_creation = 8` – Maximální výška stromu při inicializaci.

`tournament_size = 4` – Velikost turnaje.

`zero_depth = 0` – Udává, zda používat jednouzlové jedince při inicializaci (0 ne, 1 ano).

`mutation_step = 1` – Velikost mutačního kroku u GSM. Není ale v programu využito, místo něj je používána náhodně vygenerovaná hodnota.

`num_random_constants = 0` – Počet náhodných konstant.

`min_random_constant = -10` – Dolní hranice pro náhodné konstanty.

`max_random_constant = 10` – Horní hranice pro náhodné konstanty.

`minimization_problem = 1` – Přepínání mezi typem problému. 1 značí úlohu minimalizace, 0 maximalizace.

`random_tree = 500` – Počet náhodných stromů. Spolu s `population_size` udává počet vygenerovaných stromů při inicializaci.

`expression_file = 0` – Možnost načtení jedinců ze souboru. 0 značí klasickou inicializaci bez načítání, 1 indikuje načtení ze souboru *individuals.txt*.

`USE_TEST_SET = 0` – Umožnění testování a evaluace na již vytvořeném výstupu. 0 je hodnota indikující evoluci, 1 značí provedení evaluace testovacích dat na již vytvořeném modelu.

Druhý soubor s parametry pro CGP a GSGP, *cgp_config.ini*, obsahuje:
`/chromosome_file_path: ./chromosome/data` – Cesta k souboru obsahujícímu uloženou instanci CGP, která bude načtena.

`/output_file_path: ./output/data` – Cesta a název souboru, který bude obsahovat výsledek evoluce. Soubory se stejným počátkem názvu mohou být vytvořeny pro mezivýsledky a statistiky.

`generations: 50` – Počet generací, po kolika bude evoluce ukončena.

`population_size: 8` – Velikost populace.

`mutation_rate: 0.08` – Pravděpodobnost mutace.

`chromosome_stretch_rate_by: 0.1` – O jaký násobek aktuální velikosti chromozomu se má chromozom zvětšit (používáno po výběru pouze aktivních uzlů do nového chromozomu).

`increasing_mutation: false` – Povolení postupně se zvětšující pravděpodobnosti mutace po neúspěšném zlepšení jedince. Hodnota 1 nebo True indikuje povolení, ostatní hodnoty tuto funkci zakazují. Funkční pouze pro CGP, SCGP neovlivňuje.

`mutation_increase_step: 1.05` – Činitel pravděpodobnosti mutace při jejím zvětšování.

`mutation_ceil: 0.1` – Maximální hodnota pravděpodobnosti mutace. Po úspěšném zlepšení jedince se opět vrátí na základní hodnotu.

`best_selection(0=fitness;1=length): 1` – Metoda výběru nejlepšího jedince. 0 značí výběr jedince s nejmenší hodnotou fitness bez ohledu na velikost, 1 udržuje fitness pod původní hodnotou a minimalizuje velikost řešení.

`join_selection(0=random;1=max_diff;2=min_diff): 1` – Metoda výběru chromozomů pro spojování v poslední části algoritmu SCGP. 0 značí náhodný výběr, 1 je výběr chromozomů s co největším možným rozdílem aktivních velikostí (postupně se zmenšuje) a 2 je naopak výběr jedinců s minimálním rozdílem aktivních velikostí (postupně se zvětšuje).

4.3.3 Trénovací a testovací data

Soubory s trénovacími a testovacími daty mají stejnou strukturu. Jedná se o textové soubory, na jejichž prvním řádku se nachází počet prvků ve vstupním vektoru (počet sloupců bez posledního sloupce, který obsahuje výstupní hodnotu funkce). Druhý řádek značí počet trénovacích/testovacích vektorů (řádků bez dvou) v souboru. Zbylé řádky jsou samotné vektory, jejichž prvky jsou odděleny znakem tabulátoru.

4.3.4 Výstup

Výstupem GSGP je v základu několik souborů se statistikou, soubor *individuals.txt* se stromy a soubor *trace.txt* s uloženými operacemi nad těmito stromy v jednotlivých generacích. Rozšíření o převod na instanci CGP přináší i uložení statistik do souborů *stat.txt* a *stat_compressed.txt*, stejně jako uložení této instance v původní i komprimované verzi se znovupoužitím náhodných stromů. Název odpovídá parametru `/chromosome_file_path` z *cgp_config.ini*, případně je doplněn o '_compressed'.

CGP ukládá statistiky a nejlepšího jedince na začátku evoluce, po skončení a při každém zlepšení. Název odpovídá parametru `/output_file_path` z *cgp_config.ini*, soubor se statistikou má příponu '.cgp_stat' a uložený jedinec '_gN.cgp', kde N značí generaci.

Výstup GSGP funguje podobně jako u CGP, jen v příponách je '.cgp_stat' nahrazeno '.stcgp_stat' a '_gN.cgp' nahrazeno '_sN.stcgp'. Ukládání také neprobíhá vždy v generaci, kdy je splněna podmínka na zlepšení, ale pokaždé, když je provedeno sjednocení chromozomů.

Výsledek evaluace je vypsán pouze na standardní výstup.

4.4 Překlad a spuštění

Překlad probíhá pomocí kompilátoru `g++` s parametry `-std=c++11 -Wall -O0 -g`. Program byl testován pouze na Unixových operačních systémech a obsahuje knihovnu pro měření času `<sys/time.h>`, která nemusí na jiných operačních systémech fungovat. Ke zdrojovým kódům je dodán makefile, který umožňuje kompilaci (`make` / `make cmp`), smazání zkompilovaného souboru (`make clean`) a spuštění přidaných skriptů. Skript *script_gsgp.sh* (`make rungsgp`) spustí 20× paralelně GSGP. Obdobně tomu je pro CGP a skript *script_cgp.sh* (`make runcgp`) a SCGP a skript *script_scgp.sh* (`make runscgp`. Poslední skript *script_use.sh* (`make runuse`) spustí vyhodnocení získaného řešení nad všemi trénovacími a testovacími soubory. Ve všech těchto skriptech je potřeba nezapomenout nastavit příslušné datové sady, se kterými se pracuje.

Kapitola 5

Experimentální vyhodnocení metody

Cílem této kapitoly je navrhnout, provést a vyhodnotit experimenty na podstromovém kartézském genetickém programování. Tyto experimenty jsou zaměřené na vyhodnocení účinnosti metody a na optimalizace nastavení parametrů algoritmu.

5.1 Trénovací a testovací data

Jako testovací a trénovací data byly využity stejné datové sady (dále jen datasety) jako při testech geometrického sémantického genetického programování. Jedná se o čtyři úlohy z oblasti farmakokinetiky[1]. První tři úlohy jsou podrobněji popsány v [1] a jsou jimi lidská orální biodostupnost *bioav* (v článku označeno *%F*), medián smrtící dávky – toxicita *Tox* (v článku označeno *LD50*) a úroveň vazby plazmatických bílkovin *PPB* (v článku označeno *%PPB*). Poslední úloha je popis 3D struktury proteinu *P3D*. Rozměry datasetů jsou uvedeny v tabulce 5.1 (sloupců uvedeno o jeden méně, neboť v posledním sloupci se nachází výstup). Každý dataset je 30× rozdělen na různé trénovací a testovací části v poměru 70:30. Pro experimenty bylo náhodně vybráno jedno rozdělení (číslo 6) a všechna rozdělení byla použita pro otestování optimalizovaného řešení vytvořeného pomocí SCGP. Z odezvy na data použitá k ohodnocení jedince při evoluci vznikne hodnota *trénovací fitness*. Ohodnocení jedince na jiných než trénovacích datech je nazváno hodnotou *testovací fitness*.

	vstupní parametry (sloupce-1)	záznamy (řádky)
bioav	241	359
PPB	628	131
Tox	626	234
P3D	9	45730

Tabulka 5.1: Rozměry použitých datasetů

[1] Farmakokinetika je obor farmakologie, který zkoumá působení léčiv v těle. [http://www.webster-dictionary.org/definition/Pharmacokinetics]

5.2 Nastavení parametrů

Parametry pro GSGP jsou nastaveny experimentálně. V případě úlohy *P3D* je zvolen menší počet generací kvůli většímu datasetu a tím i zvýšenému počtu evaluací a potřebného výpočetního času. Hlavní parametry GSGP jsou zaneseny v tabulce 5.2.

S částí parametrů SCGP je experimentováno – s počtem generací a s metodou selekce podstromů. Ostatní jsou zvoleny fixně (tabulka 5.3).

Poslední nastavení je pro CGP, které je použito na výsledek SCGP (tabulka 5.4).

Velikost populace	2000	Pravděpodobnost mutace	0,9
Počet generací	1000 (300 pro *P3D*)	Maximální hloubka stromu	8
Počet náhodných stromů	500	Použití jednouzlových jedinců	Ne
Metoda tvorby stromů	Ramped Half-and-half	Velikost turnaje	4
Pravděpodobnost křížení	0	Počet náhodných konstant	0

Tabulka 5.2: Parametry GSGP

Počet generací	10/50/100	Zvětšení chromozomu	0,1
Velikost populace	8	Nejlepší jedinec	Minimální velikost
Pravděpodobnost mutace	0,08	Metoda selekce podstromů	Největší rozdíl/ Nejmenší rozdíl/ Náhodný výběr

Tabulka 5.3: Parametry SCGP

Počet generací	10000	Zvětšení chromozomu	0,1
Velikost populace	8	Nejlepší jedinec	Minimální velikost
Pravděpodobnost mutace	0,08	Zvětšení pravděpodobnosti mutace	Ne

Tabulka 5.4: Parametry CGP

5.3 Experimenty

Experimenty proběhly na ostravském superpočítači Salomon (`http://www.it4i.cz/`). Pro každou úlohu bylo spuštěno 40× GSGP a pro další experimenty byl vybrán jedinec s nejmenší hodnotou fitness. Pro experimenty s SCGP a CGP bylo provedeno vždy 20 nezávislých běhů a data z nich jsou interpretována ve formě krabicových grafů (boxplotů).

Každý výsledek evolučního běhu je popsatelný trojicí (velikost řešení, tj. počet aktivních uzlů; hodnota trénovací fitness; hodnota testovací fitness). Velikost řešení po proběhnutí SCGP je porovnána s velikostí řešení získaného pomocí GSGP a kompresního převodu na jedince CGP. Kompresním převodem rozumíme převod GSGP na CGP se znovupoužitím

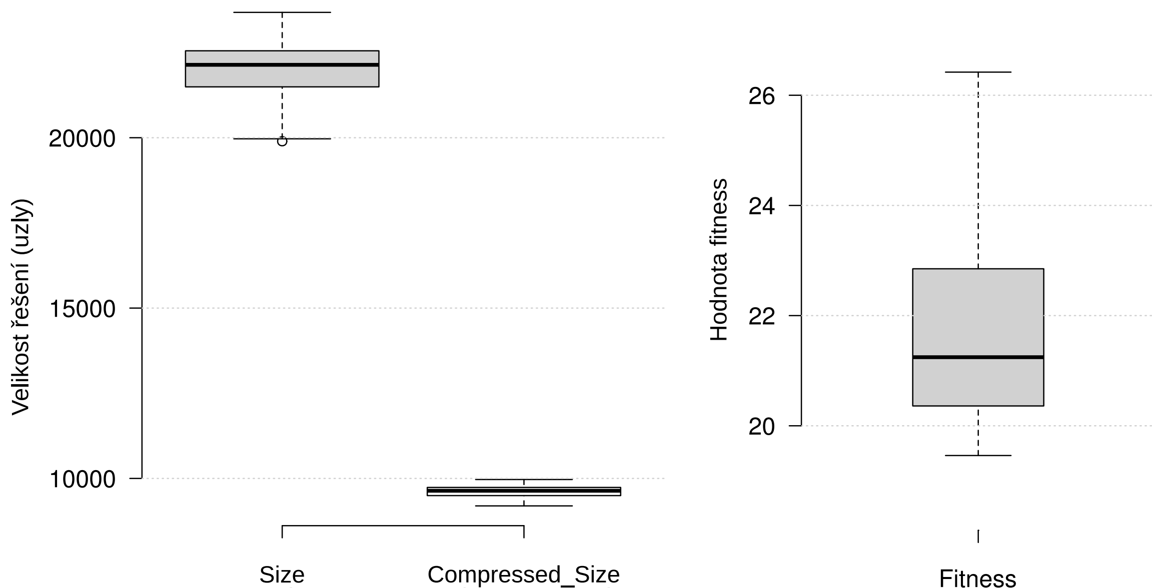
náhodných stromů (algoritmus 3). Pro hodnotu testovací fitness jsou v experimentech použity dvě meze pod kterými se má udržet – stejná hodnota jako výstup GSGP (hlavní mez) a dvojnásobek této hodnoty (krajní mez). Dle hlavní meze se určí, kolik řešení je stejně kvalitních jako řešení pomocí GSGP. Krajní mez je použita pro detekci řešení, která nedokážou vůbec zobecňovat. V experimentech je uváděno jak nejlepší řešení (co nejmenší a zároveň splňuje i podmínku hlavní meze), tak i úplně nejmenší řešení (bez ohledu na hodnotu testovací fitness).

S klasickým kartézským genetickým programováním bylo experimentováno ve třech fázích. První je použití na náhodně vygenerovanou počáteční populaci. Výstup je pak zhruba stejně dobrý jako u genetického programování, tudíž horší než řešení získané GSGP. Druhá možnost je počáteční populaci vytvořit z výstupu GSGP a optimalizovat jej. Takto vytvoření jedinci mají ale tisíce až statisíce uzlů a operace nad nimi jsou časově náročné (zejména výpočet hodnoty fitness) a za hodinu je provedena evaluace jen několik stovek, maximálně pár tisíců generací (při velikosti populace 8). Navíc za tuto dobu nebylo zaznamenáno jediné zlepšení. Poslední možností je použít CGP na nejlepší výstup SCGP. Chromozomy jedinců jsou již zmenšeny a tak evaluace probíhá rychleji. Kandidátní řešení teď projde více generacemi za méně času a je tak prohledán větší stavový prostor.

5.3.1 Výstup geometrického sémantického programování

První z experimentů je pouze ukázkou kvality výstupu GSGP. Použité parametry jsou popsány v tabulce 5.2.

Výsledky pro *bioav* jsou zobrazeny na sjednoceném grafu 5.1. Nejlepší jedinec má hodnotu fitness 19,455. Velikost tohoto řešení je 22285 uzlů a nebo 9863 uzlů při kompresním převodu.



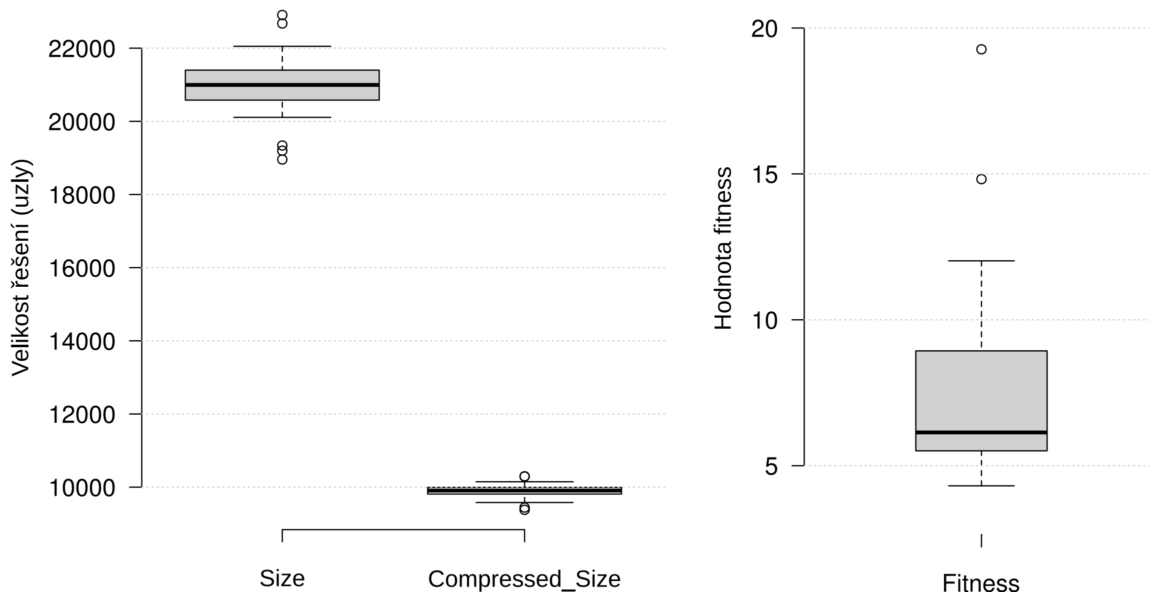
Obrázek 5.1: Výstup GSGP pro úlohu *bioav*

Graf 5.2 ukazuje výsledky úlohy *PPB*. Nejmenší dosažená hodnota fitness je 4,312 a řešení je vytvořeno z 22907 uzlů, případně 10291 uzly při kompresi.

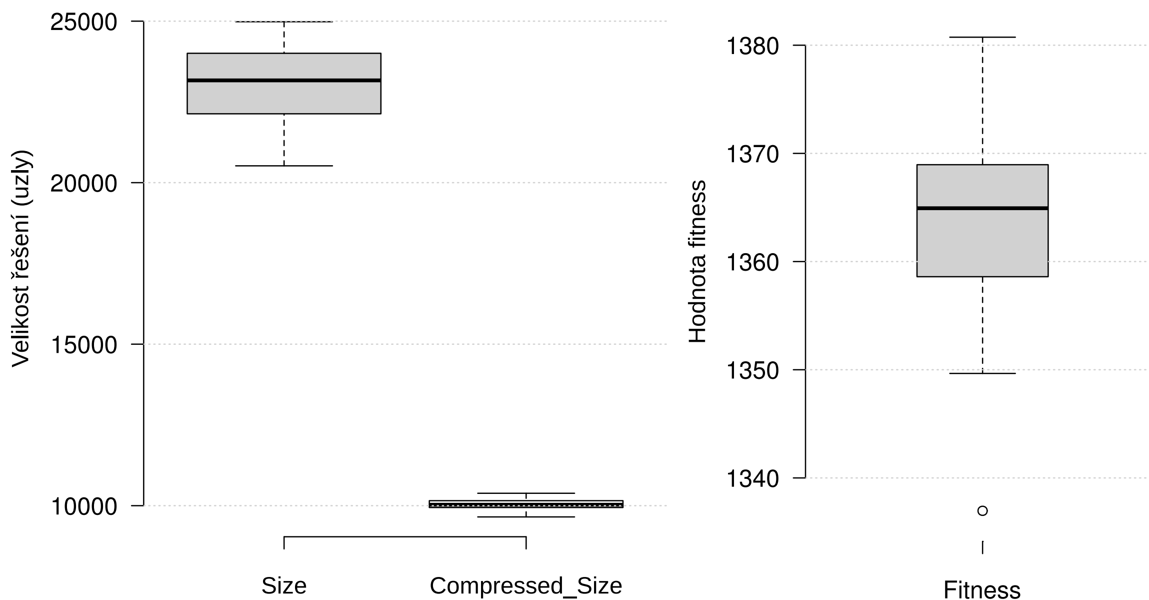
Úloha *Tox* (výsledky na grafu 5.3) má nejlepší nalezené řešení s hodnotou fitness

1336,960, velikostí 21568 uzlů a kompresní velikostí 9934 uzlů.

Statistické výsledky pro úlohu *P3D* jsou na grafu 5.4. Nejlepší řešení má hodnotu fitness 3,981, velikost 5764 uzlů a 3992 uzlů při kompresi.



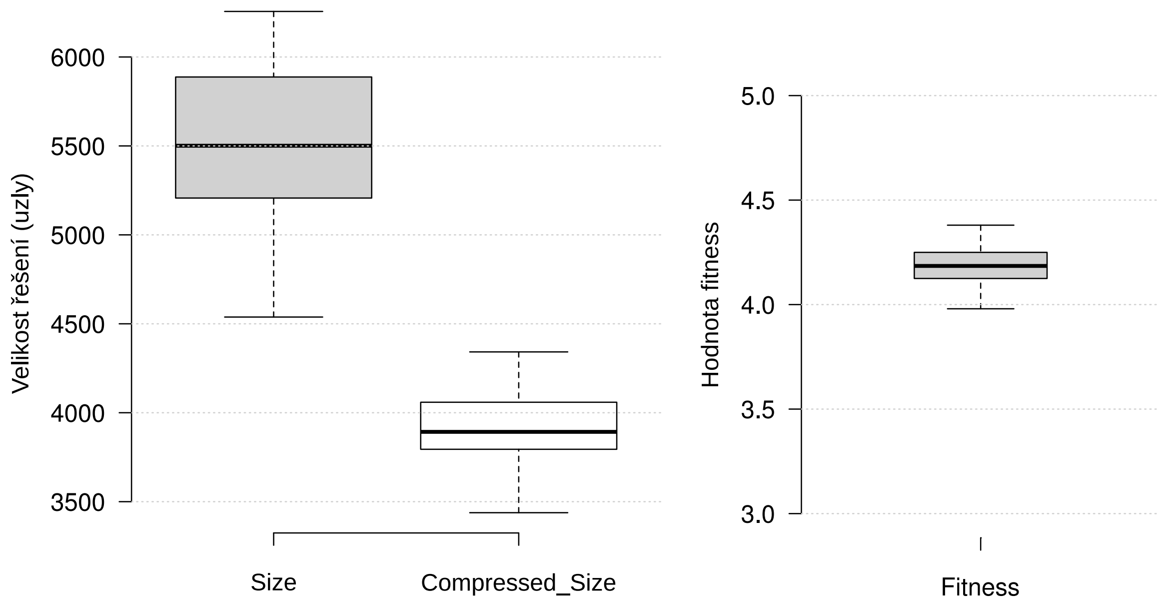
Obrázek 5.2: Výstup GSGP pro úlohu *PPB*



Obrázek 5.3: Výstup GSGP pro úlohu *Tox*

5.3.2 Změna počtu generací v SCGP

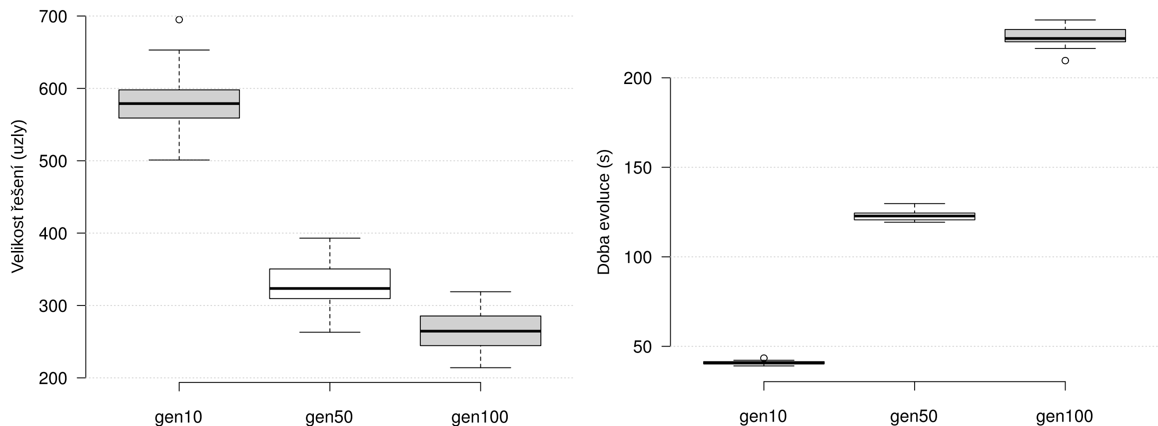
Prvním experimentem s SCGP je vliv počtu generací na chování algoritmu. Nastavení je uvedeno v tabulce 5.3 a jako metoda selekce podstromů je zvolen náhodný výběr.



Obrázek 5.4: Výstup GSGP pro úlohu *P3D*

Statistické výsledky úlohy *bioav* jsou k vidění na grafu 5.5. V tabulce 5.5 jsou zaneseny hlavní hodnoty pro jednotlivé nastavení. Nejlepší řešení pro 10, 50 a 100 generací jsou v poměru velikostí 2,7:1,33:1. Čím více generací je nastaveno, tím lepší řešení jsou nacházena – nejen dle velikosti řešení, ale také dle počtu řešení spadajícího pod dané meze.

Výsledky běhů programu pro úlohu *PPB* jsou znázorněny na grafu 5.6 a klíčové hodnoty v tabulce 5.6. Poměr velikostí mezi nejlepšími jedinci z různého počtu generací je 4,44:1,26:1. Čím více generací, tím lepší řešení jsou nacházena (s výjimkou nalezení pouze jednoho řešení alespoň tak dobrého, jako řešení GSGP).

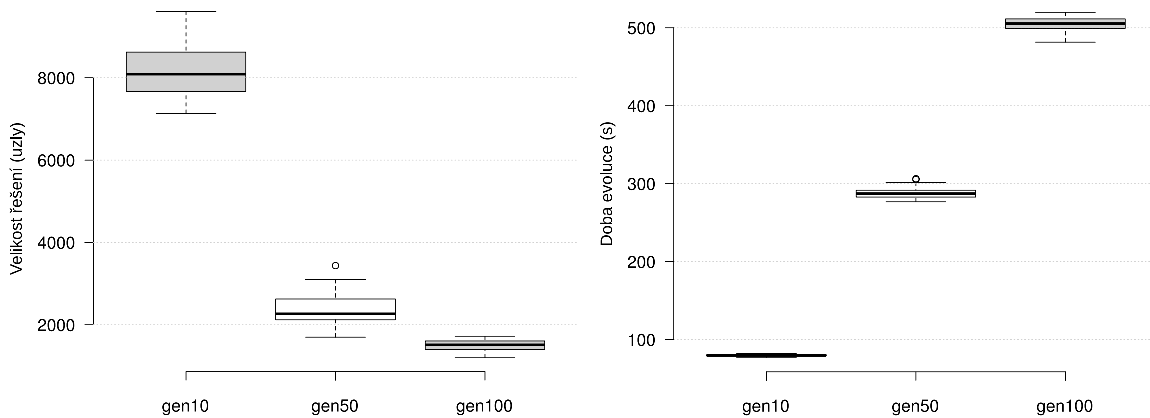


Obrázek 5.5: Vliv různého počtu generací u *bioav* pro SCGP

Na grafu 5.7 si lze povšimnout, že všechna řešení pro úlohu *Tox* jsou mnohonásobně menší než po kompresním převodu. Z tabulky 5.7 lze vyčíst poměr velikostí nejlepších jedinců 2,71:1,12:1. Počet jedinců pod mezemi není tentokrát přímo úměrný počtu generací a kvalita řešení s nimi není spjata.

	10 generací	50 generací	100 generací
Nejlepší řešení	(653; 19,46; 26,51)	(322; 19,45; 26,39)	(242; 19,40; 26,65)
Nejmenší řešení	(501; 19,45; 47,20)	(263; 19,45; 30,61)	(214; 19,36; 32,49)
Pod hlavní mezí	1	2	5
Pod krajní mezí	13	16	19

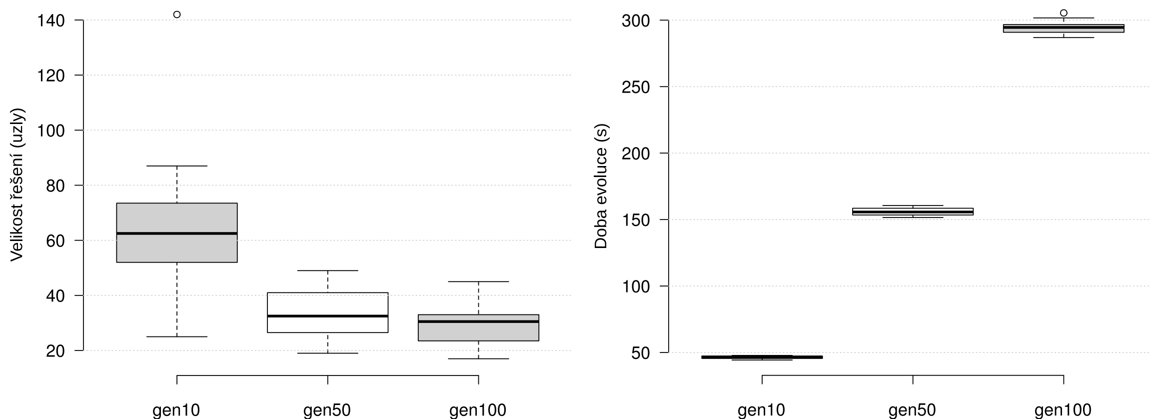
Tabulka 5.5: Vliv různého počtu generací u *bioav* pro SCGP



Obrázek 5.6: Vliv různého počtu generací u *PPB* pro SCGP

	10 generací	50 generací	100 generací
Nejlepší řešení	(7643; 4,29; 30,04)	(2178; 4,29; 28,86)	(1723; 4,26; 26,80)
Nejmenší řešení	(7137; 4,30; 31,16)	(1700; 4,29; 37,91)	(1198; 4,31; 53,24)
Pod hlavní mezí	3	3	1
Pod krajní mezí	9	14	16

Tabulka 5.6: Vliv různého počtu generací u *PPB* pro SCGP

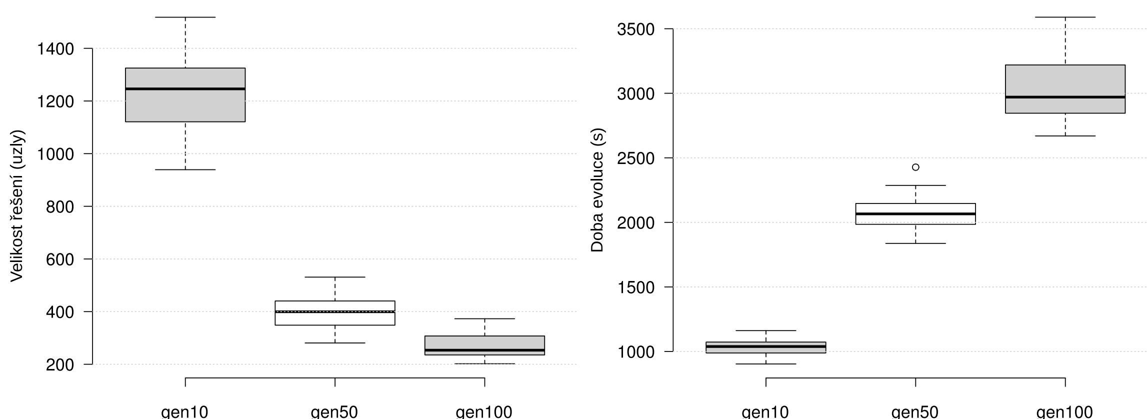


Obrázek 5.7: Vliv různého počtu generací u *Tox* pro SCGP

	10 generací	50 generací	100 generací
Nejlepší řešení	(46; 1334,80; 1489,06)	(19; 1333,02; 1492,06)	(17; 1327,21; 1473,77)
Nejmenší řešení	(25; 1334,58; 1511,61)	(19; 1333,02; 1492,06)	(17; 1327,21; 1473,77)
Pod hlavní mezí	11	13	9
Pod krajní mezí	20	19	15

Tabulka 5.7: Vliv různého počtu generací u *Tox* pro SCGP

Graf 5.8 ukazuje výsledky pro úlohu *P3D*. Tabulka klíčových hodnot 5.8 ukazuje, že velikost nejlepších jedinců je v poměru 4,65:1,39:1. Téměř všechna řešení jsou pod hlavní mezí a úplně všechna se vejdou pod krajní mez.



Obrázek 5.8: Vliv různého počtu generací u *P3D* pro SCGP

	10 generací	50 generací	100 generací
Nejlepší řešení	(939; 3,98; 3,99)	(281; 3,97; 3,98)	(202; 3,98; 3,98)
Nejmenší řešení	(939; 3,98; 3,99)	(281; 3,97; 3,98)	(202; 3,98; 3,98)
Pod hlavní mezí	20	18	18
Pod krajní mezí	20	20	20

Tabulka 5.8: Vliv různého počtu generací u *P3D* pro SCGP

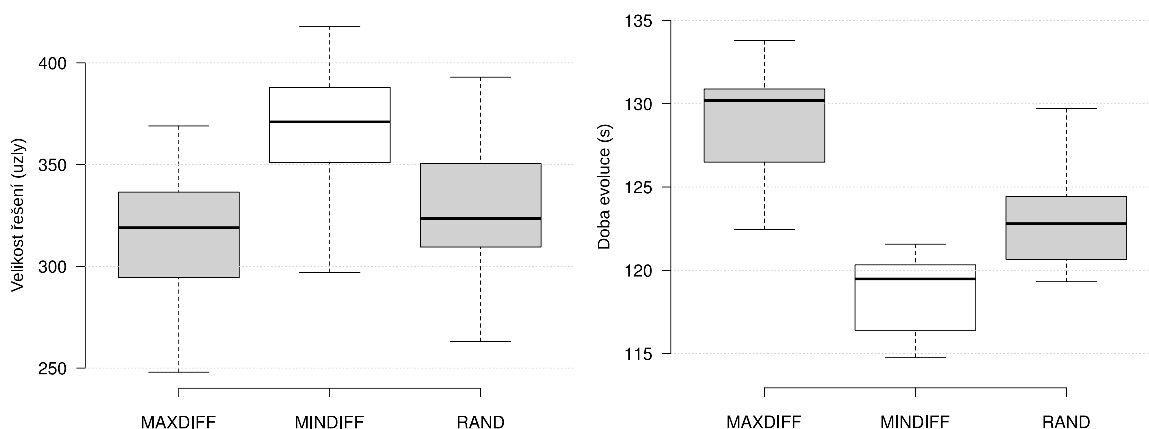
Experimenty s počtem generací ukázaly, že zvýšení počtu generací nemusí vždy zvýšit počet kvalitních řešení, která jsou nalezena, ale vždy naleznou menší řešení – jak absolutně, tak i dle mediánu nebo průměru. Všechna nalezená řešení pomocí SCGP jsou menší než řešení vytvořená kompresním převodem. U prvních tří úloh se doba běhu algoritmu ukázala jakožto lineárně narůstající vzhledem k počtu generací. U úlohy *P3D* závisí doba běhu nejen na zvoleném počtu generací, ale kvůli velkému počtu záznamů v trénovací množině také na velikosti jedince. Čím rychleji a čím víc se jedinec zmenší, tím kratší bude doba běhu.

5.3.3 Změna metody selekce podstromů v SCGP

Dalším experimentem s nastavením je vliv metody selekce na chování algoritmu SCGP. Nastavení je zapsáno v tabulce 5.3 s tím, že jako počet generací je zvolena hodnota 50.

Výsledky úlohy *bioav* jsou zaneseny v grafu 5.9. Klíčové hodnoty z tabulky 5.9 ukazují na vztah velikostí u nejlepších řešení. Výsledky pro největší rozdíl, nejmenší rozdíl a náhodný výběr jsou v poměru 1:1,33:1,12. Rozdíly v počtu řešení pod mezemi pro jednotlivé metody nejsou významné.

Graf 5.10 ukazuje statistické výsledky pro úlohu *PPB*, stejně tak i tabulka 5.10. Velikosti nejlepších řešení jsou v poměru 1:2,07:1,2 a rozdíly v počtu řešení pod mezemi nejsou významné.



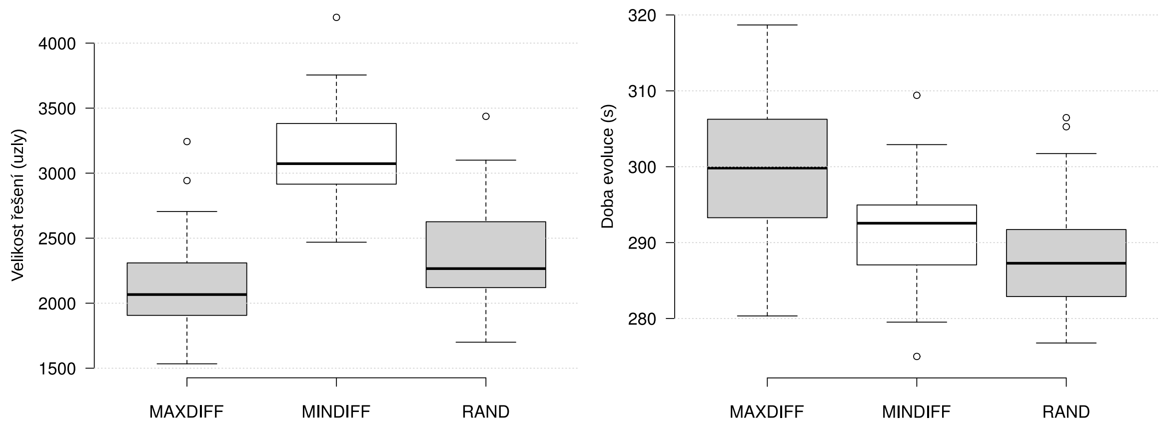
Obrázek 5.9: Vliv výběru metody selekce podstromů u *bioav* pro SCGP

	Největší rozdíl	Nejmenší rozdíl	Náhodný výběr
Nejlepší řešení	(288; 19,45; 25,38)	(383; 19,45; 26,69)	(322; 19,45; 26,39)
Nejmenší řešení	(248; 19,45; 29,74)	(297; 19,43; 32,68)	(263; 19,44; 30,61)
Pod hlavní mezí	3	1	2
Pod krajní mezí	18	16	16

Tabulka 5.9: Vliv různého počtu generací u *bioav* pro SCGP

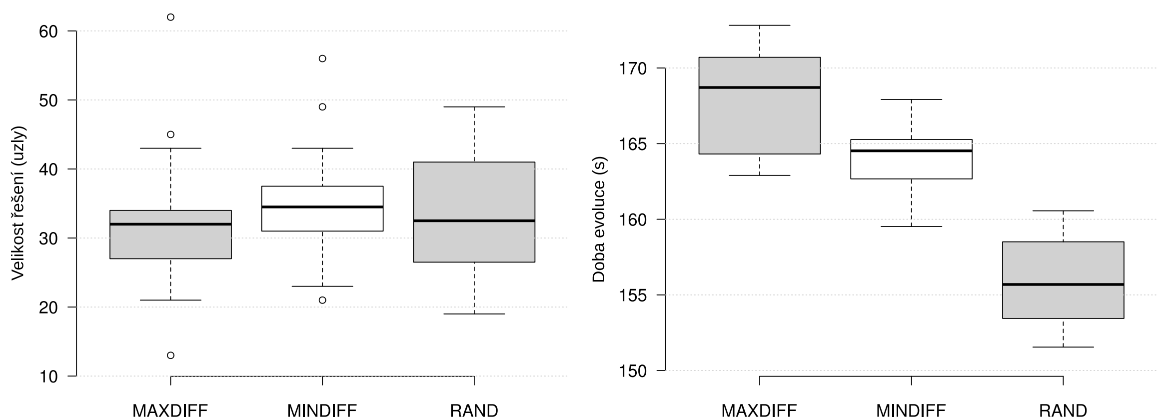
	Největší rozdíl	Nejmenší rozdíl	Náhodný výběr
Nejlepší řešení	(1816; 4,29; 30,67)	(3755; 4,22; 28,94)	(2178; 4,29; 28,86)
Nejmenší řešení	(1534; 4,29; 33,77)	(2469; 4,28; 33,15)	(1700; 4,29; 37,91)
Pod hlavní mezí	1	1	3
Pod krajní mezí	16	15	14

Tabulka 5.10: Vliv různého počtu generací u *PPB* pro SCGP



Obrázek 5.10: Vliv výběru metody selekce podstromů u *PPB* pro SCGP pro SCGP

Na grafu 5.11 jsou zobrazeny výsledky měření různých metod selekce podstromů pro úlohu *Tox*. Z tabulky 5.11 lze určit poměr velikostí nejlepších řešení na 1:0,78:0,7. Počet řešení spadajících pod meze je relativně vysoká.



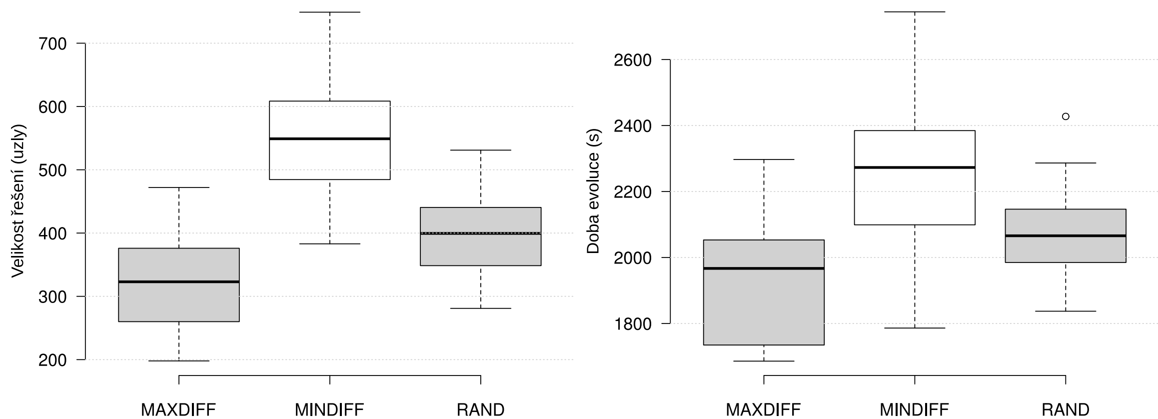
Obrázek 5.11: Vliv výběru metody selekce podstromů u *Tox* pro SCGP

	Největší rozdíl	Nejmenší rozdíl	Náhodný výběr
Nejlepší řešení	(27; 1334,03; 1406,88)	(21; 1308,45; 1382,91)	(19; 1333,02; 1492,06)
Nejmenší řešení	(13; 1333,53; 1501,03)	(21; 1308,45; 1382,91)	(19; 1333,02; 1492,06)
Pod hlavní mezí	11	8	13
Pod krajní mezí	18	15	19

Tabulka 5.11: Vliv různého počtu generací u *Tox* pro SCGP

Výsledky měření pro úlohu *P3D* jsou zaneseny v grafu 5.12 a v tabulce 5.12. Poměr velikostí nejlepších řešení je 1:2,31:1,42. Všechna řešení spadají alespoň pod krajní mez a přinejhorším převyšují hlavní mez ani ne o 5 %.

Nejdůležitějším výsledkem experimentu je zjištění, že metoda selekce podstromů s nejmenším rozdílem dává nejhorší výsledky. To bylo i předpokládáno, neboť sdružování podstromů



Obrázek 5.12: Vliv výběru metody selekce podstromů u *P3D* pro SCGP

	Největší rozdíl	Nejmenší rozdíl	Náhodný výběr
Nejlepší řešení	(198; 3,98; 3,98)	(457; 3,98; 3,99)	(281; 3,97; 3,98)
Nejmenší řešení	(198; 3,98; 3,98)	(383; 3,98; 4,01)	(281; 3,97; 3,98)
Pod hlavní mezí	19	17	18
Pod krajní mezí	20	20	20

Tabulka 5.12: Vliv různého počtu generací u *P3D* pro SCGP

s málo uzly tvoří jedince s malým prostorem pro zmenšení a tudíž i evoluce nemá tolik prostoru, aby řešení zmenšila. Nicméně i tato nejhorší metoda je stále ve všech případech lepší než kompresní převod řešení získaného GSGP. Metoda největšího rozdílu se ukázala o něco lepší než náhodný výběr u všech měření. O časové složitosti algoritmu v závislosti na použité metodě nelze z experimentů vyvodit jednoznačný závěr. Je ale viditelné, že u úlohy *P3D* hraje roli velikost řešení – evaluace zde trvá déle než u ostatních úloh a menší řešení je tak i značně rychleji získané řešení.

5.3.4 Nejlepší nalezené nastavení SCGP

Jako nejlepší nastavení z testovaných parametrů se ukázala volba 100 generací a metody největšího rozdílu selekce podstromů. Volbou těchto hodnot u daných parametrů a zbylých hodnot z tabulky 5.3 je optimalizováno nastavení.

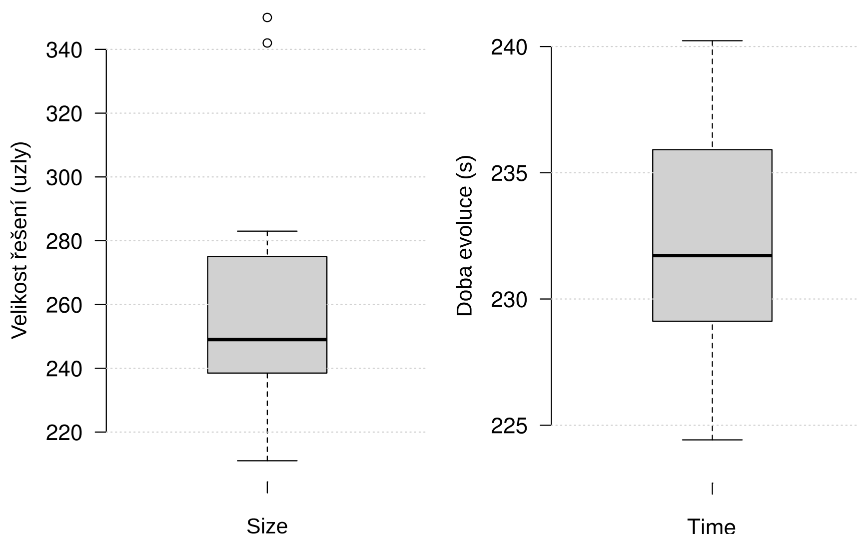
Výsledky úlohy *bioav* jsou zaneseny v grafu 5.13. Nejlepší řešení má hodnoty (224; 19,39; 26,69), což je redukce velikosti 97,73 % oproti řešení získanému kompresním převodem. Nejmenším řešením pak je (211; 19,44; 33,49), které je 5,8 % redukcí (13 uzlů) nejlepšího řešení a spadá pod krajní mez. Nejmenší redukce je dosaženo u řešení (350; 19,43; 28,43) a nejhorším řešením je (246; 19,41; 2,49e10), které trpí přetrénováním. Celkově se pak pod hlavní mezí nachází 4 řešení a pod krajní 18. Řešení pod hlavní mezí dokáží zobecňovat stejně, jako výsledek GSGP.

Nejrychleji nalezené řešení (224 sekund) je pod krajní mezí, stejně jako nejpomaleji nalezené řešení (240 sekund). Nejlepší řešení bylo nalezeno za 232 sekund.

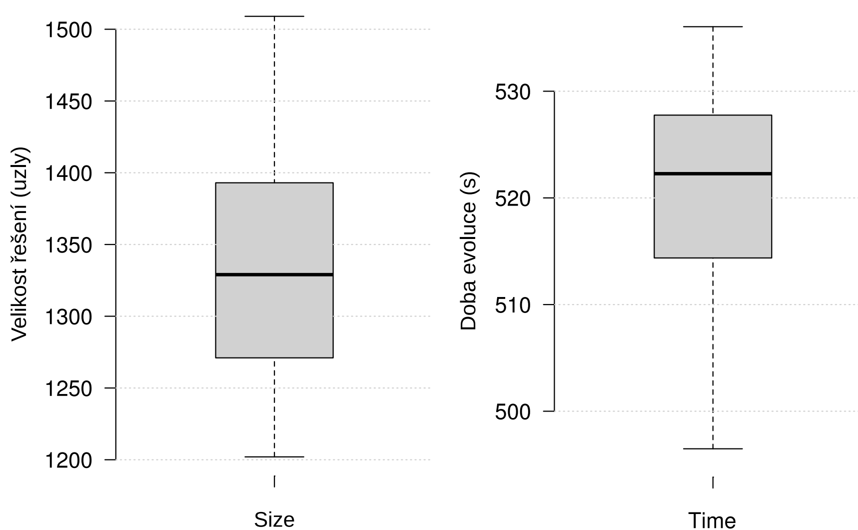
Druhá úloha, *PPB*, jejíž výsledky jsou zaneseny v grafu 5.14, má nejlepší řešení o hodnotách (1270; 4,27; 32,30). Toto řešení lehce trpí přetrénováním (hodnota testovací fitness je o 1,53 větší než hlavní mez), avšak je o 87,66 % menší než komprimované řešení GSGP.

Nejmenší řešení (1202; 4,25; 1,21e11) je sice o 5,35 % (68 uzlů) menší než nejlepší řešení, zato je to třetí nejhorší řešení s ohledem na obecnost. Největším řešením je (1509; 4,29; 33,15) a jakožto nejhorší řešení je (1339; 4,26; inf), které patří mezi jedno ze tří řešení, která nedokáží vůbec zobecňovat. Jak je vidět již na nejlepším řešení, pod hlavní mezí se pro tuto úlohu nenachází žádné řešení. Pod krajní mezí se nachází 11 řešení. Řešení získaná pro tuto úlohu nejsou schopna zobecňovat.

Nejrychleji nalezené řešení (496 sekund) je pod krajní mezí, nejpomaleji nalezené řešení (536 sekund) mělo více jak dvojnásobně větší hodnotu než je vyžadována pro krajní mez. Nejlepší řešení bylo nalezeno za 511 sekund.



Obrázek 5.13: Nejlepší nalezené nastavení parametrů pro SCGP u *bioav*

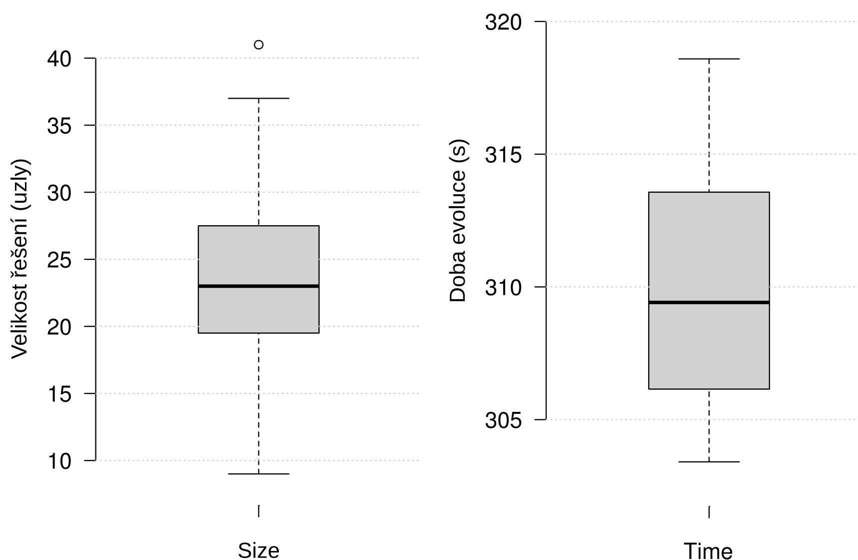


Obrázek 5.14: Nejlepší nalezené nastavení parametrů pro SCGP u *PPB*

Na grafu 5.15 jsou výsledky úlohy *Tox*. Nejlepší řešení má hodnoty (13; 1333,78; 1417,65) a je tak o 99,87 % menší než řešení po komprimovaném převodu. Řešení o hodnotách (9;

1331,82; 1508,89) je nejmenším řešením, které je o 30,77 % (4 uzly) menší než řešení nejlepší a hodnota testovací fitness je jen o něco málo větší, než hlavní mez. Největším řešením je (41; 1336,77; 1511,87) a nejhorším (16; 1336,94; 14190,69). Nejhorší řešení jsou dvě a obě obdobně špatná. Mimo ně je ostatních 18 řešení pod krajní mezí a 12 dokonce pod mezí hlavní. Řešení pod hlavní mezí se s přetrénováním vyrovnávají dobře.

Nejrychleji nalezená řešení (303 sekund) byla 3, a to jak řešení pod hlavní mezí, tak i řešení nejhorší. Řešení, které se vyvíjelo nejdéle (319 sekund), bylo přijato za kvalitní. Nejlepšímu řešení trvala evoluce 308 sekund.



Obrázek 5.15: Nejlepší nalezené nastavení parametrů pro SCGP u *Tox*

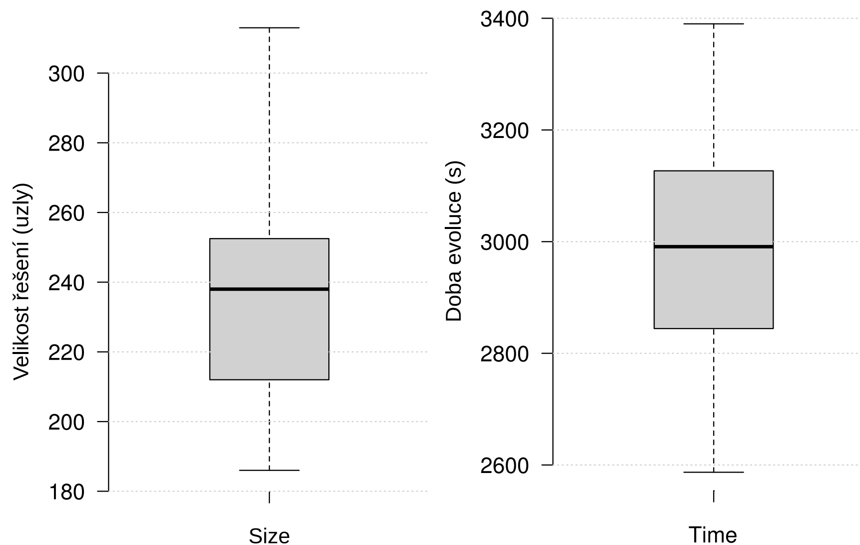
Výsledky úlohy *P3D* jsou zobrazeny na grafu 5.16. Řešení o hodnotách (186; 3,98; 3,98) bylo shledáno jakožto nejlepší nalezené a je o 95,34 % menší než řešení získané pomoci komprimovaného převodu. Toto řešení je taktéž nejmenším řešením. Největší řešení má hodnoty (313; 3,98; 3,98) a nejhorší (194; 3,98; 4,01). Nejhorší řešení je jen zanedbatelně horší s ohledem na hodnotu testovací fitness a tak všechna řešení jsou bezpečně pod krajní mezí. Pod hlavní mezí je 16 řešení a zbylá 4 jsou těsně nad ní. Řešení nejsou vůbec postižena přetrénováním.

Nejkratší čas potřebný k nalezení řešení je 2587 sekund, oproti tomu nejdelší je 3390 sekund. Obě řešení jsou pod hlavní mezí a jsou jedny z nejkratších. Nejlepší řešení bylo nalezeno za 2764 sekund.

Výsledky měření s optimalizovaným nastavením se dle předpokladů ukázaly jakožto nejlepší (lepší než z minulých experimentů). Výjimkou je pouze úloha *PPB*, kde se nepodařilo nalézt řešení, které je pod hlavní mezí. I přesto nebyla nejlepší řešení této úlohy daleko nad touto mezí a velikost řešení byla nejmenší ze zatím dosažených. Měření neukázala na vztah doby trvání běhu programu a kvalitu nebo velikost řešení.

5.3.5 Optimalizace výstupu SCGP pomocí CGP

Výsledek SCGP již nelze rozdělit na sémanticky spojené podstromy a proto připadá v úvahu ještě použití klasického CGP k optimalizaci výstupu SCGP. Parametry pro tyto experimenty jsou zaneseny v tabulce 5.4.

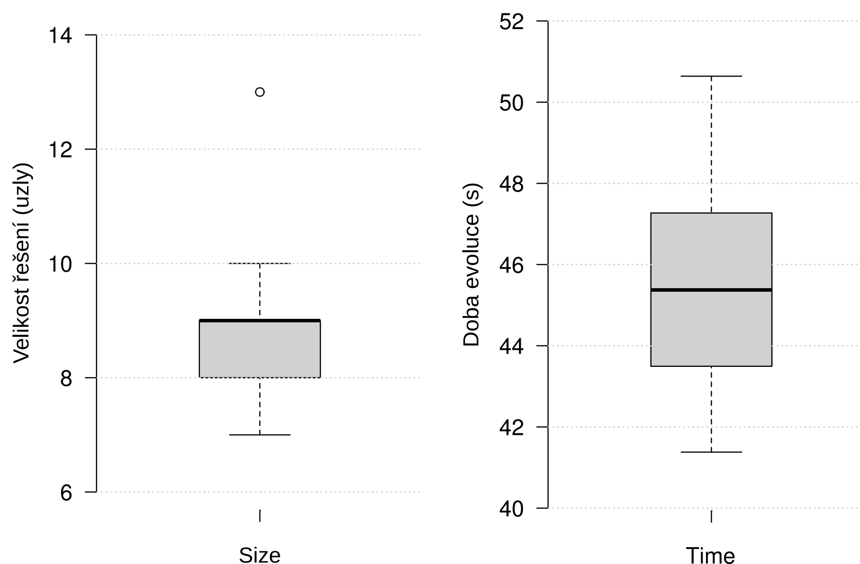


Obrázek 5.16: Nejlepší nalezené nastavení parametrů pro SCGP u *P3D*

Pro úlohy *bioav* , *PPB* a *P3D* nenastala jediná změna ve fenotypu. Aplikace CGP tak neměla na tyto výsledky vliv.

Jedinou výjimkou je úloha *Tox* (graf 5.17). CGP dokázalo zlepšit řešení až na (7; 1333,66; 1370,06) z původních (13; 1333,78; 1417,65). To je zlepšení o 6 uzlů bez ubrání na obecnosti řešení. Celkově se 18 řešení zvládlo dostat pod hlavní mez a všechna pak pod mez krajní. 10000 generací zvládlo u nejlepšího řešení proběhnout za 43 sekund.

Aplikace CGP přinesla jediné zlepšení a to v případě úlohy Tox. Zmenšení činí víc jak 45 %, ale řešení je i bez tohoto vylepšení již tak dost kompaktní.



Obrázek 5.17: Aplikace CGP na nejlepšího jedince pro úlohu *Tox*

5.3.6 Shrnutí

Řešení každé z testovaných úloh získaná pomocí GSGP se podařilo zmenšit pomocí SCGP. Následná aplikace CGP neukázala zlepšení s výjimkou úlohy *Tox*. První tři experimenty se týkaly nastavení parametrů SCGP a získání původního řešení pro hlavní testování.

Pro úlohu *bioav* mělo nejlepší řešení 224 uzlů a zmenšení oproti komprimovanému GSGP je 97,73 %. Takto nalezené řešení dokáže zobecňovat stejně dobře jako to původní.

Řešení úlohy *PPB* mělo při optimalizovaném nastavení malý problém s přetrénováním, zato však bylo zmenšeno původní řešení o 87,66 % na 1270 uzlů. Nejlepší řešení (bez přetrénování) bylo nalezeno při hledání nejlepší metody selekce podstromů. Toto řešení mělo velikost 1816 uzlů, tj. dosáhlo 82,35 % redukce. Nejhorší na řešeních získaných pomocí SCGP pro tuto úlohu je skutečnost, že nedokáží vůbec zobecňovat (ani ty bez přetrénování).

Úlohu *Tox* se podařilo zmenšit na pouhých 7 uzlů (redukce 99,93 %) pomocí SCGP a následnou aplikací CGP. Řešení nebylo ubráno na obecnosti i přes to, že z původních 626 vstupů jsou použity pouze 3. Problémem zde je vysoká hodnota fitness, okolo 1400, zatímco ostatní úlohy se s hodnotou fitness pohybují maximálně v desítkách. Tu poté není obtížné splnit ani nejhruběji vytvořenou funkcí.

Poslední úloha, *P3D*, má nejlepší řešení o velikosti 186, což je redukce 95,34 %. Řešení této úlohy obecně netrpí přetrénováním, nejspíše za to může vysoký počet záznamů datasetu, který je i příhodně rozložen.

Počty uzlů nejmenších řešení s ohledem na metodu a úlohu jsou zaneseny v tabulce 5.13.

Metoda:	GSGP	komprimované GSGP	SCGP	SCGP + CGP
bioav	22285	9863	224	224
PPB	22907	10291	1270	1270
Tox	21568	9934	13	7
P3D	5764	3992	186	186

Tabulka 5.13: Počty uzlů nejmenších řešení získaných testovanými metodami

Kapitola 6

Závěr

Prvním cílem práce bylo nastudovat geometrické sémantické genetické programování (GSGP), jeho předchůdce a podobné metody v genetickém programování. Druhým cílem byl návrh algoritmu převodu jedince z GSGP na jedince v kartézském genetickém programování (CGP). Jedná se o převod několika syntaktických stromů a struktur, které je spojují, na jeden chromozom kódující propojené uzly ve dvojrozměrné mřížce. Byly představeny dva algoritmy lišící se v přístupu ke znovupoužití stromů. První, který se neohlíží na to, zda byl již strom použit a vytvoří jeho obdobu v chromozomu znovu. Druhý se při znovupoužití stromu pouze odkazuje na příslušné místo v chromozomu.

Třetí a hlavní cíl práce byl návrh a otestování algoritmu, který dokáže řešení převedené do CGP velikostně optimalizovat – vzniklo tak podstromové CGP (SCGP). Jako vstup je použito řešení GSGP bez křížení převedené na instance CGP pomocí převodu bez znovupoužití stromů. Algoritmus pracuje se sémanticky spojenými částmi jedince, které postupně upravuje a spojuje. Testy ukázaly, že tato metoda dokáže jedince úspěšně zmenšit a ve 3 úlohách ze 4 také tato řešení nevykazují přetrénování. Toto zmenšení činí více jak 95 %. V případě poslední úlohy *PPB* byla dosažena redukce přes 87 %, ale řešení nedokáže zobecňovat.

Jakožto navazující práci navrhuji otestování metody SCGP na nové formě GSGP s lokálním prohledáváním představené v článku [3] z roku 2015. Jedná se zde o upravenou formu mutace, díky které je hodnota trénovací fitness menší na úkor hodnoty testovací fitness (která u SCGP ale stejně není nijak korigována). Algoritmus SCGP zůstává stále stejný a je pouze potřeba pozměnit implementaci, aby pracovala s novým typem mutace.

Literatura

[1] Archetti, F.; Lanzeni, S.; Messina, E.; aj.: Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, ročník 8, č. 4, Dec 2007: s. 413–432, ISSN 1573-7632, doi:10.1007/s10710-007-9040-z.
URL https://doi.org/10.1007/s10710-007-9040-z

[2] Castelli, M.; Silva, S.; Vanneschi, L.: A C++ framework for geometric semantic genetic programming. *Genetic Programming and Evolvable Machines*, ročník 16, č. 1, Mar 2015: s. 73–81, ISSN 1573-7632, doi:10.1007/s10710-014-9218-0.
URL https://doi.org/10.1007/s10710-014-9218-0

[3] Castelli, M.; Trujillo, L.; Vanneschi, L.; aj.: Geometric Semantic Genetic Programming with Local Search. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, New York, NY, USA: ACM, 2015, ISBN 978-1-4503-3472-3, s. 999–1006, doi:10.1145/2739480.2754795.
URL http://doi.acm.org/10.1145/2739480.2754795

[4] Goldman, B. W.; Punch, W. F.: Reducing Wasted Evaluations in Cartesian Genetic Programming. In *Genetic Programming*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, s. 61–72, ISBN 978-3-642-37206-3.

[5] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge : Bradford Book, London : MIT Press, 1992, ISBN 0-262-11170-5.

[6] Miller, J. F.: Cartesian Genetic Programming. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 17–34, ISBN 978-3-642-17309-7.

[7] Miller, J. F.: Introduction to Evolutionary Computation and Genetic Programming. In *Cartesian Genetic Programming*, Springer Berlin Heidelberg, 2011, s. 1–16, ISBN 978-3-642-17309-7.

[8] Nguyen, Q.; University College, D. S. o. C. S. . I.: *Examining Semantic Diversity and Semantic Locality of Operators in Genetic Programming*. Ph.D. Thesis, University College Dublin, 2011.
URL https://books.google.cz/books?id=45MBkAEACAAJ

[9] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Praha : Academia, 2009, 328 s., ISBN 978-80-200-1729-1.

[10] Vanneschi, L.; Castelli, M.; Silva, S.: A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, ročník 15, č. 2, Jun 2014: s. 195–214, ISSN 1573-7632, doi:10.1007/s10710-013-9210-0.
URL https://doi.org/10.1007/s10710-013-9210-0