VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOUTĚŽIVÁ KOEVOLUCE V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

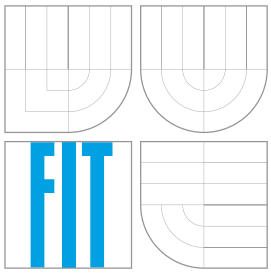
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

BARBORA SKŘIVÁNKOVÁ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOUTĚŽIVÁ KOEVOLUCE V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

COMPETITIVE COEVOLUTION IN CARTESIAN GENETIC PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — BARBORA SKŘIVÁNKOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAELA ŠIKULOVÁ
SUPERVISOR

BRNO 2014

Abstrakt

Symbolická regrese je metoda hledání předpisů funkcí, které co nejpřesněji prochází danými body v rovině nebo prostoru. V této práci je řešena symbolická regrese s využitím kartézského genetického programování a soutěživé koevoluce. Tato úloha byla již dříve řešena pomocí kartézského genetického programování a koevoluce prediktorů fitness. V této práci je zkoumáno, zda-li jednodušší soutěživá koevoluce dokáže dosáhnout obdobných výsledků jako koevoluce prediktorů fitness. Symbolická regrese je v této práci testována na pěti různě složitých úlohách. Při testování se ukázalo, že při řešení jednodušších úloh dosahuje soutěživá koevoluce oproti klasickému kartézskému genetickému programování výrazně vyššího zrychlení než koevoluce prediktorů fitness. Složitější úlohy, ve kterých koevoluce prediktorů fitness obstála stejně dobře jako v jednodušších, však soutěživá koevoluce vyřešit nedokázala.

Abstract

Symbolic regression is a function formula search approach dealing with isolated points of the function in plane or space. In this thesis, the symbolic regression is performed by Cartesian Genetic Programming and Competitive Coevolution. This task has already been resolved by Cartesian Genetic Programming using Coevolution of Fitness Predictors. This thesis is concerned with comparison of Coevolution of Fitness Predictors with simpler Competitive Coevolution approach in terms of approach effort. Symbolic regression has been tested on five functions with different complexity. It has been shown, that Competitive Coevolution accelerates the symbolic regression task on plainer functions in comparison with Coevolution of Fitness Predictors. However, Competitive Coevolution is not able to solve more complex functions in which Coevolution of Fitness Predictors succeeded.

Klíčová slova

Symbolická regrese, evoluční algoritmy, kartézské genetické programování, koevoluce.

Keywords

Symbolic regression, evolutionary algorithms, cartesian genetic programming, coevolution.

Citace

Barbora Skřivánková: Soutěživá koevoluce v kartézském
genetickém programování, bakalářská práce, Brno, FIT VUT v Brně, 2014

Soutěživá koevoluce v kartézském genetickém programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Michaely Šikulové.

.....
Barbora Skřivánková
31. července 2014

Poděkování

Na tomto místě bych ráda poděkovala svojí vedoucí Ing. Michaele Šikulové za precizní konzultace plné cenných rad pro vypracování této práce a v neposlední řadě také za ochotu pravidelně konzultovat i za ztížených podmínek způsobených studijním pobytem 2 000 km od univerzity.

Obsah

1 Úvod **3**

2 Teoretický základ práce **4**
2.1 Evoluční algoritmy . . . 4
2.1.1 Terminologie . . . 4
2.1.2 Princip evolučního algoritmu . . . 5
2.1.3 Genetický algoritmus . . . 7
2.1.4 Genetické programování . . . 8
2.2 Kartézské genetické programování . . . 9
2.2.1 Definice CGP . . . 9
2.2.2 Genotyp-fenotyp mapování . . . 12
2.3 Koevoluce . . . 12
2.3.1 Problémy řešené s pomocí koevoluce . . . 12
2.3.2 Princip koevoluce pro problémy založené na testech . . . 13
2.3.3 Princip koevoluce prediktorů fitness . . . 15
2.4 Symbolická regrese . . . 15

3 Návrh **16**
3.1 Evoluce kandidátních řešení . . . 16
3.1.1 Kandidátní řešení . . . 16
3.1.2 Generování počáteční populace . . . 17
3.1.3 Hledání aktivních uzlů . . . 18
3.1.4 Spuštění kandidátního programu nad trénovacím vektorem . . . 18
3.1.5 Vyhodnocení fitness . . . 18
3.1.6 Výběr rodičů . . . 19
3.1.7 Vytváření nové generace . . . 19
3.2 Evoluce podmnožin množiny všech trénovacích vektorů . . . 20
3.2.1 Test . . . 20
3.2.2 Archivy . . . 20
3.2.3 Vyhodnocení fitness . . . 21
3.2.4 Výběr rodičů . . . 21
3.2.5 Vytváření nové generace . . . 21

4 Implementace **23**
4.1 Paralelizace . . . 23
4.2 Generování pseudonáhodných čísel . . . 24
4.3 Formát vstupních souborů . . . 24
4.3.1 Soubor s povolenými funkcemi . . . 24

4.3.2 Soubor s trénovacími vektory . . . 25
4.4 Načítání vstupu . . . 25
4.5 Záznamy statistických dat . . . 25
4.6 Výpis řetězce řešení . . . 26
4.7 Kompilace a spuštění . . . 26

5 Experimentální vyhodnocení **27**
5.1 Nastavení evoluce kandidátních řešení . . . 27
5.2 Nastavení evoluce testů . . . 29
5.3 Skripty pro testování a vyhodnocování . . . 29
5.4 Posuzovaná kritéria a výsledky . . . 31
5.4.1 Funkce f_1 . . . 31
5.4.2 Funkce f_2 . . . 31
5.4.3 Funkce f_3 . . . 32
5.4.4 Funkce f_4 . . . 33
5.4.5 Funkce f_5 . . . 33
5.5 Shrnutí výsledků . . . 34

6 Závěr **35**

A Obsah CD **37**

B Grafy s rozšířenými výsledky testů **38**
B.1 Grafy k výběru parametrů koevoluce . . . 38
B.2 Grafy k výsledkům testování pro jednotlivé funkce . . . 42
B.2.1 Funkce f_1 . . . 42
B.2.2 Funkce f_2 . . . 42
B.2.3 Funkce f_3 . . . 43
B.2.4 Funkce f_4 . . . 43
B.2.5 Funkce f_5 . . . 44

Kapitola 1

Úvod

V průběhu vývoje informačních technologií byly výpočetní nástroje schopny řešit složitější a složitější úlohy. Až do dnešní doby výkon hardwarových komponent stále roste a tento trend nejspíše bude nadále pokračovat. V takové situaci je třeba věnovat se algoritmům, které jsou schopny těchto pokroků využít a dokáží hledat řešení problémů ve větších a větších prohledávacích prostorech. Jedním z typů těchto algoritmů jsou evoluční algoritmy, kterým je věnována tato práce.

Problematikou, která je za pomoci evolučních algoritmů v této práci řešena, je symbolická regrese, tedy hledání předpisů funkcí procházejících množinou izolovaných bodů. Kouzlo symbolické regrese je v tom, že hledání funkčních předpisů je prováděno bez rozsáhlých výpočtů. Lze ji tedy uplatnit i v laboratořích, kde není nutná znalost složitých výpočetních metod. Prostor všech možných řešení je systematicky prohledáván, dokud není nalezeno dostačující řešení. Způsoby optimalizace prohledávání prostoru řešení jsou v současné době předmětem výzkumu a tato práce je věnována právě jednomu z nich – kartézskému genetickému programování s využitím soutěživé koevoluce.

Cílem této práce je seznámit se s problematikou evolučních algoritmů, kartézského genetického programování a symbolické regrese. Na základě těchto znalostí navrhnout a implementovat program řešící symbolickou regresi jak pomocí standardního kartézského genetického programování, tak s využitím soutěživé koevoluce. Vytvořený program na sadě vybraných funkcí otestovat a výsledky porovnat s jinými, dříve vyzkoušenými, metodami koevoluce v kartézském genetickém programování.

Tato práce je členěna do šesti kapitol. Kapitola 2 se zabývá teoretickým základem evolučních algoritmů. V práci jsou potom k řešení problematiky symbolické regrese (2.4) využity algoritmy popsané v této kapitole – algoritmus kartézského genetického programování (2.2), genetický algoritmus (2.1.3) a koevoluční mechanismy (2.3).

Kapitola 3 je podrobně věnována návrhu praktické části této práce – programu řešícímu symbolickou regresi. Je rozdělena do dvou částí – 3.1, která se věnuje algoritmům vybraným pro jednotlivé fáze kartézského genetického programování a 3.2, která se věnuje výběru algoritmů pro jednotlivé fáze genetického algoritmu a koevoluce. Na tuto kapitolu navazuje kapitola 4, která je věnována implementaci. Je zde popsáno řešení některých složitějších otázek v implementaci zadaného programu.

Kapitola 5 dokumentuje výběr co nejvýhodnějších parametrů pro obě varianty programu. Dále popisuje způsob testování, jeho výsledky a nakonec je zde provedeno srovnání výsledků soutěživé koevoluce s výsledky koevoluce prediktorů fitness.

Závěrečná kapitola 6 obsahuje shrnutí dosažených výsledků a možné směry dalšího vývoje práce.

Kapitola 2

Teoretický základ práce

Mezi evoluční algoritmy se řadí mnoho různých algoritmů, z nichž jsou v této práci použity dva: genetický algoritmus a kartézské genetické programování. V této kapitole jsou tyto dva algoritmy uvedeny do širšího kontextu tříd evolučních algoritmů a jejich obecných vlastností.

V části 2.1 jsou postupně představeny základní vlastnosti evolučních algoritmů, jejich jednotlivé fáze a rozdíly různých přístupů k nim. V části 2.1.3 je potom představen genetický algoritmus tak, jak je použit v této práci. V části 2.2 je představeno kartézské genetické programování.

Část 2.3 představuje mechanismy pro řešení evolučních algoritmů s více populacemi pomocí koevoluce. Následuje část 2.4, která uvádí problematiku symbolické regrese tak, jak je řešena v této práci.

2.1 Evoluční algoritmy

Pod pojmem evoluční algoritmy se skrývá celá skupina prohledávacích algoritmů, které jsou inspirovány Darwinovou teorií evoluce a neodarwinismem. Darwinova evoluční teorie vysvětluje adaptivní změny vlastností popisovaných druhů takzvanou přirozenou selekcí. Dle této teorie přežívají (a dále se rozmnožují) pouze jedinci s nejlepšími vlastnostmi vzhledem k aktuálním okolním podmínkám. V průběhu generací má tento proces za následek postupné změny vlastností jedinců a zlepšování jejich schopnosti přežití v daném prostředí.

Analogický princip jako v Darwinově evoluční teorii je využíván i v evolučních algoritmech ve výpočetní technice. Celý algoritmus probíhá nad generací kandidátních řešení a v každé generaci jsou všechna kandidátní řešení ohodnocena fitness funkcí. Jako rodič pro další generaci je vybrán jedinec s nejlepší hodnotou fitness.

Evoluční algoritmy se od tradičních prohledávacích algoritmů odlišují tím, že operují nad celou množinou (populací) kandidátních řešení. Klasické prohledávací algoritmy se oproti tomu zabývají jedním řešením. Evoluční algoritmy se často používají pro řešení problémů, jejichž prohledávací prostor je velmi rozsáhlý, případně pro řešení dynamických problémů, u kterých se v průběhu výpočtu mění požadavky na řešení [2, 10].

2.1.1 Terminologie

Stejně jako většina specifických oborů, tak i evoluční algoritmy mají svoji zavedenou terminologii. Terminologie se v různé literatuře liší, pro tuto práci budou využity pojmy defi-

nované v této kapitole a potom dále v textu. Terminologie pro tuto práci volně vychází z literatury ([1, 11]).

Gen – základní stavební jednotka chromozomu. V biologické evoluci jsou pomocí genů kódovány proteiny. V evolučních algoritmech jeho hodnota patří do abecedy specifické pro danou úlohu (binární čísla, celá čísla apod.). Gen může reprezentovat např. hranu nebo uzel v grafu, logické členy, propojení apod.

Chromozom – obvykle lineární pole genů, v některých typech úloh může mít proměnnou délku.

Genotyp – termín používaný v evolučních algoritmech pro kódovaný tvar řešení pomocí jednoho nebo více chromozomů.

Fenotyp – kandidátní řešení problému, které je (obvykle deterministicky) sestavováno podle genotypu. Mapování genotypů na fenotypy a naopak je v jednotlivých evolučních algoritmech rozdílné. Některé evoluční algoritmy dokonce mezi genotypem a fenotypem vůbec nerozlišují.

Kandidátní řešení – jedinec se všemi svými vlastnostmi a projevy.

Populace – množina kandidátních řešení, nad kterou pracuje evoluční algoritmus. Její velikost (počet jedinců v populaci) může značně ovlivnit efektivitu celého výpočtu.

Fitness funkce – matematické vyjádření kvality kandidátního řešení vzhledem k hledaným vlastnostem. Jedná se o funkci, která přiřazuje každému jedinci hodnotu fitness, na jejímž základě se dá porovnat kvalita dvou různých genotypů.

2.1.2 Princip evolučního algoritmu

Na obrázku 2.1 je znázorněn princip evolučního algoritmu, jehož jednotlivé fáze budou popsány dále.

Vytvoření počáteční populace

Na počátku algoritmu je vytvořena počáteční populace. K vytváření počáteční populace existuje více známých přístupů závisejících na povaze řešené úlohy. Pokud je cílem evolučního vývoje optimalizace již existujícího řešení problému, v počáteční populaci se mohou objevit jedinci reprezentující různá, již existující, řešení. Pokud je však hledáno zcela nové řešení zatím nevyřešených problémů, počáteční populace bývá generována náhodně [11].

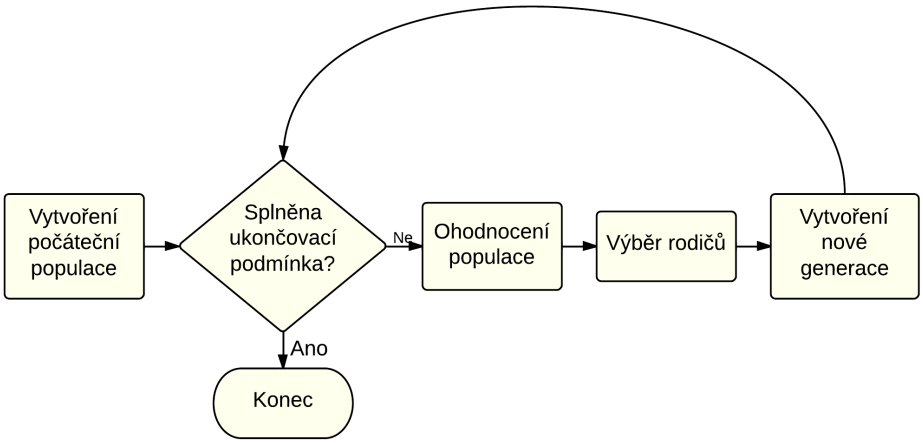
Ohodnocení populace

V dalším kroku jsou jedinci v populaci ohodnoceni, což znamená, že každému je přiřazena hodnota fitness. Hodnota fitness může být vyjádřena různými způsoby, jako jsou:

Hrubá fitness – přirozené numerické vyjádření fitness, kdy vyšší číslo znamená lepší fitness.

Standardizovaná fitness – hrubá fitness transformovaná tak, že žádanější jsou nižší hodnoty. Řešení, které řeší přesně zadanou úlohu má tedy fitness hodnotu 0.

Normalizovaná fitness – leží v intervalu $\langle 0, 1 \rangle$. Vzniká podílem hrubé hodnoty fitness zkoumaného jedince a součtu hrubých hodnot fitness všech jedinců.



Obrázek 2.1: Evoluční algoritmus.

Výběr rodičů

Na základě ohodnocení jednotlivých jedinců jsou dále vybíráni ti nejvhodnější pro reprodukci. Tomuto výběru se říká selekce a lze k ní použít některý z následujících tří selekčních mechanismů:

Deterministická selekce – nejjednodušší způsob selekce. Jde o případ, kdy je vždy vybráno n jedinců s nejvyšší fitness v generaci.

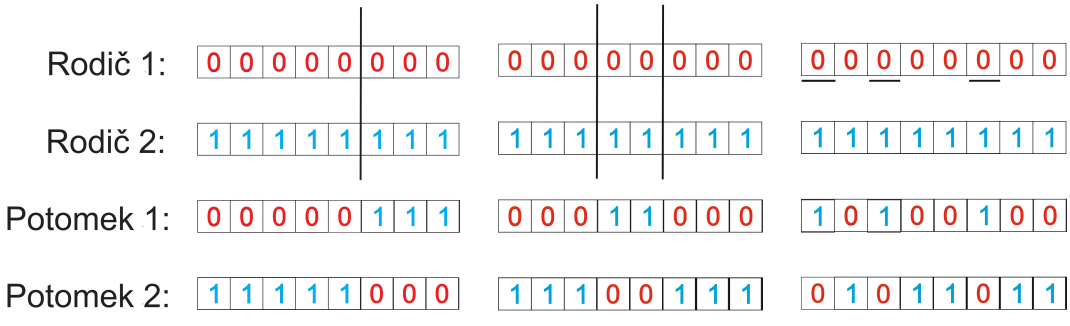
Turnajová selekce – selekce připomínající turnajová utkání. Z generace je náhodně vybráno n jedinců účastnících se turnaje. Ti jsou potom po dvojicích porovnáváni, vítěz vždy postupuje do dalšího kola, dokud není získán vítěz turnaje. Algoritmus je opakován tolikrát, kolik rodičů je třeba pro další generaci.

Proporcionální selekce – algoritmus selekce, ve kterém je interval $\langle 0, 1\rangle$ rozdělen na n podintervalů. Každý podinterval patří jednomu jedinci a má velikost odpovídající jeho normalizované fitness. Pomocí rulety (vygenerování náhodného čísla v intervalu $\langle 0, 1\rangle$) je potom vybrán jedinec, do jehož intervalu náleží vygenerovaná hodnota. Tento jedinec se stane rodičem. Je zřejmé, ze jedinci s vyšší hodnotou fitness mají vyšší šanci na výběr. Pokud má ale některý jedinec fitness příliš vysokou, hrozí, že dojde k degeneraci populace.

Vytvoření nové generace

V dalším kroku jsou z vybraných rodičů vytvořeni potomci. K modifikaci rodičů na potomky existují dva základní přístupy – mutace a křížení.

Křížení – křížení probíhá na principu využití části genů jednoho rodiče a části genů druhého rodiče. Křížení se dělí na jednobodové, vícebodové a uniformní (viz obrázek 2.2). Při jednobodovém křížení je náhodně vygenerován bod křížení a poté dojde k prohození genů nacházejících se za bodem křížení. U vícebodového křížení se vygeneruje více křížicích bodů a následně se prohodí každý druhý podřetězec ohraničený



(a) jednobodové křížení (b) dvoubodové křížení (c) uniformní křížení

Obrázek 2.2: Křížení.

body křížení. Při využití uniformního křížení je pro každý gen zvlášť rozhodováno, zda-li dojde k jeho prohození nebo ne.

Mutace – mutace je náhodná změna genů chromozomu. V klasických evolučních algoritmech se využívá spíše jako doplňková změna pro dosažení větší rozmanitosti v generaci než jako hlavní operátor tvorby potomků. Při mutaci je provedena náhodná změna náhodného genu jedince.

Ukončovací podmínka

Ukončovací podmínkou celého výpočtu bývá buďto dosažení cílených vlastností kandidátního řešení nebo provedení určitého počtu generací. Cílené vlastnosti kandidátních řešení jsou obvykle vyjádřeny určitou hodnotou fitness. Maximální počet generací je stanoven staticky a odvíjí se od náročnosti řešeného problému.

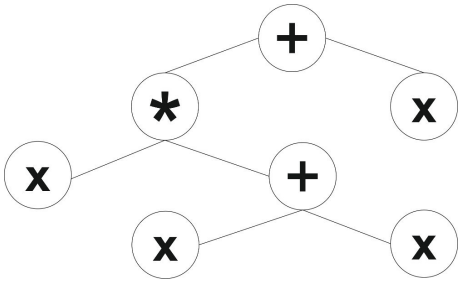
2.1.3 Genetický algoritmus

Jak uvádí [1], původní pohled na evoluční algoritmy se soustředí pouze na Hollandovy genetické algoritmy (představené v [4] v roce 1975), Kozovo genetické programování (představeno v roce 1992 v [6]), Fogelovo evoluční programování (představeno v roce 1966) a evoluční strategie podle Bienerta, Rechenberga a Schwefela. Tato práce je věnována genetickým algoritmům a genetickému programování, které je dále specializováno na kartézské genetické programování.

Genetický algoritmus (dále *GA*) je principiálně velmi podobný evolučnímu algoritmu ukázanému na obrázku 2.1. Generování počáteční populace je provedeno náhodně, každý fenotyp je zakódován řetězcem, který má předem danou délku. Rodiče vybraní pomocí selekčních algoritmů jsou kříženi, případně mutováni a z jejich potomků je vytvořena další generace. Při tvorbě nové generace existují dvě možnosti:

Generační varianta – celá další generace je vytvořena z potomků předchozí generace. Rodiče, ani žádní jiní členové předchozí generace nejsou členy generace následující.

Překrývání populací – nová populace je vytvořena ze směsi nově vygenerovaných jedinců a jedinců z předcházející generace.



Obrázek 2.3: Reprezentace programu ve formě binárního stromu.

Používané operátory

V genetických algoritmech jsou využívány jak operátory křížení, tak operátory mutace. Vzhledem k tomu, že jedinci jsou často zakódováni jako binární řetězce, operace se provádějí nad binárními čísly. Mutace nad binárním řetězcem probíhá jako přepnutí bitu na opačnou hodnotu. Křížení potom jednoduše aplikuje na lineární řetězce mechanismus křížení definovaný výše obrázkem 2.2.

2.1.4 Genetické programování

Genetické programování je modifikovanou variantou genetického algoritmu. Cílem genetického programování je vyvinout program, který co nejlepším způsobem řeší zadaný problém. Je tedy tvořena sekvence příkazů, která je přímo proveditelným programem. Dále jsou popsány vlastnosti genetického programování.

Reprezentace jedinců

V genetickém programování jedinci reprezentují programy obvykle ve formě abstraktních syntaktických stromů. Příklad velmi jednoduchého jedince genetického programování je znázorněn na obrázku 2.3.

Množiny terminálů a funkcí

Ve stromu se mohou vyskytovat dva druhy uzlů – terminální uzel, který ukončuje větev, ve které se nachází, a uzel funkční, který se vyskytuje uvnitř struktury stromu.

Terminály – množina terminálů obsahuje všechny prvky, které se ve stromové struktuře vyskytují na pozici listu. Jde o vstupy do programu, konstanty a funkce bez argumentů. Jako konstanty lze používat buďto výčet definovaných konstant, nebo je možné při použití konstanty vygenerovat náhodné číslo.

Funkce – množina funkcí obsahuje funkce vhodné k řešení problému nebo obecné matematické funkce. Kvůli náročnosti výpočtu není vhodné zařazovat příliš náročné funkce. Aby nedocházelo k chybám za běhu, je třeba definovat výstupy funkce pro speciální případy, jako je například dělení nulou.

Generování počáteční populace

Do programu se náhodně vybírají terminály a funkce. Pro sestavení stromu se využívají tři metody: Grow (náhodné stromy nepravidelného tvaru s terminály i funkcemi na kterékoliv

pozici), Full (stromy maximální hloubky) a Ramped Half-and-Half (polovina stromů metodou grow a polovina stromů metodou full s různými hloubkami) [11].

Výpočet fitness

Při výpočtu hodnoty fitness je spouštěn kód kandidátního řešení nad referenčními vstupy. Fitness je potom vypočtena z rozdílu výstupu kandidátního programu oproti referenčnímu výstupu. Skupina referenčních vstupů a výstupů je nazývána *trénovací množinou*, pomocí které jsou jedinci hodnoceni během procesu evoluce. Po skončení evoluce je výsledné řešení ověřeno pomocí *testovací množiny*.

Používané operátory

Používají se běžné operátory křížení a mutace, lze je ale obohatit o další operátory umožňující vytváření podprogramů, vkládání modulů a podobně.

Křížení – křížení je základní operací genetického programování a funguje tak, že z každého rodiče je náhodně vybrán uzel a jeden jeho podstrom. Tyto podstromy jsou potom mezi rodiči prohozeny, čímž vznikají dva potomci (viz obrázek 2.4).

Mutace – mutace je doplňkovým operátorem a neprovádí se vždy. Provádí se tím způsobem, že v jedinci je náhodně vybrán uzel a jeden jeho podstrom je nahrazen novým, náhodně vygenerovaným, podstromem.

2.2 Kartézské genetické programování

Kartézské genetické programování (dále *CGP* – Cartesian Genetic Programming) je varianta genetického programování, která byla poprvé představena J. F. Millerem v roce 1999 [8]. V CGP je jedinec reprezentován pomocí orientovaných grafů, které jsou zakódovány jako dvourozměrná pole výpočetních uzlů. Tyto uzly se skládají z několika čísel, která určují odkud daný uzel získává svoje data a jakou operaci nad nimi provádí.

2.2.1 Definice CGP

Kartézský program je definován svými devíti parametry:

G – vlastní matice všech uzlů

n_i – počet primárních vstupů jedince

n_o – počet primárních výstupů jedince

n_n – arita výpočetního uzlu

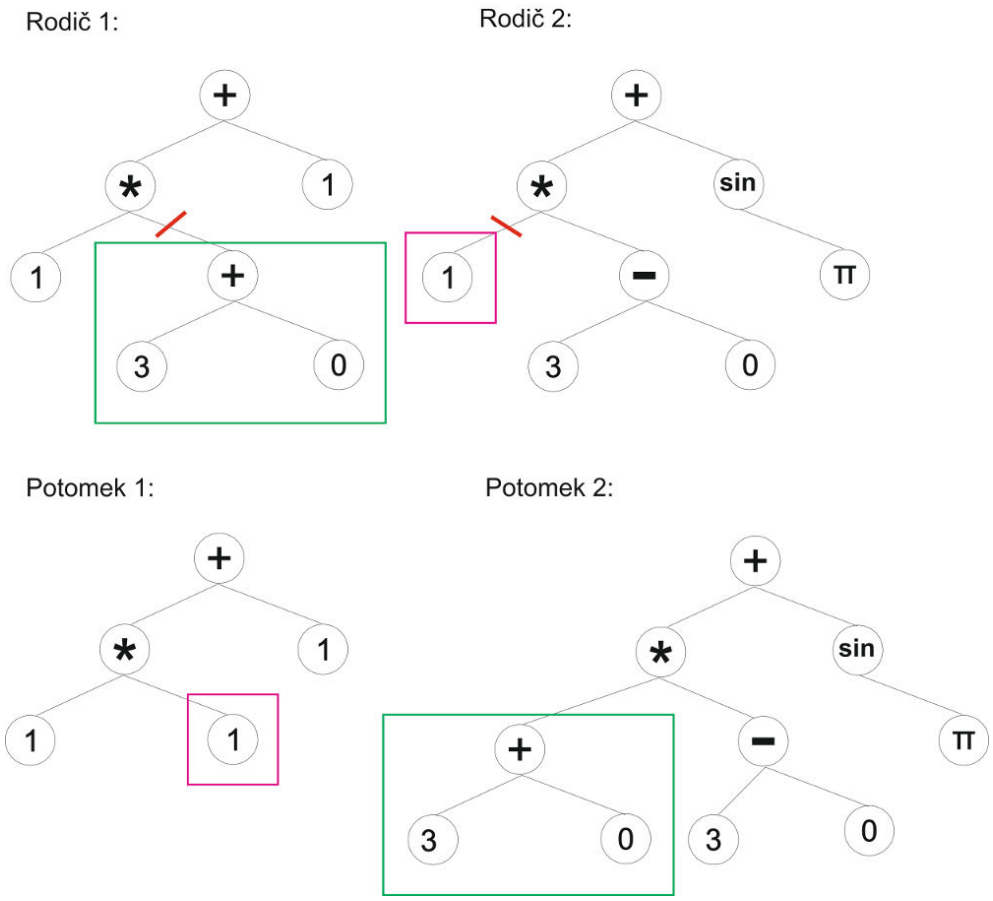
F – množina dostupných funkcí, bývá uživatelsky volena

n_f – počet dostupných funkcí

n_c – počet sloupců kartézského programu

n_r – počet řádků kartézského programu

l – l-back, číslo označující, z kolika bezprostředně předcházejících sloupců lze vybírat hodnoty pro vstup aktuálního uzlu.



Obrázek 2.4: Princip křížení dvou stromů.

Reprezentace jedinců

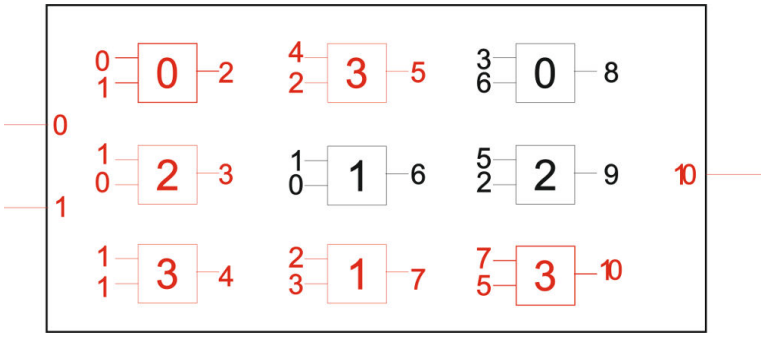
Kartézský program je reprezentován jako matice o n_r řádcích a n_c sloupcích obsahující jednotlivé výpočetní uzly. Ty jsou zakódovány jako textové řetězce. Na konec genotypu je přidáno n_o celých čísel označujících které uzly budou primárními výstupy kartézského programu. Každý výpočetní uzel se skládá z dvou typů genů, které kódují jednotlivé vlastnosti uzlu.

Funkcionální gen – gen, který značí, jakou operaci bude daný výpočetní uzel se svými vstupy provádět. V každém výpočetním uzlu se vyskytuje jen jeden.

Propojovací geny – ostatní geny uzlu (mimo funkcionální) jsou tzv. propojovací. Tyto geny určují, odkud daný uzel bude načítat své vstupy. Vstupem výpočetního uzlu může být buďto primární vstup kartézského programu, nebo výstup některého uzlu z předcházejících sloupců. Vstupem uzlu nemůže být výstup uzlu ze stejného sloupce, ani uzlu ze sloupců následujících. Při načítání výstupu uzlu z předchozího sloupce musí být také splněna podmínka pro l-back:

$$c_i - c_j \leq l \tag{2.1}$$

kde c_i je aktuální sloupec a c_j je požadovaný sloupec.



Obrázek 2.5: Náhodně vygenerované kandidátní řešení s následujícími parametry: $n_i = 2$, $n_o = 1$, $n_n = 2$, $n_f = 4$, $n_c = 3$, $n_r = 3$, l = 2. Množina dostupných funkcí F může být jakákoliv čtyřprvková množina, například definovaná výčtem: F = $\{AND_0, OR_1, PLUS_2, MINUS_3\}$.

Primární vstupy kartézského programu jsou postupně očíslovány, výstupy jednotlivých uzlů jsou potom také očíslovány. Pomocí těchto čísel je identifikováno, které hodnoty mají přijít na vstup jednotlivých uzlů. Názorná ukázka CGP programu je na obrázku 2.5, kde je zobrazena maticová reprezentace kadidátního řešení. Reprezentace řetězce tohoto chromozomu by vypadala následovně: 0 1 0 1 0 2 1 1 3 4 2 3 1 0 1 2 3 1 3 6 0 5 2 2 7 5 3 10. Uzly, které jsou součástí výpočtu – tedy jsou nepřímo připojeny na výstup, jsou znázorněny červeně. Tyto uzly jsou označovány jako aktivní uzly.

Generování počáteční populace

V kartézském genetickém programování je počáteční populace obvykle náhodně generovaná. Hodnoty jednotlivých genů kartézského programu však nemůžou být zcela náhodné, na každý typ genu jsou kladena určitá omezení. Hodnota funkcionálního genu f_i musí být platným indexem do tabulky funkcí, musí pro ni tedy platit vztah:

$$0 \leq f_i < n_f \tag{2.2}$$

Hodnota propojovacích genů má omezení poněkud složitější, je třeba zajistit, aby nebyla záporná a aby zároveň splňovala omezení pro l-back. Hodnoty propojovacích genů uzlu C_{ij} musí splňovat následující podmínky:

$$0 \leq C_{ij} \leq j * n_r + n_i \text{ (pro } j < l) \tag{2.3}$$

$$(j - l) * n_r + n_i \leq C_{ij} \leq j * n_r + n_i \text{ (pro } j \geq l) \tag{2.4}$$

kde i značí číslo řádku a j značí číslo sloupce.

Výpočet fitness

Výpočet fitness v CGP je prováděn stejným způsobem, jako v genetickém programování. Pro vývoj populace máme k dispozici množinu *trénovacích vektorů*, které obsahují vstupní hodnoty a očekávaný výstup ěpracují. Nad všemi těmito trénovacími vektory je každé kandidátní řešení spuštěno a dle rozdílu očekávaného výstupu a výstupu kandidátního programu je kandidátnímu řešení přiřazena fitness. Dva nejznámější způsoby provádění tohoto vyhodnocení jsou následující:

Střední kvadratická odchylka – po spuštění kandidátního řešení nad celou množinou trénovacích vektorů je vypočtena střední kvadratická odchylka získaných řešení od očekávaných řešení, která se stane hodnotou fitness.

Metoda skóre – po spuštění kandidátního řešení nad každým trénovacím vektorem je vyhodnoceno, zda-li odchylka od očekávaného řešení je menší, než stanovená tolerance. Pokud ano, je k hodnotě fitness přičtena jednička, jinak se neděje nic.

Používané operátory

V CGP je používáno výlučně mutačního operátoru, který náhodně mění jednotlivé geny programu (funkcionální nebo propojovací). Počet genů, které se v každé mutaci změní, bývá vyjádřen v procentech celkového počtu genů v programu a standardně se pohybuje okolo 5%. Po změně musí být opět dodrženy přípustné hodnoty jednotlivých genů. Operátor křížení se ve standardním CGP nevyužívá. V některých specifických problémech se však operátor křížení ukázal být velmi užitečným [9].

2.2.2 Genotyp-fenotyp mapování

Základní vlastností CGP je mapování genotypů na fenotypy. Zatímco klasické genetické programování mezi genotypy a fenotypy nerozlišuje, CGP ano. Genotypem je myšlen celý kartézský program se všemi svými vnitřními uzly, ať už jsou připojeny na výstup nebo ne. Fenotypem potom jsou pouze uzly, které jsou připojeny na výstup (aktivní uzly). Fenotyp tedy může mít velikost minimálně nula uzlů, pokud jsou všechny primární výstupy napojeny na primární vstupy kartézského programu. Nejvýše může mít stejný počet uzlů jako genotyp, pokud se v kartézském programu nevyskytuje ani jeden nekódový (neaktivní) uzel [3].

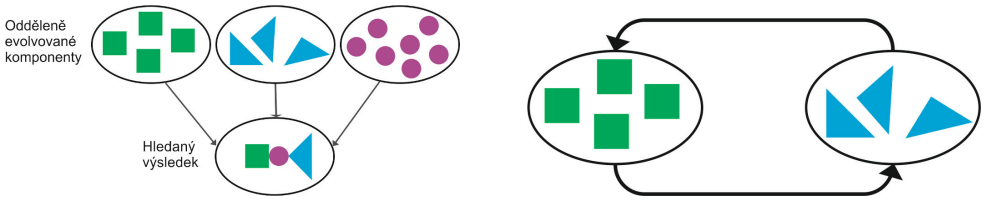
2.3 Koevoluce

Koevoluce je mechanismem, který je v biologii definován jako vzájemné genetické ovlivňování mezi dvěma a více různými druhy. Algoritmy využívající koevoluci se nazývají koevoluční algoritmy (dále *CoEA*). Díky koevolučním mechanismům je možné značně zkrátit jinak dlouho trvající proces ohodnocení všech kandidátních programů v genetickém programování [12]. Vyhodnocení fitness jedinců v populaci probíhá na základě interakce s jinými jedinci, což je hlavní rozdíl oproti klasickým evolučním algoritmům, kde je fitness odvozována pomocí předem definované fitness funkce.

Navzdory klasické definici z biologie, podle které koevoluce znamená vzájemné ovlivňování se mezi více živočišnými druhy, v CoEA je definována i koevoluce v rámci jednoho druhu. Při hodnocení fitness potom jedinci interagují s jedinci stejné populace. Na základě této přidané definice je v umělé inteligenci rozlišována koevoluce jedné populace a vícepopulační koevoluce. Problémem jednopopulační koevoluce se blíže zabývá D. Jansen [5]. Tato práce je věnována koevoluci více, konkrétně dvou, populací.

2.3.1 Problémy řešené s pomocí koevoluce

Koevoluce se nasazuje hlavně ve dvou typech problémů, kterými jsou problémy kompozičního charakteru a problémy založené na testech (viz obrázek 2.6). Oba přístupy jsou blíže popsány v této části.



(a) Problémy kompozičního charakteru (b) Problémy založené na testech

Obrázek 2.6: Typy koevoluce.

Problémy kompozičního charakteru

Při řešení problémů kompozičního charakteru je využíváno více populací, přičemž každá z nich obsahuje část kandidátního řešení. Pro získání celistvého řešení problému je třeba využít jedince ze všech populací, protože v každé populaci je vyvíjena pouze nějaká část řešení. Představit si to lze třeba jako vývoj optimálního stolního počítače. V jedné populaci se vyvíjí optimální zdroj, ve druhé populaci optimální základní deska, atd. Výsledné řešení problému hledání optimálního stolního počítače obsahuje prvky ze všech vyvíjených populací.

Problémy založené na testech

Základní rozdíl problémů založených na testech oproti kompozičním problémům je v účelu jednotlivých populací koevoluce. Problémy založené na testech využívají pouze jednu populaci obsahující kandidátní řešení, ostatní populace potom obsahují testy pro ohodnocení jedinců z populace kandidátních řešení. Typickým příkladem problému založeného na testech je symbolická regrese, blíže popsaná v kapitole 2.4.

2.3.2 Princip koevoluce pro problémy založené na testech

Koevoluce pro problémy založené na testech obvykle pracuje se dvěma populacemi – jednou populací kandidátních řešení a jednou populací testů. Zobrazení spolupráce obou populací je na obrázku 2.7.

Populace kandidátních řešení

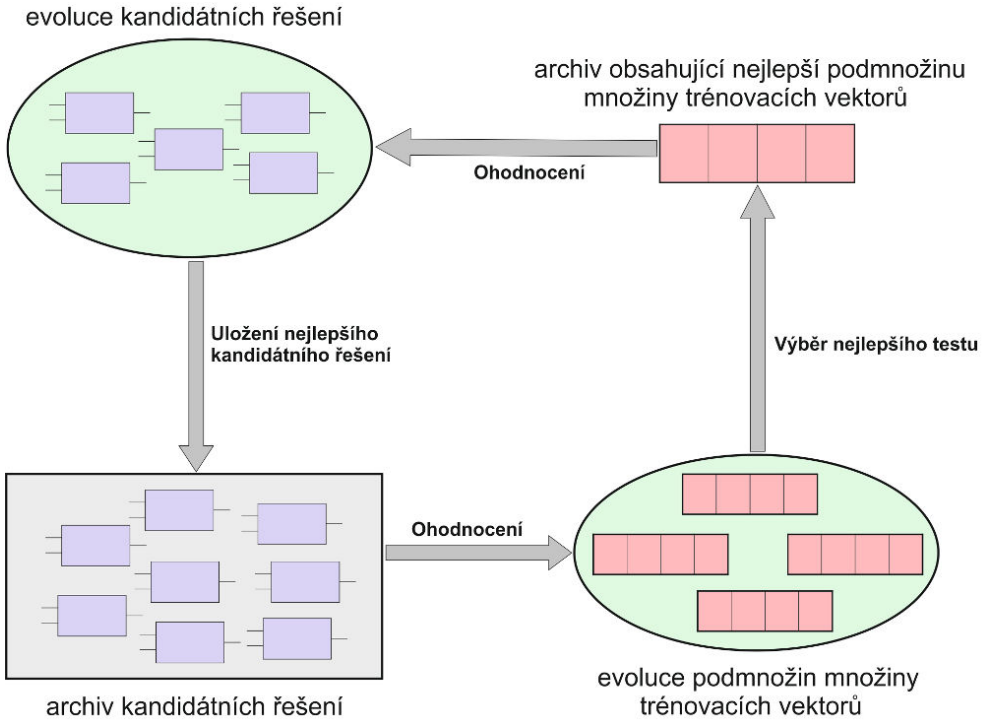
Populace kandidátních řešení obsahuje množinu jedinců, kteří jsou vyhodnocováni v aktuálním kroku evoluce. Tato množina se v CoEA nijak neliší od téže množiny v klasických EA. Obsahuje $1 + \lambda$ jedinců, ze kterých se v každé generaci vybírají rodiče a modifikují potomci.

Archiv kandidátních řešení

Množina kandidátních řešení je obnovována periodicky každých několik kroků, kdy je do ní přidán vždy nejvhodnější jedinec z populace kandidátních řešení. Tato množina je využívána k ohodnocení jedinců v evoluci testů.

Množina všech trénovacích vektorů

Množina všech trénovacích vektorů je množinou obsahující všechny trénovací vektory. Bývá obsáhlejší a proto je poněkud zdlouhavé využívat všechny její prvky při hodnocení kan-



Obrázek 2.7: Princip koevoluce.

didátních programů, což se řeší za pomoci evoluce testů.

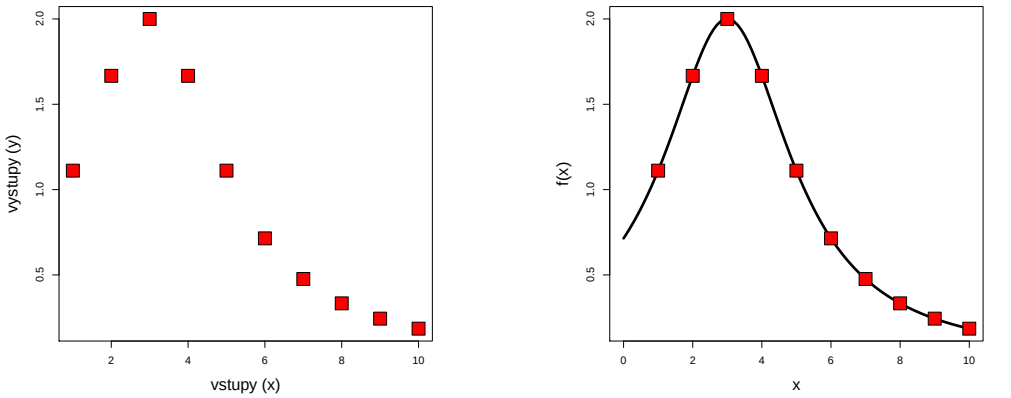
Populace testů

Test je malou podmnožinou množiny všech trénovacích vektorů, a tedy obsahuje pouze vybrané trénovací vektory. V této práci jsou porovnávány dva typy koevoluce:

- *Soutěživá koevoluce* – V populaci testů je hledána sada trénovacích vektorů, která odhalí nejvíce chyb v aktuálně vyvíjených kandidátních programech. Nejvhodnějším jedincem je tedy takový, na kterém množina kandidátních řešení v archivu dosahuje nejhorší fitness.
- *Koevoluce prediktorů fitness* – Koevoluce prediktorů fitness je složitějším typem koevoluce než soutěživá koevoluce. V populaci testů jsou vyvíjeny prediktory fitness (jiné typy testů než v soutěživé koevoluci), které jsou malými podmnožinami celé trénovací množiny. Fitness testu je určena na základě odchylky fitness určené pomocí množiny všech trénovacích vektorů a fitness určené pomocí testu pro dané kandidátní řešení. Nejvhodnějším jedincem je tedy takový, který dokáže co nejpřesněji určit fitness kandidátního řešení v porovnání s využitím množiny všech trénovacích vektorů.

Nejvhodnější test

Nejvhodnější podmnožina trénovacích vektorů vybraná z populace testů je po několik evolučních kroků používána k ohodnocování kandidátních řešení a jejich vývoji. Při každé změně této množiny se může změnit hodnota fitness jednotlivých kandidátních řešení.



(a) vstup (b) výstup: nalezená závislost y = $\frac{10}{5+(x-3)^2}$

Obrázek 2.8: Symbolická regrese.

2.3.3 Princip koevoluce prediktorů fitness

Koevoluce prediktorů fitness je druh koevoluce používaný v kartézském genetickém programování, je složitějším typem koevoluce než soutěživá koevoluce, která byla vybrána pro tuto práci. V druhé populaci jsou v koevoluci prediktorů fitness vyvíjeny prediktory fitness, které jsou menšími podmnožinami celé trénovací množiny. Prediktoru je vypočtena fitness a na jejím základě se predikuje fitness pro celou trénovací množinu [12].

2.4 Symbolická regrese

Symbolická regrese je přístupem k hledání funkčních předpisů, který umožňuje vyjádřit v symbolické formě vztah mezi závislými a nezávislými proměnnými. Úkolem symbolické regrese je zjistit, jakým způsobem jednotlivé nezávislé proměnné (vstupy) ovlivňují závislé proměnné (výstupy) a následně ověřit správnost a obecnost vytvořeného modelu. Je tedy hledána funkce, která s co nejmenší odchylkou prochází stanovenými trénovacími vektory. Příklad práce symbolické regrese je naznačen na obrázku 2.8.

V současné době je symbolická regrese velmi často řešena s pomocí automatizovaných systémů, ať už jde o evoluční algoritmy či jiný druh automatizovaných výpočtů. Uplatňuje se Kozovo genetické programování [6]. V této práci je symbolická regrese řešena pomocí CGP, které je vzhledem k náročnosti výpočtů optimalizováno pomocí koevoluce.

Kapitola 3

Návrh

Tato kapitola je věnována návrhu programu, který řeší symbolickou regresi s pomocí kartézského genetického programování a soutěživé koevoluce. Soutěživá koevoluce je jednodušším druhem koevoluce než koevoluce prediktorů fitness představená v části 2.3.3. Jak se ukázalo v [12], koevoluce prediktorů fitness dokázala vybrané úlohy symbolické regrese urychlit 2-5 krát. Cílem této práce je ověřit, zda-li je možné podobných výsledků dosáhnout i při použití jednoduššího přístupu – soutěživé koevoluce.

Koevoluční algoritmus zkoumaný v této práci využívá dvě souběžné evoluce:

- *Evoluci kandidátních řešení*, která jsou vyvíjena na principu CGP, jehož teoretické základy jsou uvedeny v části 2.2.
- *Evoluci podmnožin množiny všech trénovacích vektorů*, která probíhá na principu GA, teoreticky popsaném v části 2.1.3.

Mechanismy, které využívají tyto souběžné evoluce, jsou popsány dále v této kapitole. Koevoluční algoritmus je v této práci porovnáván s variantou, která koevoluci nevyužívá – symbolická regrese je řešena pomocí standardního CGP. Část 3.1 proto popisuje řešení symbolické regrese pomocí CGP pro koevoluční variantu i pro variantu, jež koevoluci nevyužívá.

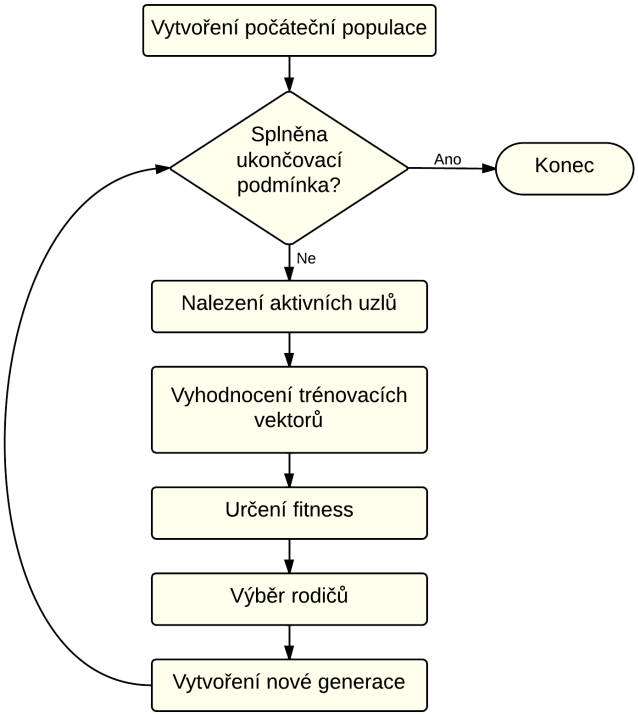
3.1 Evoluce kandidátních řešení

Kandidátní řešení jsou vyvíjena na principu CGP, jak je popsáno v části 2.2. Použitý algoritmus CGP je znázorněn na obrázku 3.1. Celý algoritmus probíhá nad populací, která je složena z dále popsaných jedinců (kandidátních řešení). Každý z kroků tohoto algoritmu lze provést několika různými způsoby, které jsou popsány a následně je mezi nimi vybráno v této kapitole.

3.1.1 Kandidátní řešení

Jedinec je reprezentován mřížkou výpočetních uzlů, které jsou vzájemně propojeny. Tato mřížka tvoří orientovaný acyklický graf, který vyjadřuje matematickou funkci – kandidátní řešení problému.

Každý výpočetní uzel má dva vstupy, nad nimiž vykonává požadovanou funkci a jeden výstup, jak je ukázáno na obrázku 3.2. Uzel na obrázku vykonává operaci sčítání nad vstupy x_1 a x_2. Výsledek je po provedení operace k dispozici na výstupu výpočetního uzlu. Každý uzel je zakódován třemi celými čísly, kdy první dvě značí propojení vstupů a poslední značí



Obrázek 3.1: Algoritmus CGP.

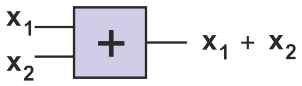
prováděnou funkci. Uzel na obrázku 3.2 by mohl být zakódován například sekvencí 1 0 3. Za předpokladu, že kandidátní řešení má jeden primární vstup by to znamenalo, že vstup x_1 je napojen na výstup prvního výpočetního uzlu, vstup x_2 je napojen na primární vstup kandidátního řešení a pro sčítání je použita konstanta 3.

Speciálním typem výpočetního uzlu je uzel realizující konstantu. Takový uzel nemá propojený vstup s žádným jiným uzlem v mřížce. Má aktivní pouze jeden vstup, který je napojen na některou konstantu z tabulky konstant. Tato načtená konstanta prochází na výstup výpočetního uzlu nezměněná. Vzhledem k charakteru testovacích úloh, které jsou popsány v kapitole experimenty, jsem zvolila následující množinu konstant: $\{0, 1, \pi, e\}$.

Společně s jedincem je třeba pro další použití uchovávat další informace. Jeho součástí tedy kromě tabulky uzlů a jejich propojení bude i hodnota výstupu a fitness pro aktuální trénovací vektor a indikátor aktivity u každého uzlu.

3.1.2 Generování počáteční populace

Vytvoření počáteční populace lze provádět dvěma způsoby závisejícími na typu zadané úlohy. Při hledání řešení zatím nevyřešeného problému se využívá náhodného generování počátečních genotypů, zatímco při optimalizaci již vyřešené úlohy se jako počáteční populace používají již existující řešení a jejich mutace. Vzhledem k tomu, že symbolická regrese jako taková spadá do kategorie první – hledání nových řešení zatím nevyřešených problémů, je v této práci použito náhodné generování počáteční populace.



Obrázek 3.2: Výpočetní uzel.

3.1.3 Hledání aktivních uzlů

Hledání aktivních uzlů je procesem označování uzlů, které jsou přímo i nepřímo připojeny na výstup a tedy jsou použity pro vlastní výpočet výstupní hodnoty programu. K hledání aktivních uzlů lze opět přistoupit dvěma způsoby. Pro označení aktivních uzlů lze využít buďto rekurzivní sestup stromem aktivních uzlů nebo zpětný průchod celým genotypem. Pro tuto práci byla z důvodu nižších paměťových nároků a jednoduchosti implementace vybrána metoda zpětného průchodu genotypem.

Zpětný průchod genotypem začíná v uzlu, který je připojen na primární výstup programu. Jeho vstupy jsou označeny jako aktivní. Poté je procházeno celým jedincem od posledního sloupce směrem k prvnímu a pokaždé, když je nalezen aktivní uzel, jeho vstupy jsou také označeny jako aktivní.

3.1.4 Spuštění kandidátního programu nad trénovacím vektorem

Souštění kandidátního programu se provádí pro každý vektor trénovací množiny. Skrze primární vstupy do programu vstoupí hodnoty, nad kterými jsou poté v odpovídajícím pořadí prováděny funkce jednotlivých aktivních uzlů, dokud se nedostanou k primárnímu výstupu. Na primárním výstupu se potom nachází výsledná hodnota kandidátního řešení pro aktuální trénovací vektor.

Při výpočtu dílčích výsledků jednotlivých výpočetních uzlů může dojít k výjimkám, pro které je třeba vložit speciální ošetření. Tato ošetření je třeba provést pro všechny funkce, které nejsou definovány na některé části definičního oboru – v případě této práce se jedná pouze o dělení, pro které je využita následující funkce.

$$\frac{x}{y} = \begin{cases} \frac{x}{y} & \text{pokud } y \in R - \{0\} \\ 1 & \text{pokud } y = 0 \end{cases} \tag{3.1}$$

3.1.5 Vyhodnocení fitness

Jakmile je vypočtena hodnota kandidátního řešení pro daný vstup, lze ji ohodnotit pomocí fitness. Pro hodnocení fitness byly zvažovány dvě metody – metoda skóre a střední kvadratické odchylky. Ve vztahu 3.2 pro metodu skóre a vztahu 3.3 pro metodu střední kvadratické odchylky jsou použity proměnné n – počet trénovacích vektorů, y_i – očekávaný výstup pro aktuální vektor a o_i – získaný výstup pro aktuální vektor:

$$fitness = \sum_{i=0}^{n} \begin{cases} 1 & \text{pokud } abs(y_i - o_i) < tolerance \\ 0 & \text{jinak} \end{cases} \tag{3.2}$$

$$fitness = \frac{\sum_{i=0}^{n} (y_i - o_i)^2}{n} \tag{3.3}$$

V praxi se ukázala jako vhodnější metoda skóre, která funguje na principu tzv. *hitu*. Uživatel stanoví toleranci, se kterou chce považovat výsledek za odpovídající očekávané

hodnotě. Pro každý trénovací vektor je potom vypočten rozdíl očekávaného a získaného výsledku. Pokud je menší než zadaná tolerance, je ke skóre jedince přičtena jednička. Nejlepší jedinec je takový, který dosáhl nejvyšší hodnoty fitness. Hodnota fitness je ponechána jako hrubá fitness, tudíž fitness jedince, který je schopen úplně vyřešit danou úlohu je rovna počtu trénovacích vektorů.

3.1.6 Výběr rodičů

Při výběru rodičů lze využít tři druhy selekce představené v části 2.2.1 – deterministickou, turnajovou a proporcionální selekci. Pro selekci rodičů v CGP byla vybrána jednoduchá deterministická selekce, kdy rodičem se stane vždy právě jeden nejlepší jedinec z předcházející generace. Tato možnost byla zvolena z toho důvodu, že v turnajové i proporcionální selekci hrozí ztráta nejlepšího jedince, což by pro velmi pomalu probíhající vývoj symbolické regrese mohlo být fatální.

Výběr nejlepšího jedince se provádí pomocí porovnání hodnot fitness všech jedinců populace. V případě shodné hodnoty fitness u dvou jedinců generace se uplatňují druhotná kritéria:

Jedinec s méně aktivními uzly – pro zajištění co nejjednoduššího vygenerovaného funkčního předpisu jsou upřednostňováni jedinci s méně aktivními uzly – tedy reprezentující jednodušší vztahy.

Jedinec, který nebyl rodičem – pro zachování diverzity populace je výhodnější jako rodiče zvolit jedince, který ještě rodičem nebyl. Ačkoliv má stejnou fitness, v jeho neaktivních uzlech se mohou objevit důležité neutrální mutace, které pozitivně ovlivní další vývoj.

3.1.7 Vytváření nové generace

V CGP se k vytváření potomků využívá výhradně operátor mutace. λ jedinců je tedy mutováno z jediného rodiče. V každém potomku je mutováno náhodně 1-8% genů, kdy genem je myšlena jedna hodnota uzlu (jeden ze vstupů nebo funkce). Výběr mutovaného genu se provádí na základě dvou náhod – nejprve je náhodně zvolen výpočetní uzel, ve kterém bude mutace provedena, poté je opět náhodně zvolen gen, který bude mutován. Mutaci lze provádět několika různými způsoby, závisejícími na mutovaném genu. Pro tyto dále popsané situace je třeba definovat dva pojmy – *funkční uzel* je uzel, který provádí některou z definovaných matematických operací, *konstantní uzel* je uzel generující konstantu.

Mutace vstupu funkčního uzlu – mutace jednoho ze vstupů uzlu, který realizuje některou matematickou funkci. Při této mutaci je náhodně generováno číslo odpovídající číslu výpočetního uzlu ležícího v tolika předcházejících sloupcích, kolik odpovídá parametru $l - back$, případně číslo indentifikující primární vstup. Toto číslo se stane novým vstupem mutovaného uzlu.

Mutace funkce funkčního uzlu – jestliže je mutován uzel provádějící některou matematickou funkci na uzel provádějící jinou matematickou funkci, nejsou třeba žádné speciální kontroly. Pokud je však mutován funkční uzel na konstantní uzel, je třeba provést další kontroly, které jsou popsány v mutaci konstantního uzlu.

Mutace konstantního uzlu – ať už je mutován konstantní uzel na funkční nebo funkční na konstantní, při těchto převodech je vždy třeba provést kontrolu a případnou korekci

prvního vstupu tohoto uzlu. Při mutaci na konstantní uzel musí první vstup spadat do rozsahu tabulky konstant. Při mutaci na funkční uzel musí první vstup odpovídat podmínkám pro vstup funkčního uzlu. Tato problematika je řešena tak, že první vstup je vždy přegenerován tak, aby odpovídal podmínkám.

Mutace primárního výstupu – při mutaci primárního výstupu je vždy náhodně vygenerováno číslo odpovídající výstupu některého z výpočetních uzlů v mřížce genotypu.

3.2 Evoluce podmnožin množiny všech trénovacích vektorů

Koevoluční varianta této práce sestává ze dvou různých evolucí – evoluce kandidátních řešení a evoluce podmnožin množiny trénovacích vektorů (dále označovaných pojmem *test*). Evoluce kandidátních řešení probíhá ve svém vlastním vlákně na principech popsaných v části 3.1. Pro evoluci testů je implementován jednoduchý genetický algoritmus, který probíhá v druhém vlákně.

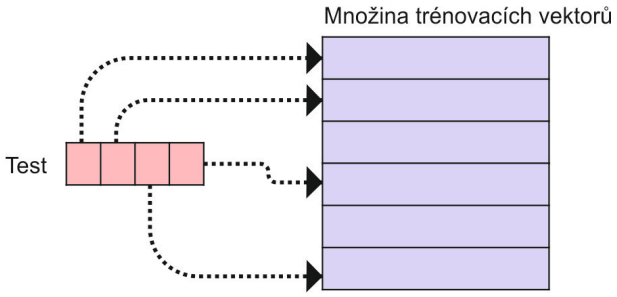
Práce koevolučního vlákna tkví v evoluci testů se snahou vytvořit takový test, který v aktuálních kandidátních řešeních odhalí co nejvíce chyb – tedy takovou podmnožinu množiny trénovacích vektorů, se kterou vybraná kandidátní řešení dosáhnou nejhorší fitness.

Paralelní běh dvou vláken lze provádět s různými metodami zpožďování. Dvě nejpoužívanější možnosti jsou zpoždění na základě doby výpočtu a umělé zpoždění pomocí synchronizačních prostředků. V této práci je koevoluční vlákno uměle zpožděno v poměru 1 generace koevoluce ke 100 generacím CGP.

Přínosem soutěživé koevoluce v kartézském genetickém programování je zrychlení procesu ohodnocování jednotlivých kandidátních řešení. Fitness každého kandidátního řešení je totiž posuzována pouze dle schopnosti reagovat na několik trénovacích vektorů v testu. Bez koevoluce je třeba vyhodnotit celou trénovací množinu, což je nepoměrně více trénovacích vektorů.

3.2.1 Test

Jedinec evoluce testů je sada trénovacích vektorů vybraná z celé množiny trénovacích vektorů pomocí genetického algoritmu (znázorněno na obrázku 3.3). Spolu s jedincem je třeba uchovávat aktuální fitness daného jedince.



Obrázek 3.3: Jedinec CGP.

3.2.2 Archivy

Pro úspěšnou práci CGP s koevolucí je třeba přidat prostředky pro komunikaci mezi evolucemi. V této práci jsou realizovány pomocí dvou archivů, jak je naznačeno na obrázku

2.7, archiv kandidátních řešení a archiv obsahující nejvhodnější test.

Archiv kandidátních řešení

Archiv kandidátních řešení je rozdělen na dvě poloviny, které se liší strategií obměňování. Jedna polovina archivu je obměňována vždy těsně před aktivací koevolučního vlákna (tedy každých 100 generací), kdy je do archivu vložen aktuální jedinec s nejvyšší fitness. Do druhé poloviny archivu je vkládán taktéž právě jeden nejlepší prvek populace, ale tehdy, když dojde ke změně fitness. V obou dvou částech archivu se novým jedincem nahrazuje nejstarší jedinec dané části archivu.

Archiv s nejvhodnějším testem

V každé generaci koevoluce je nalezen test, který má nejlepší schopnost odhalovat chyby v kandidátních řešeních a ten se pro další periodu stane testem používaným v evoluci kandidátních řešení.

3.2.3 Vyhodnocení fitness

V evoluci testů je hledán takový jedinec, který při interakci s jedinci z populace kandidátních řešení odhalí nejvíce chyb – tedy kandidátní řešení budou pro něj dosahovat co nejhorších fitness. Vztah pro výpočet fitness testu je uveden v rovnici 3.4, kde n je počet kandidátních řešení v archivu, g_{test} je hledaná fitness testu a f_i je fitness dosažená na daném testů i-tým kandidátním řešením. Na daném testu jsou spuštěna všechna kandidátní řešení v archivu, jejichž dosažené fitness jsou poté sečteny, čímž je získána fitness hodnota testu. Nejvhodnějším testem v generaci je takový test, jehož hodnota fitness je nejnižší. Tedy takový, na kterém dosáhli jedinci uloženi v archivu kandidátních řešení nejhorších výsledků.

$$g_{test} = \sum_{i=0}^{n} f_i(test) \tag{3.4}$$

3.2.4 Výběr rodičů

Výběr rodičů lze dělat několika různými způsoby, jak bylo popsáno v kapitole 2.2.1. Pro potřeby koevolučního algoritmu z nich byla vybrána turnajová selekce. V turnajové selekci sice hrozí, že rodičem se nestane nejkvalitnější jedinec populace, což ale nevadí, protože je využíváno překrývání generací, kde jsou nejlepší jedinci z předchozí generace zachováni do další.

Turnaj je prováděn jednoúrovňově. Provede se náhodný výběr dvou jedinců z generace, následně je porovnána jejich fitness a jedinec s lepší (tedy nižší) fitness se stává rodičem. Toto je opakováno tolikrát, kolik rodičů je pro novou generaci potřeba.

3.2.5 Vytváření nové generace

Nová generace sestává ze tří druhů jedinců – z několika nejlepších jedinců předchozí generace, z několika potomků rodičů vybraných z předchozí generace a z několika nově náhodně vygenerovaných jedinců. Poměr jednotlivých složek je předmětem experimentů, které jsou dokumentovány dále v této práci, v kapitole 5.

Tvorba potomků je prováděna pomocí vícebodového křížení, které je znázorněno výše na obrázku 2.2. Ze dvou rodičů jsou vytvořeni dva potomci, takže je třeba vybrat právě tolik rodičů, kolik je požadováno pro další generaci potomků.

Kapitola 4

Implementace

Praktická část práce – program řešící symbolickou regresi s pomocí soutěživé koevoluce a s pomocí standardního CGP je napsán v jazyce C/C++. Přístup k problému je ryze funkcionální, odpovídá tedy přístupu v jazyce C, nicméně pro možnost využití užitečných rozšíření v podobě některých datových typů a kontejnerů je implementace provedena v jazyce C++ dle standardu C++98. Pro překlad a sestavení programu je použit překladač G++.

Praktická část této práce sestává dvou programů – symbolické regrese s klasickým CGP a koevoluční varianty CGP. Tyto dvě varianty implementace však mají mnoho společného kódu, z důvodu konzistence jsem se tedy rozhodla obě varianty uchovávat v jednom zdrojovém textu. Části, ve kterých se kód klasického CGP liší od koevoluční varianty, jsou rozděleny pomocí direktiv preprocesoru, které se řídí existencí (nebo neexistencí) symbolické konstanty COEVOLUTION. Tímto způsobem lze z jednoho zdrojového kódu dvěma různými způsoby překladu získat dva různé programy.

Zdrojový kód je rozdělen do několika modulů, zastupujících jednotlivé fáze zpracování, jako jsou například vstupně-výstupní operace, tvorba počáteční populace, výpočet hodnot, evoluční krok nebo koevoluce. Funkce obsažené v jednotlivých modulech jsou volány z hlavního modulu *cocgp.cpp*.

Tato kapitola pojednává o problémech týkajících se přímo praktické části této práce. Nejde o kompletní postup celé implementace, nýbrž o několik implementačních detailů, které byly vybrány jako nejzajímavější nebo nejspecifičtější pro vytvořený program. V části 4.1 jsou zmíněny využité způsoby paralelizace, část 4.2 je věnována generátoru náhodných čísel a jeho počáteční hodnotě.Část 4.3 a 4.4 jsou potom věnovány práci se vstupem a výstupem. V poslední části 4.7 se nachází jednoduchý návod ke zkompilování a spuštění výsledného programu pod operačním systémem Linux.

4.1 Paralelizace

Jak bylo uvedeno výše, koevoluční varianta symbolické regrese je spouštěna ve dvou paralelních vláknech. Pro paralelizaci jsem zvolila vlákna daná standardem POSIX, jak jsou popsána v [7]. Jako sdílenou paměť využívanou pro oba dva archivy – archiv kandidátních řešení a archiv obsahující aktuálně využívaný test, využívám objekty sdílené paměti dle standardu POSIX. Tyto objekty jsou namapovány pomocí funkcí *shm_open* a *mmap*, které jsou definovány v systémové knihovně pro management sdílené paměti – *sys/mman.h*.

Pro zamezení konfliktům při přístupu do sdílené paměti jsou oba dva archivy chráněny binárními semafory dle standardu POSIX – mutexy (*mutual exclusion*). Pro minimalizaci

doby strávené v kritických sekcích jsem se rozhodla při každé práci se sdílenou pamětí nejprve obsah sdílené paměti zkopírovat do lokální paměti a teprve poté nad ní provádět dané funkce.

4.2 Generování pseudonáhodných čísel

Způsob generování pseudonáhodných čísel je v této práci klíčovou otázkou, protože generátor je využíván téměř ve všech fázích evoluce, jako je generování počáteční populace nebo výběr mutovaných uzlů a genů v každém evolučním kroku.

Využila jsem funkci *rand* ze standardní knihovny jazyka C. Tato funkce generuje sekvenci pseudonáhodných čísel, která je opakovatelná za předpokladu, že funkce *rand* dostane stejnou počáteční hodnotu. V jednodušších úlohách lze jako počáteční hodnotu využít návrat funkce *time*, která vrací rozdílnou hodnotu každou sekundu. Vzhledem k tomu, že při testování této práce pomocí testovacích skriptů mnohdy dochází k několikerému spuštění během jedné sekundy, nelze tento nedostatek tolerovat. Proto jsem se rozhodla použít funkci *gettimeofday* ze systémové knihovny *sys/time.h*, která vrací strukturu timeval, ze které využívám součet hodnot v sekundách a mikrosekundách. Toto zaručuje rozdílnou počáteční hodnotu generátoru každou mikrosekundu s opakováním každých 24 hodin. Pravděpodobnost, že se testovacímu skriptu povede spustit program dvakrát přesně ve stejnou mikrosekundu dne je

$$\frac{1}{24 * 60 * 60 * 1000} = 1.15741 * 10^{-8} \tag{4.1}$$

což jsem se rozhodla považovat za zanedbatelné číslo.

4.3 Formát vstupních souborů

Jako uživatelský vstup využívám mimo parametrů příkazové řádky dva soubory – soubor s trénovacími vektory a soubor s povolenými funkcemi. Tyto soubory mají podobnou strukturu. Vždy na prvním řádku je zapsán počet hodnot nacházejících se v souboru. Na každém dalším řádku potom následuje právě jedna hodnota daného formátu.

4.3.1 Soubor s povolenými funkcemi

Pomocí souboru s povolenými funkcemi uživatel definuje, jaké funkce bude možné realizovat ve výpočetním uzlu, tj. ze kterých základních funkcí má být hledaná funkce složena. Jednotlivé funkce jsou definovány následujícími symbolickými konstantami.

- PLUS – sčítání
- MINUS – o1dčítání
- MUL – násobení
- DIV – dělení
- EXP – e^x
- POW – y^x
- SIN – $sin(x)$
- COS – $cos(x)$
- LOG – $ln(x)$
- ABS – absolutní hodnota
- CONST – konstanta

Každá ze zadaných hodnot se nachází právě na jednom řádku ukončeném znakem konce řádku. Pokud soubor nevyhovuje specifikované definici, program skončí s chybou.

4.3.2 Soubor s trénovacími vektory

Prostřednictvím tohoto souboru uživatel definuje trénovací vektory pro danou úlohu. Na každém řádku se nachází právě jeden trénovací vektor na jehož prvních n_i pozicích se nachází hodnoty odpovídající primárním vstupům, na dalších n_o pozicích se nachází odpovídající očekávané výstupy.

Následuje krátká ukázka testovacího souboru použitého pro funkci $e^x * sin(x)$. Na prvním řádku je specifikováno, že v souboru se nachází právě 201 trénovacích vektorů, na každém dalším řádku je vstup funkce a očekávaný výstup.

```
201
-10.000000 11982.862391
-9.900000 9118.859852
-9.800000 6608.991138
...
```

4.4 Načítání vstupu

Hodnoty z obou dvou vstupních souborů jsou na začátku načteny do datových struktur. Načítání probíhá po znaku pomocí funkce *fgetc* ze standardní knihovny jazyka C, dokud není dosaženo bílého znaku (mezery či konce řádku), poté je načtený řetězec zpracován. V případě zpracování dat ze souboru s používanými funkcemi je vyhodnoceno, zda se jedná o validní symbolickou konstantu pro některou z funkcí. V případě zpracování dat ze souboru s trénovacími vektory je řetězec převeden na číslo typu double za pomoci funkce *strtod* ze standardní knihovny jazyka C. Tato funkce je schopna zpracovat i čísla zapsaná v exponenciálním tvaru. Jako oddělovač desetinné části čísla je používán takový znak, který odpovídá lokalizačnímu nastavení operačního systému – většinou tedy tečka, v českých variantách systému čárka.

4.5 Záznamy statistických dat

Pro porovnávání běhu programu s různými nastaveními je třeba zaznamenávat statistiky různých druhů. V této práci je průběžně počítáno několik různých statistických hodnot:

Počet generací – celkový počet evolučních kroků od počátku běhu programu po dosažení požadovaného řešení.

Čas výpočtu – metriky využití zdrojů během výpočtu měřené pomocí funkce *getrusage* ze systémové knihovny *sys/resource.h*. V případě koevoluční varianty se jedná o součet využití zdrojů z obou vláken.

Počet vyčíslených trénovacích vektorů – celkový počet vyčíslení trénovacích vektorů za dobu běhu programu. Tedy kolikrát během celé evoluce byly do některého kandidátního řešení načteny vstupní hodnoty a vypočteny výstupní.

Po skončení běhu jsou společně s výsledným řešením vypsány konečné hodnoty těchto měření. Jednou za 100 generací je také při běhu programu vypisována aktuální nejvyšší fitness, díky čemuž lze sledovat, jakým způsobem se fitness při jednotlivých nastaveních vyvíjí.

4.6 Výpis řetězce řešení

Pro usnadnění práce s výsledným řešením je na konci běhu také vypsán matematický vztah, který výsledné řešení představuje. Tato funkcionalita slouží hlavně k usnadnění kontroly správnosti dosaženého řešení.

Výpis řetězce představujícího nalezenou funkci probíhá pomocí rekurzivního sestupu s využitím dvousměrně vázaného seznamu. Na začátku je do seznamu vložen neterminální uzel stromu řešení, který je postupně rozvíjen dle hodnot v chromozomu. Algoritmus končí, jakmile jsou všechny větve rozvinuty až k listům – k primárním vstupům programu.

4.7 Kompilace a spuštění

Pro kompilaci a spuštění programu využívám *Makefile* pro program *make*, pomocí kterého lze program jak zkompilovat, tak spustit varianty s koevolucí i bez koevoluce. Nastavení vstupních souborů a veškerých parametrů je třeba provést přímo v souboru Makefile.

`make` – *přeložení a slinkování celého programu*
`make run` – *spuštění varianty bez koevoluce*
`make coev` – *spuštění varianty s koevolucí*

Kapitola 5

Experimentální vyhodnocení

Účelem této práce je vytvořit program, který řeší symbolickou regresi pomocí kartézského genetického programování. Pro ověření funkčnosti programu byla použita sada testů používaná v rámci komunity věnující se symbolické regresi.

Tato kapitola shrnuje průběh testování a metody výběru parametrů pro jednotlivé části výpočtu (5.1 pro evoluci kandidátních řešení a 5.2 pro evoluci testů). V části 5.3 jsou popsány skripty využité pro automatizované testování a poloautomatizované zpracování výsledků. Důležitou částí této kapitoly je také 5.4, ve které jsou shrnuty výsledky testování pro jednotlivé funkce.

Funkce f_1, f_2 a f_3 lze zařadit do kategorie jednodušších testů, které slouží hlavně k ověření, zda symbolická regrese pracuje správně. Funkce f_4 a f_5 jsou náročnější testy, které jsou převzaty z webové stránky věnující se zátěžovým testům symbolické regrese [13]. Použití testů spočívá ve využití předepsaných vztahů pro vygenerování trénovací množiny (201 vzorků předepsané funkce na zadaném intervalu), nad kterou potom probíhá evoluce. Množiny trénovacích vektorů pro jednotlivé funkce jsou znázorněny na obrázku 5.1.

Funkční předpisy jednotlivých zátěžových testů

$$f_1 : f(x) = x^2 - x^3,\ x \in \langle -10, 10 \rangle \tag{5.1}$$

$$f_2 : f(x) = e^{|x|} sin(x),\ x \in \langle -10, 10 \rangle \tag{5.2}$$

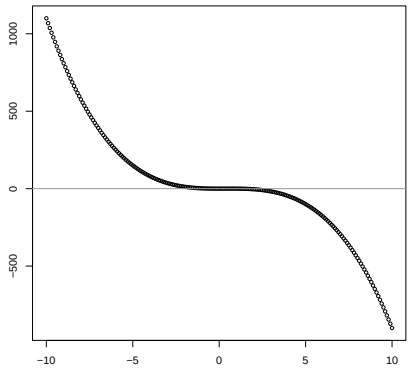
$$f_3 : f(x) = x^2 e^{sin(x)} + x + sin(\frac{\pi}{x^3}),\ x \in \langle -10, 10 \rangle \tag{5.3}$$

$$f_4 : f(x) = e^{-x} x^3 cos(x) sin(x) (cos(x) sin(x)^2 - 1),\ x \in \langle 0, 10 \rangle \tag{5.4}$$

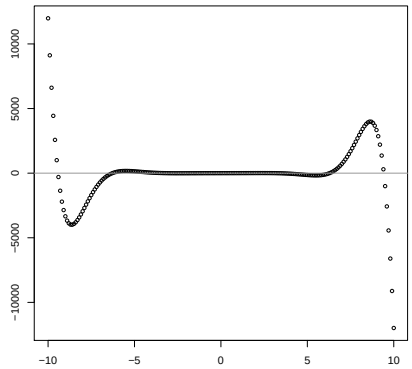
$$f_5 : f(x) = \frac{10}{5 + (x-3)^2},\ x \in \langle -2, 8 \rangle \tag{5.5}$$

5.1 Nastavení evoluce kandidátních řešení

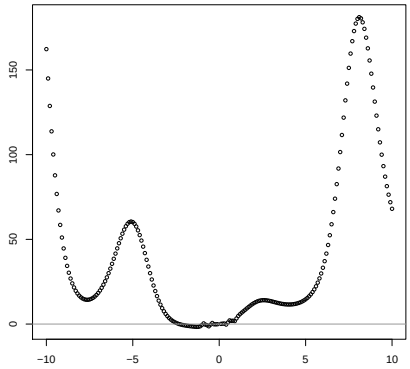
V symbolické regresi se vyskytuje mnoho uživatelsky modifikovatelných parametrů. Vzhledem k charakteru této práce – porovnání techniky soutěživé koevoluce s technikou koevoluce prediktorů fitness, byly parametry pro CGP zvoleny tak, jak je popsáno v článku [12]. Tento článek popisuje experimenty na sadě testů použité i v této práci za využití koevoluce prediktorů fitness. Zvolené parametry jsou zapsány v tabulce 5.1.



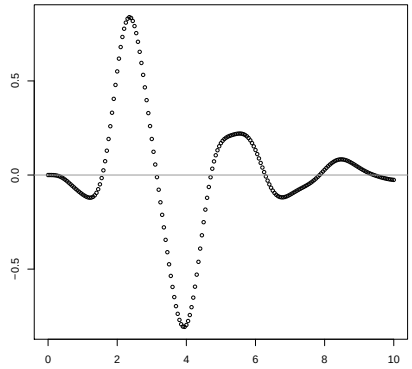
(a) Trénovací množina f_1



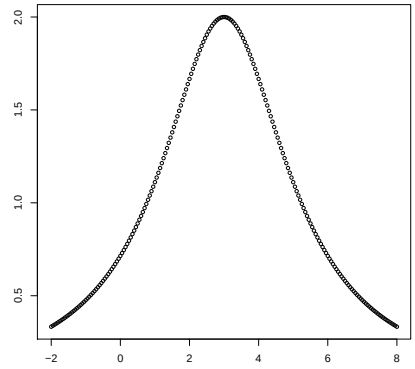
(b) Trénovací množina f_2



(c) Trénovací množina f_3



(d) Trénovací množina f_4



(e) Trénovací množina f_5

Obrázek 5.1: Zobrazení trénovacích vektorů v rovině.

Počet řádků v chromozomu	1
Počet sloupců v chromozomu	32
Počet mutovaných genů [1]	1-8%
Procento maximální fitness akceptované jako řešení	97
Tolerance hitu u jednotlivých funkcí	f_1, f_2: 0.5, f_3: 1.5, f_4, f_5: 0.025
Maximální počet generací	f_1, f_2, f_3: 8 000 000, f_4, f_5: 16 000 000

Tabulka 5.1: Nastavení CGP přejaté z [12].

Parametr	Testované hodnoty	Vybraná hodnota
Velikost archivu (v násobcích velikosti populace)	1, 2, 3	2
Počet bodů křížení	1, 2, 3	3
Počet prvků v generaci	8, 16, 32	32
Složení generace (nejlepší jedinci - kříženci - nově generovaní jedinci)	12-10-10, 10-12-10, 10-10-12, 16-8-8 8-16-8, 8-8-16	8-16-8
Počet trénovacích vektorů v testu	8, 12, 16, 20	f_1, f_2, f_3: 8, f_4, f_5: 16

Tabulka 5.2: Experimentálně vybrané nastavení koevoluce.

5.2 Nastavení evoluce testů

S koevolucí přichází mnoho dalších parametrů, jejichž co nejvýhodnější nastavení je třeba najít. Hledání tohoto nastavení bylo provedeno na funkci f_1, nad kterou byla koevoluce spuštěna s několika nastaveními, ze kterých poté byla vybrána co nejvýhodnější hodnota. Toto srovnání bylo provedeno vizuální metodou v grafech znázorněných na obrázku 5.2. Jako hlavní parametr pro srovnání byl zvolen počet vyčíslených trénovacích vektorů, jehož znázornění je uvedeno na obrázku 5.2. Dále byly porovnávány následující parametry: počet generací, čas řešení a procento úspěšných běhů. Grafy pro tyto parametry jsou uvedeny v příloze B.1.

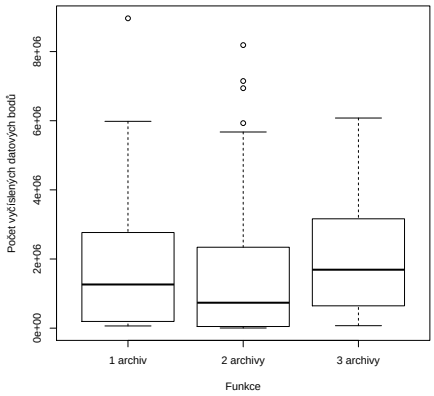
Parametry, jejichž vliv na rychlost řešení je zaznamenán v jednotlivých grafech, byly testovány za nezměněných okolních podmínek. Na základě těchto výsledků byly jako nejvhodnější vybrány parametry uvedené v tabulce 5.2.

5.3 Skripty pro testování a vyhodnocování

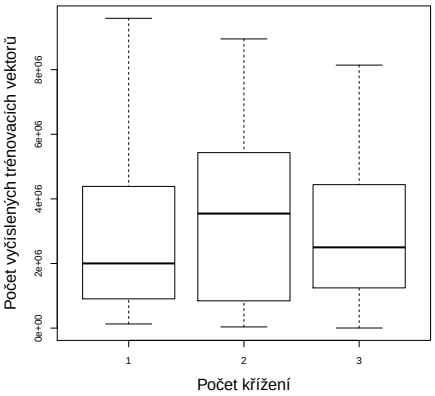
Pro testování byla použita sada jednoduchých testovacích skriptů v jazyce bash. Každý ze skriptů obsahoval právě jedno testované nastavení programu. Pro dané nastavení byl běh opakovaně spouštěn (50krát u jednodušších funkcí, 30krát u funkcí složitějších). Při každém spuštění byly vytvořeny dva soubory - soubor s výsledným řešením a souhrnnými statistikami za celý běh a soubor obsahující průběžné hodnoty fitness vypisované každých 100 generací.

Pro vyhodnocení výsledků byla použita dvojice skriptů – skript v jazyce bash pro

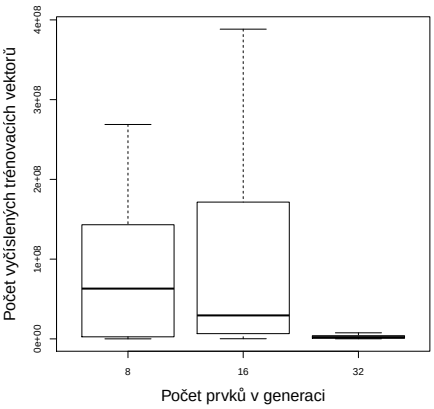
[1] Pro obecnost je v této práci počet vyjádřen v procentech – 1-8% genů je mutováno, tedy 1-8% z 97 (32*3+1) genů. Při zaokrouhlení nahoru tato hodnota přesně odpovídá 1-8 mutovaným genům na jedince.



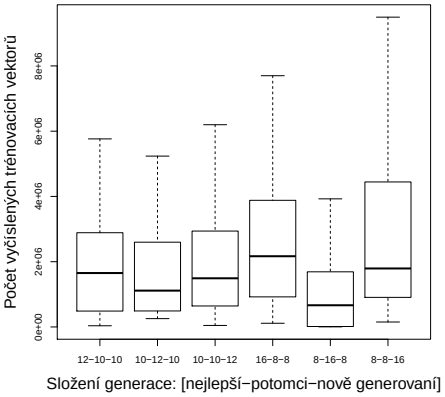
(a)



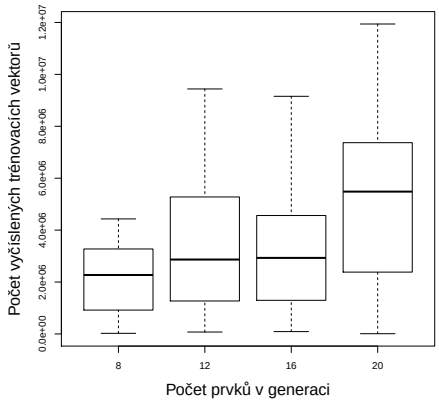
(b)



(c)



(d)



(e)

Obrázek 5.2: Porovnání nastavení koevoluce.

načtení dat zaznamenaných do statistických souborů. Z tohoto skriptu byl opakovaně volán skript v jazyce R, který zajišťoval zpracování načtených dat do kvartilových grafů a jejich vykreslení do finální podoby prezentované v této práci.

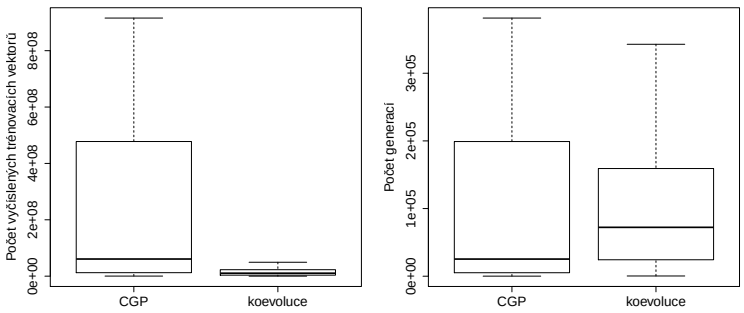
5.4 Posuzovaná kritéria a výsledky

Při porovnání výsledků programu s koevolucí a bez koevoluce byly porovnávány stejné parametry jako při výběru parametrů pro koevoluci. Hlavními očekávanými přínosy koevolučního řešení bylo zkrácení doby výpočtu a omezení případů, kdy dojde k uváznutí v lokálním extrému.

5.4.1 Funkce f_1

U nejjednodušší funkce f_1 bylo při standardním CGP nalezeno řešení v 96% spuštění. Při zbylých 4% spuštění došlo k uváznutí v lokálním extrému. Tento nedostatek byl použitím koevoluce odstraněn – při jejím využití bylo nalezeno řešení ve 100% spuštění. I ostatní předpokládané přínosy koevoluce byly dosáhnuty. Graf ukazující, že při využití koevoluce bylo třeba podstatně méně vyčíslení trénovacích vektorů i generací, je uveden na obrázku 5.3, další grafy jsou uvedeny v příloze (B.4).

Urychlení výpočtu bylo vypočteno podle dvou kritérií – podle počtu vyčíslených trénovacích vektorů a podle času. V čase byla koevoluční varianta na této funkci 2x rychlejší, vyčíslení trénovacích vektorů však bylo potřeba 27x méně. Rozdílné výsledky jsou způsobeny hlavně tím, že koevoluční varianta pracuje ve dvou vláknech, což znamená mnoho režie navíc, která zabere procesorový čas.



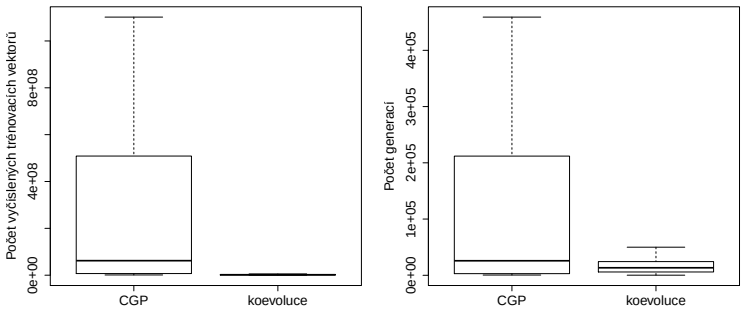
Obrázek 5.3: Srovnání výsledků CGP a koevoluce pro funkci $x^2 - x^3$.

5.4.2 Funkce f_2

Při testování na funkci f_2 bylo při využití standardního CGP nalezeno řešení v 98% spuštění. Při zbylých 2%, tedy při jednom běhu z padesáti, došlo k uváznutí v lokálním extrému. Při nasazení koevoluční varianty programu bylo nalezeno řešení při 100% spuštění. Graf ukazující porovnání počtu generací a počtu vyčíslení trénovacích vektorů pro obě varianty je uveden na obrázku 5.4, další grafy jsou uvedeny v příloze (B.5).

Na této funkci bylo urychlení pomocí koevoluce nejvýznamnější, průměrný čas výpočtu byl s koevoluční variantou 181x kratší, průměrný počet vyčíslených trénovacích vektorů byl

pak dokonce 336x nižší. Z grafu na obrázku 5.4b je patrné, že v koevoluční variantě byl k nalezení řešení potřeba výrazně nižší počet generací, v každé generaci navíc místo 201 trénovacích vektorů bylo vyčíslováno pouze 8.



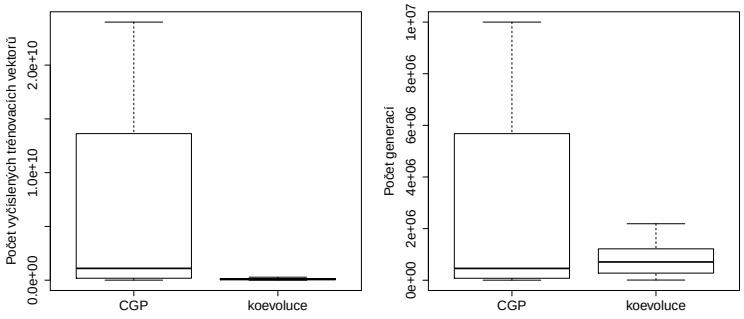
Obrázek 5.4: Srovnání výsledků CGP a koevoluce pro funkci $sin(x) * e^{|x|}$.

5.4.3 Funkce f_3

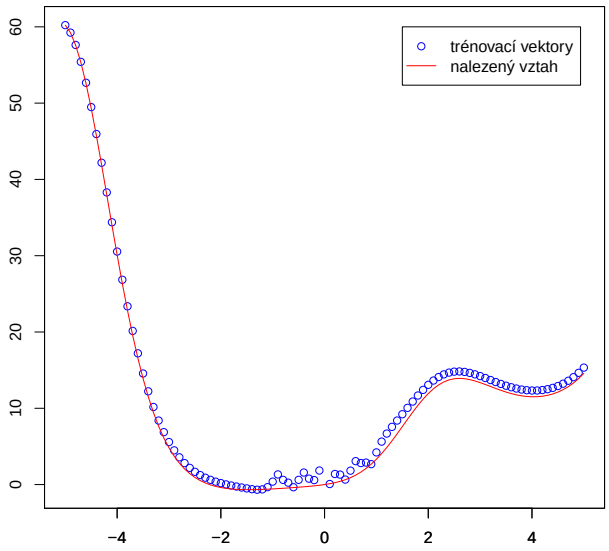
Při hledání funkce f_3 pomocí standardního CGP bylo řešení nalezeno v 82% spuštění. Při řešení s koevolucí bylo řešení nalezeno ve 100% případů. Srovnání efektivity hledání na základě počtu vyčíslených trénovacích vektorů a počtu generací je uvedeno na obrázku 5.5. Další grafy jsou uvedeny v příloze (B.6).

Urychlení pomocí koevoluce na této funkci bylo velmi významné. Při porovnání z hlediska výpočetního času byl výpočet pomocí koevoluce urychlen 14x, z hlediska počtu vyčíslených trénovacích vektorů byl potom výpočet urychlen 28x.

Zajímavostí u funkce f_3 je člen $sin(\frac{\pi}{x^3})$. Tento člen je totiž ve výsledných řešeních bez ztráty fitness naprosto zanedbáván. Nalezená řešení odpovídají funkci $x^2e^{sin(x)} + x$. Porovnání nalezené funkce s trénovacími vektory (na části definičního oboru, kde má zanedbaný člen největší vliv) je ukázáno na obrázku 5.6. Je vidět drobná nepřesnost, která však odpovídá toleranci zvolené pro tuto funkci.



Obrázek 5.5: Srovnání výsledků CGP a koevoluce pro funkci $x^2e^{sin(x)} + x + sin(\frac{\pi}{x^3})$.



Obrázek 5.6: Srovnání nalezené funkce $x^2e^{sin(x)}+x$ s hledanou funkcí $x^2e^{sin(x)}+x+sin(\frac{\pi}{x^3})$.

5.4.4 Funkce f_4

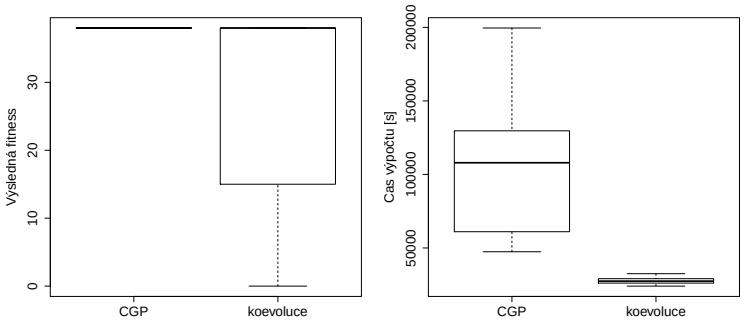
Při hledání funkce f_4 klasické CGP selhalo a nenalezlo řešení ani v jednom ze třiceti provedených spuštění. Způsobeno to bylo uváznutím v lokálním extrému. Ačkoliv některé druhy koevoluce hledání řešení této funkce dokázaly optimalizovat, soutěživá koevoluce nebyla při testování schopna hledání řešení nijak ovlivnit. Ani jedno řešení nebylo nalezeno ani při testování koevoluční metody. V tomto případě se však nemluví o uváznutí v lokálním extrému, nýbrž o cyklení ve výpočetních smyčkách.

Na obrázku 5.7a je znázorněn rozdíl mezi uváznutím v lokálním extrému u CGP a cyklení u soutěživé koevoluce. CGP se pokaždé dostalo na stejnou hodnotu fitness (lokální extrém), na které uvázlo a skončilo. Koevoluce končila s několika rozdílnými hodnotami fitness. Na obrázku 5.7b lze potom pozorovat časový rozdíl ve vyčíslení 16 milionů generací pomocí CGP oproti CGP s koevolucí. Ostatní grafy jsou uvedeny v příloze (B.7).

5.4.5 Funkce f_5

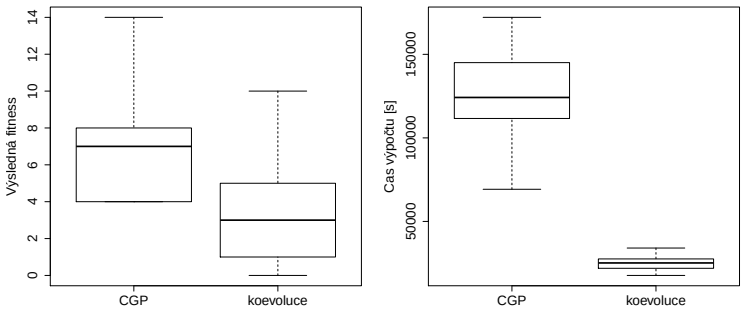
Při hledání funkce f_5, stejně jako u funkce f_4, klasické CGP selhalo a nenalezlo řešení ani v jednom ze třiceti provedených spuštění. Koevoluční varianta na tom nebyla lépe – také se jí nepodařilo nalézt ani jedno řešení. Ačkoliv se bezpochyby jedná o nejsložitější ze zadaných funkcí, jiné koevoluční mechanismy dokázaly procento nalezených řešení podstatně zvýšit. Toto se soutěživé koevoluci nepodařilo, což poukazuje na zjištěnou vlastnost sotěživé koevoluce – dokáže velmi efektivně optimalizovat řešení jednoduchých funkcí, ale k řešení složitějších funkcí není vhodná.

Na obrázku 5.8a lze pozorovat, že ve funkci f_5 bylo více lokálních extrémů, ve kterých CGP při výpočtu uvázlo. Koevoluční varianta opět cyklila ve výpočetních cyklech. Výpočetní čas (znázorněn na obrázku 5.8b), ačkoliv nevedl k úspěšnému řešení, je i při řešení f_5 podstatně nižší u koevolučního přístupu, než u CGP. Toto je způsobeno množstvím



(a) (b)

Obrázek 5.7: Srovnání výsledků CGP a koevoluce pro funkci $e^{-x}x^3cos(x)sin(x)(cos(x)sin(x)^2 - 1)$.



(a) (b)

Obrázek 5.8: Srovnání výsledků CGP a koevoluce pro funkci $\frac{10}{5+(x-3)^2}$.

vyčíslených trénovacích vektorů při jednotlivých variantách. Další grafy jsou uvedeny v příloze (B.8).

5.5 Shrnutí výsledků

Testování této práce bylo provedeno na pěti funkcích, které byly rozděleny na skupinu jednodušších (f_1, f_2 a f_3) a složitějších (f_4, f_5) funkcí. Koevoluční varianta se u funkcí f_1 až f_3 dokázala vyhnout případům uváznutí v lokálním extrému, zkrátila dobu výpočtu a snížila jak počet generací, tak počet vyčíslených trénovacích vektorů. U funkcí f_4 a f_5 se koevoluční variantě nepodařilo nalézt požadované výsledky, což bylo pravděpodobně způsobeno známým problémem soutěživé koevoluce – zacyklením se do výpočetních smyček.

Porovnání s koevolucí prediktorů fitness dle [12] ukazuje, že soutěživá koevoluce, jako jednodušší typ koevoluce, je vhodnější pro řešení jednodušších úloh, na složitějších úlohách potom ale selhává. Koevoluci prediktorů fitness se podařilo urychlit všechny úlohy popsané v této kapitole 2-5krát. Soutěživé koevoluci se pak povedlo urychlit úlohy f_1 až f_3 2-181krát, úlohy f_4 a f_5 ale vyřešit nedokázala.

Kapitola 6

Závěr

Hlavním cílem této práce bylo navrhnout, implementovat a otestovat program řešící symbolickou regresi pomocí kartézského genetického programování a soutěživé koevoluce. Tento program byl implementován ve dvou variantách – s využitím koevoluce a za pomoci standardního CGP – a funguje v prostředí operačního systému Linux. Výstupem této práce jsou testy provedené na pěti vybraných úlohách, které prověřily funkčnost programu. Během testování se ukázalo, že koevoluční varianta vykazuje lepší výsledky ve všech ohledech – čas výpočtu, počet vyčíslených trénovacích vektorů, ale i zamezení případům uváznutí v lokálním extrému u jednodušších funkcí. U složitějších funkcí pak ale soutěživá koevoluce selhává v základním požadavku na symbolickou regresi – nalézt řešení.

Součástí práce bylo také srovnání výsledků s předchozím přístupem řešení symbolické regrese pomocí kartézského genetického programování s využitím koevoluce prediktorů fitness. Očekávaným výsledkem bylo, že soutěživá koevoluce bude dosahovat podstatně menších zrychlení než koevoluce prediktorů fitness. V průběhu testování se však ukázalo, že při řešení jednodušších úloh je urychlení pomocí soutěživé koevoluce vyšší, než urychlení pomocí koevoluce prediktorů fitness. Při řešení složitějších úloh se však soutěživá koevoluce dostává do výpočetních cyklů, které mají za důsledek nenalezení výsledného řešení ve zkoumaném počtu generací.

Za stejných podmínek, za jakých se koevoluci prediktorů fitness povedlo urychlit testovací úlohy f_1 až f_3 2-5krát, se soutěživé koevoluci povedlo výpočet urychlit 2-181krát. Při řešení funkcí f_4 a f_5 však soutěživá koevoluce nedokázala nalézt řešení ve stanoveném počtu 16 milionů generací.

Na tuto práci by bylo možné navázat jejím rozšířením o uživatelské rozhraní, které by mohlo program přiblížit potenciálním uživatelům například v prostředí výzkumných laboratoří a podobně.

V průběhu řešení této práce jsem se seznámila s metodami soft-computingu, kterým se budu dále při studiu věnovat. Při implementaci jsem si v praxi vyzkoušela metody paralelizace a naučila jsem se práci s jazykem R, který je velmi užitečný při zpracování statistických dat. V neposlední řadě jsem se také naučila zpracovávat technickou dokumentaci pomocí sazebního systému LaTeX.

Literatura

[1] Bidlo, M.: *Evolutionary Design of Generic Structures Using Instruction-Based Development*. Dizertační práce, 2008.

[2] Chalupník, V.: Biologické algoritmy (1) - Evoluční algoritmy. *root.cz*.
URL
http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/

[3] Harding, S. L.; Miller, J. F.; Banzhaf, W.; aj.: Self-modifying cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*, New York: Springer Berlin Heidelberg, 2007, ISBN 978-3-642-17309-7, s. 133–144.

[4] Holland, J. H.: *Adaptation in natural and artificial systems*. Cambridge: Bradford Book, c1992, ISBN 02-625-8111-6.

[5] Janzen, D. H.: When is it Coevolution? *Evolution*, ročník 34, č. 3, 1980: s. 611–6, ISSN 0014-3820.

[6] Koza, J.; Langdon, W. B.; Miller, J. F.: *Genetic programming as a means for programming computers by natural selection*. ISBN 978-0-387-25067-0.

[7] Lampa, P.: Paralelní programování s použitím vláken dle normy POSIX 1003.1.
URL http://www.fit.vutbr.cz/~lampa/papers/vlakna96.html.cs

[8] Miller, J. F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, ročník 2, 1999.

[9] Miller, J. F.; Thomson, P.: Cartesian genetic programming. In *Genetic Programming*, Springer, 2000, s. 121–132.

[10] Schwefel, H.-P.: Advantages (and disadvantages) of evolutionary computation over other approaches. *Evolutionary computation*, ročník 1, 2000: s. 20–22.

[11] Sekanina, L.; Vašíček, Z.; Ružička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojum*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.

[12] Šikulová, M.; Sekanina, L.: Coevolution in Cartesian Genetic Programming. In *Genetic programming*, New York: Springer, první vydání, 2012, ISBN 3642291384.

[13] Vladislavleva, K.: Symbolic Regression. 2011.
URL http://symbolicregression.com/

Příloha A

Obsah CD

Technická zpráva

Technická zpráva ve formátu `pdf` se nachází v adresáři `theory`. Zdrojové kódy pro LaTeXa všechny využité balíčky se nachází v adresáři `theory/src`. Po případně provedených změnách lze technickou zprávu ve formátu pdf vygenerovat pomocí příkazu `make`.

Praktická část

Adresář `practice` obsahuje následující souborovou strukturu:

src - v tomto adresáři se nachází veškeré zdrojové kódy, po přeložení příkazem `make` se zde nachází také soubory spustitelné příkazy `make run` - pro variantu využívající klasické CGP, případně `make coev` pro variantu využívající koevoluci.

bin - v tomto adresáři se nachází obě verze spustitelného programu. `cgp` provádí symbolickou regresi se standardním CGP, `coecgp` pak provádí optimalizovanou koevoluční variantu CGP.

testfiles - tento adresář obsahuje testovací soubory pro funkce f_1 až f_5 využité při testování.

testscripts - v tomto adresáři se nachází testovací skripty pro jednotlivé funkce, využité v rámci testování vytvořené aplikace.

results - v tomto adresáři se nachází adresář `res_processing`, ve kterém se nachází dvojice skriptů v jazycích `bash` a `R`, které byly využity pro zpracování výsledků. Dále se zde nachází adresář `res_data` obsahující statistické soubory z testování funkcí f_1 až f_5. V adresáři `res_graphs` jsou umístěny grafy vygenerované dvojicí skriptů v adresáři `res_processing`.

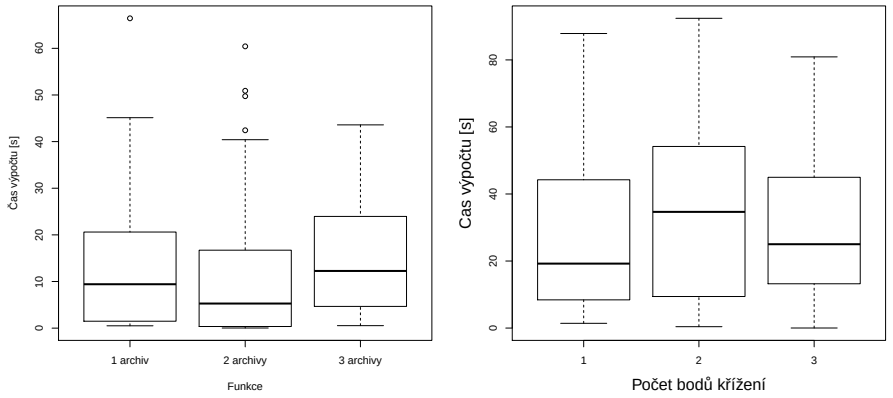
Příloha B

Grafy s rozšířenými výsledky testů

B.1 Grafy k výběru parametrů koevoluce

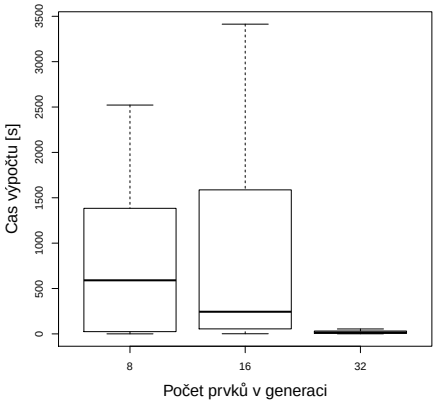
Tato část práce obsahuje grafy znázorňující srovnání jednotlivých nastavení koevoluce. Ve vlastním textu práce bylo uvedeno srovnání na základě počtu vyčíslených vektorů. V této příloze jsou přidána také srovnání na základě času stráveného na procesoru, na základě počtu generací vedoucích k nalezení řešení a na základě maximální dosažené fitness. Hodnota maximální dosažené fitness má význam v případě, že nebylo nalezeno řešení.

Testy byly prováděny na padesáti spuštěních s každým nastavením. Změny porovnávaných parametrů byly prováděny za nezměněných ostatních podmínek.

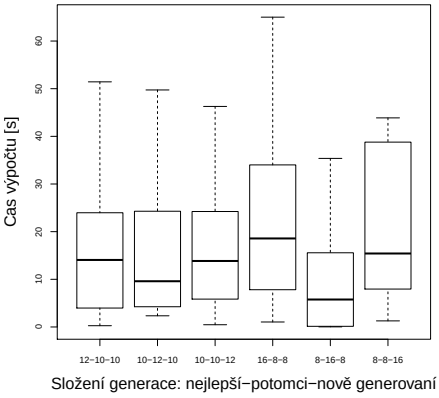


(a)

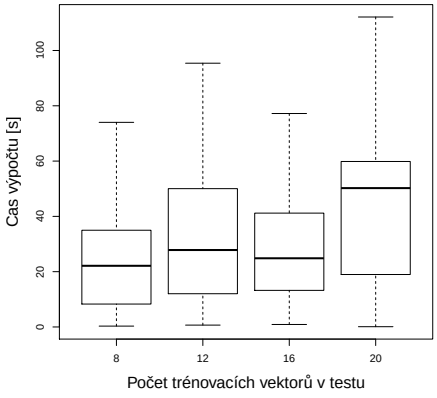
(b)



(c)

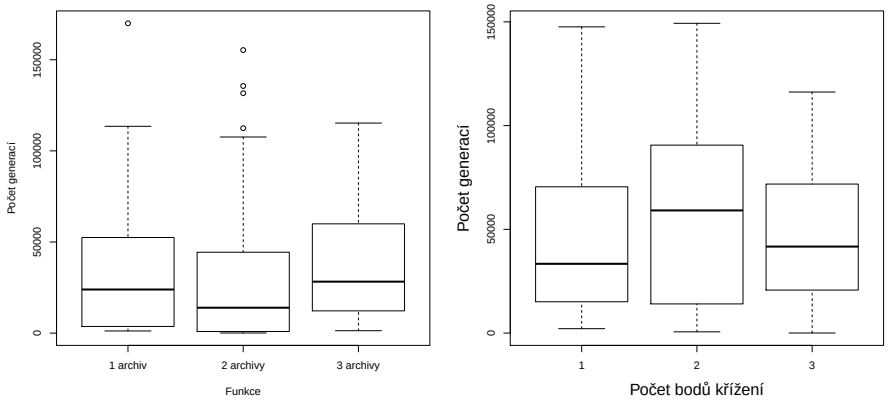


(d)



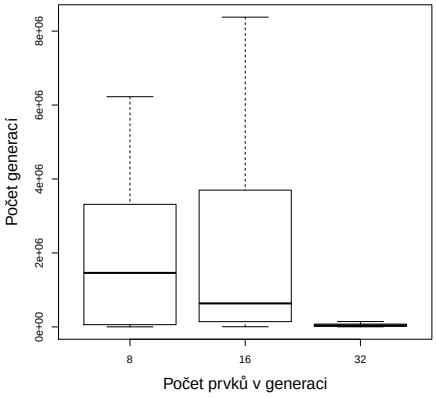
(e)

Obrázek B.1: Porovnání jednotlivých nastavení koevoluce na základě času.

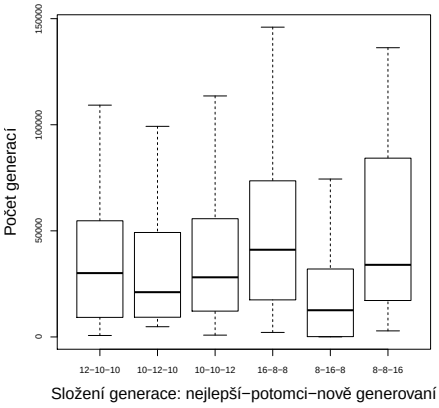


(a)

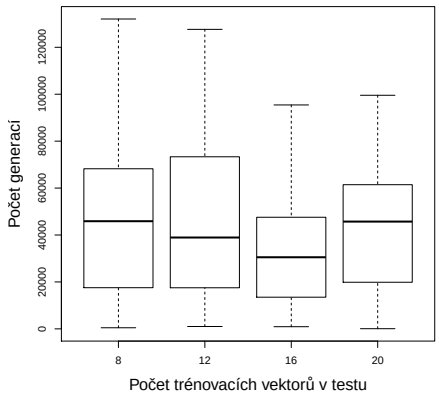
(b)



(c)

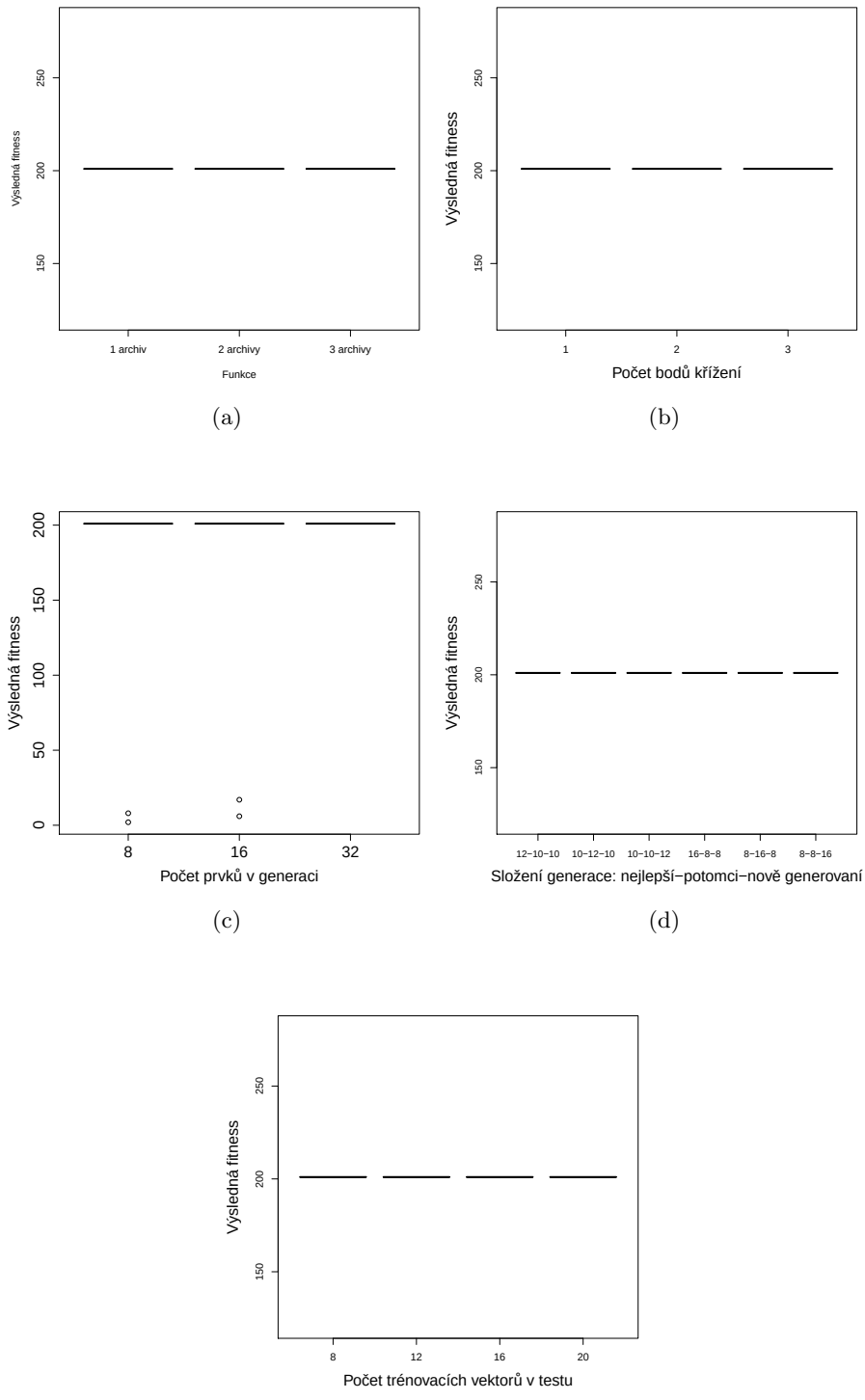


(d)



(e)

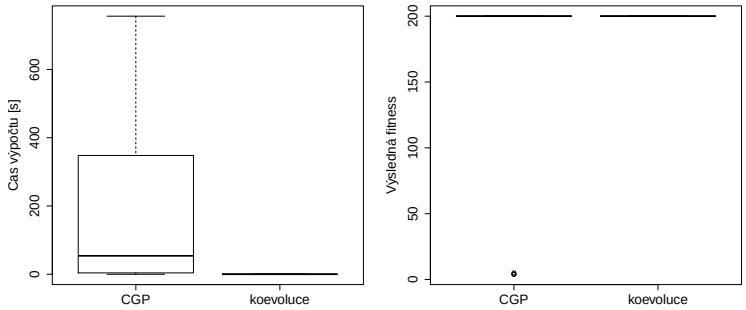
Obrázek B.2: Porovnání jednotlivých nastavení koevoluce na základě počtu generací.



Obrázek B.3: Porovnání jednotlivých nastavení koevoluce na základě maximální dosažené fitness.

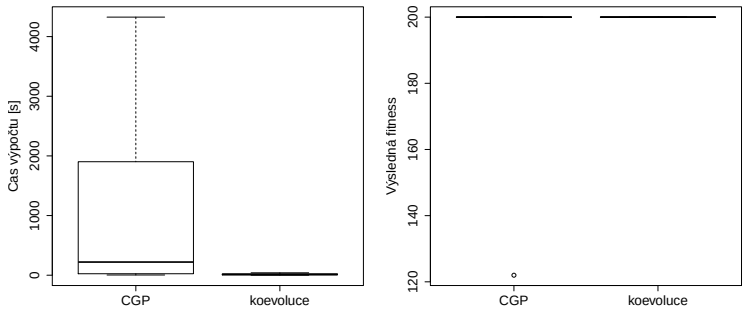
B.2 Grafy k výsledkům testování pro jednotlivé funkce

B.2.1 Funkce f_1



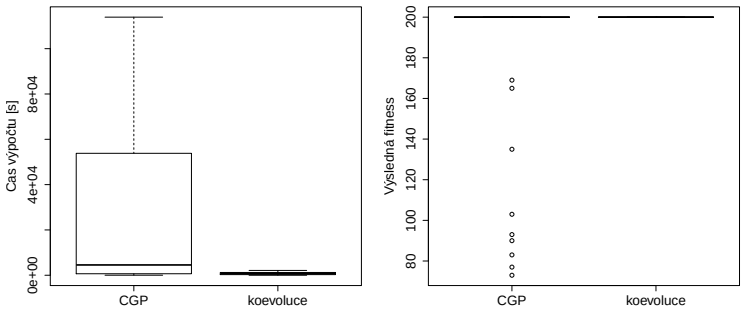
Obrázek B.4: Srovnání výsledků CGP a koevoluce pro funkci $x^2 - x^3$.

B.2.2 Funkce f_2



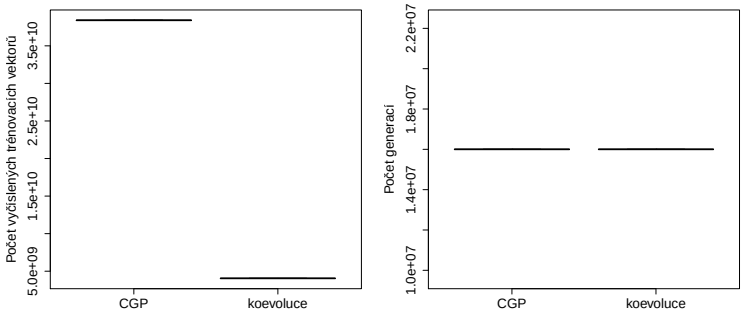
Obrázek B.5: Srovnání výsledků CGP a koevoluce pro funkci $sin(x) * e^{|x|}$.

B.2.3 Funkce f_3



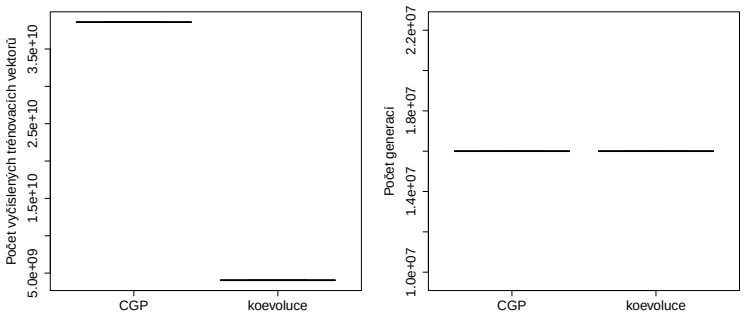
Obrázek B.6: Srovnání výsledků CGP a koevoluce pro funkci $x^2 e^{sin(x)} + x + sin(\frac{\pi}{x^3})$.

B.2.4 Funkce f_4



Obrázek B.7: Srovnání výsledků CGP a koevoluce pro funkci $e^{-x} x^3 cos(x) sin(x)(cos(x) sin(x)^2 - 1)$.

B.2.5 Funkce f_5



Obrázek B.8: Srovnání výsledků CGP a koevoluce pro funkci $\frac{10}{5+(x-3)^2}$.