VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

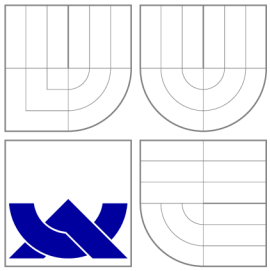
SYMBOLICKÁ REGRESE A KOEVOLUCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

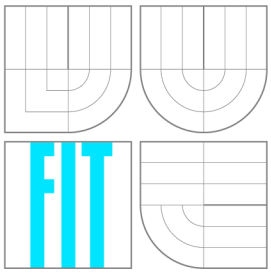
AUTOR PRÁCE
AUTHOR

Bc. MICHAELA ŠIKULOVÁ

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYMBOLICKÁ REGRESE A KOEVOLUCE

SYMBOLIC REGRESSION AND COEVOLUTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAELA ŠIKULOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2011

Abstrakt

Symbolická regrese je úloha identifikace matematického popisu skryté závislosti experimentálně získaných dat. Symbolická regrese je úzce spjata se základními úlohami strojového učení. Tato práce se zabývá symbolickou regresí a jejím řešením založeném na principu genetického programování a koevoluce. Genetické programování je evolucí inspirovaná metoda strojového učení, která automaticky generuje celé programy v určitém programovacím jazyce. Koevoluce fitness prediktorů je optimalizační metoda modelování fitness, která snižuje náročnost a frekvenci výpočtu fitness. Tato práce se zabývá návrhem a implementací řešení symbolické regrese s užitím koevoluce fitness prediktorů a srovnáním s řešením bez užití koevoluce. Experimenty byly provedeny s použitím kartézského genetického programování.

Abstract

Symbolic regression is the problem of identifying the mathematic description of a hidden system from experimental data. Symbolic regression is closely related to general machine learning. This work deals with symbolic regression and its solution based on the principle of genetic programming and coevolution. Genetic programming is the evolution based machine learning method, which automaticaly generates whole programs in the given programming language. Coevolution of fitness predictors is the optimalization method of the fitness modelling that reduces the fitness evaluation cost and frequency, while maintainig evolutionary progress. This work deals with concept and implementation of the solution of symbolic regression using coevolution of fitness predictors, and its comparison to a solution without coevolution. Experiments were performed using cartesian genetic programming.

Klíčová slova

Symbolická regrese, evoluční algoritmy, genetické programování, kartézské genetické programování, koevoluce, modelování fitness, prediktory fitness.

Keywords

Symbolic regression, evolutionary algorithms, genetic programming, cartesian genetic programming, coevolution, fitness modeling, fitness predictors.

Citace

Michaela Šikulová: Symbolická regrese a koevoluce, diplomová práce, Brno, FIT VUT v Brně, 2011

Symbolická regrese a koevoluce

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením docenta Lukáše Sekaniny. Uvedla jsem všechny literární prameny, ze kterých jsem čerpala.

.....
Michaela Šikulová
18. května 2011

Poděkování

Na tomto místě bych ráda poděkovala doc. Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení a také za cenné rady a konzultace při vypracování této diplomové práce.

Obsah

Kapitola 1

Úvod

V této práci se zabývám symbolickou regresí a jejím řešením pomocí evolucí inspirovaných metod genetického programování a koevoluce. Pro tento účel využívám znalostí získaných z předmětů Aplikované evoluční algoritmy a Biologií inspirované počítače.

Symbolická regrese je základní úlohou strojového učení a zabývá se identifikací matematického popisu skryté závislosti experimentálně získaných dat – hledá se funkce, která nejlépe aproximuje data. Již po staletí se vědci snaží nalézt přírodní zákony. Symbolická regrese umožňuje identifikovat tyto zákony bez znalostí fyziky, kinematiky, geometrie, atp., na základě zpracování experimentálně získaných dat. Symbolická regrese se využívá pro popis systémů, které obsahují mezery navzdory velkému množství dat z pozorování, od biologie až po kosmologii.

Cílem této práce je nastudovat problematiku symbolické regrese a jejího řešení pomocí genetického programování a koevoluce, navrhnout a implementovat řešení s užitím koevoluce a bez koevoluce a tyto dva přístupy porovnat. Oproti stávajícím přístupům se tato práce zaměřuje na využití kartézského genetického programování.

Technická zpráva je rozdělena do osmi kapitol. Kapitola 2 je teoretickým úvodem a obsahuje základní pojmy, principy a východiska, na kterých je tato práce postavena. Část 2.1 se věnuje problematice evolučních algoritmů, část 2.2 popisuje základní způsoby přístupu k symbolické regresi. V části 2.3 je představena koevoluce prediktorů fitness jako metoda modelování fitness souběžně s řešením problému, v části 2.4 je koevoluce prediktorů fitness uvedena do souvislosti s řešením symbolické regrese.

Kapitola 3 se zabývá návrhem řešení symbolické regrese pomocí kartézského genetického programování. Jelikož se v práci porovnává přístup s užitím koevoluce a bez koevoluce, obsahuje tato kapitola návrh obou řešení. V kapitole 4 je popis implementace řešení symbolické regrese pomocí kartézského genetického programování s užitím koevoluce a bez koevoluce, včetně technologií použitých při implementaci.

Ověření funkčnosti implementovaných programů na zadaných úlohách je v kapitole 5. V následující kapitole 6 se zabývám nalezením optimálního nastavení evoluce kandidátních řešení a optimálního nastavení koevoluce. Tato kapitola také obsahuje popis skriptu pro dávkové testování řešení symbolické regrese, porovnání způsobů výpočtu fitness kandidátních řešení a popis vlastností koevoluce.

Souhrn výsledků a porovnání řešení s koevolucí a bez koevoluce je v kapitole 7. Nachází se zde i shrnutí chování koevoluce podle nastavených parametrů. V závěrečné kapitole 8 je celkový souhrn výsledků koevoluce a náměty na rozšíření práce.

Kapitola 2

Základní pojmy a principy

2.1 Evoluční algoritmy a genetické programování

Řada úloh umělé inteligence se řeší pomocí technik hledání v prohledávacím prostoru, který reprezentuje všechna nebo předem povolená řešení dané úlohy. Pokud je tento prostor dostatečně malý vzhledem k dostupné výpočetní síle, je možné prozkoumat všechna možná řešení a najít globální optimum. Rozsáhlé prostory jsou potom prohledávány různými heuristickými stochastickými algoritmy. Patří mezi ně i evoluční algoritmy – genetické algoritmy, evoluční strategie, evoluční programování a genetické programování, kterým se budu v této práci nejvíce zabývat. Výzkumníci, kteří prováděli výzkum v těchto oblastech odděleně, spolu začali spolupracovat až začátkem 90. let 20. století, kdy vznikl termín evoluční algoritmus.

Evoluční algoritmy jsou inspirovány základními principy Darwinovy teorie evoluce a neodarwinismu – fylogenezí. Evoluční proces je umožněn schopností reprodukce genetického programu. Tato reprodukce vykazuje jen velmi malou chybovost, což zaručuje, že se potomci liší od rodičů jen velmi málo. Občasné mutace zajišťují vznik nového genetického materiálu, který je nezbytný pro přežití druhů, jejich adaptaci v proměnném prostředí a vznik nových druhů. Evoluční algoritmy nacházejí uplatnění pro praktické řešení složitých problémů v oblastech umělé inteligence, inženýrského návrhu a optimalizací. Základní pojmy a principy uvedené v následujících částech této kapitoly jsou čerpány z publikací [4], [7], [10], [11], [12], [13] a [14].

Základní pojmy v evolučních algoritmech

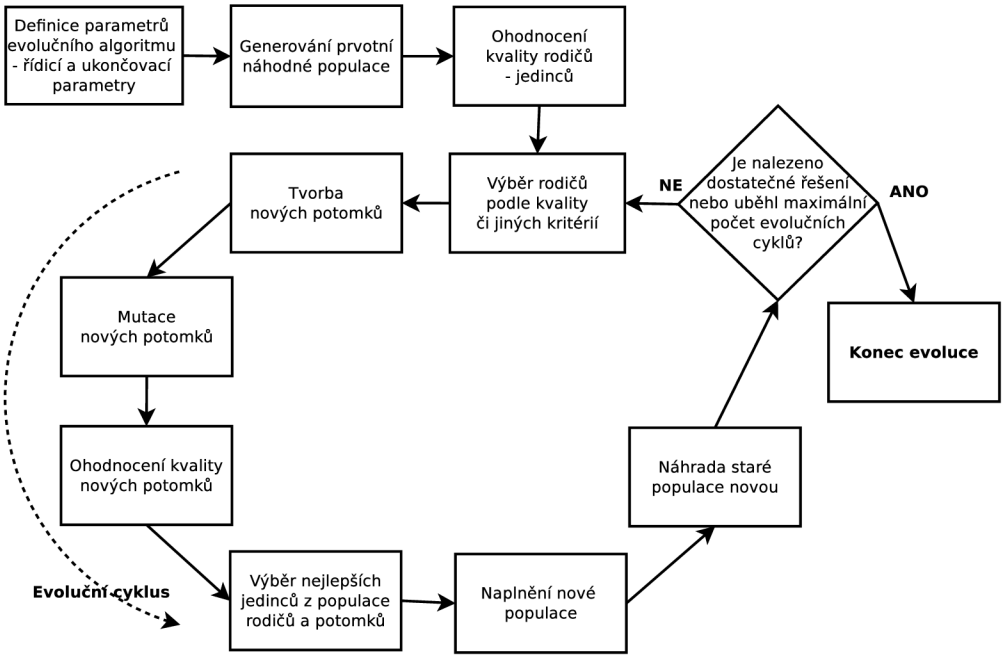
Gen: Je to základní stavební jednotka chromozomu. Jeho hodnota (tzv. alela) patří do definované abecedy (binární čísla, celá čísla, atp.).

Chromozom: Obvykle lineární pole genů, které může mít proměnnou velikost. Reprezentuje jedno řešení ve stavovém prostoru (také jedinec nebo individuum).

Genotyp: Představuje kódovaný tvar řešení (pomocí chromozomu).

Fenotyp: Kandidátní řešení problému, které je sestaveno podle obsahu chromozomu – například cesta v grafu.

Populace: Množina kandidátních řešení, která obsahuje obvykle konstantní počet jedinců.



Obrázek 2.1: Obecný cyklus evolučního algoritmu.

Fitness funkce: Funkce, která určí vhodnost jedince – daného kandidátního řešení. Při řešení se pak hledá její maximum (resp. minimum, např. nejkratší cesta v grafu mezi dvěma uzly).

2.1.1 Obecný evoluční algoritmus

Pro každý algoritmus musí být definovány parametry, které řídí běh algoritmu, nebo jej regulérně ukončí. Možné řešení problému je reprezentováno jedincem a jeho chromozomem. Na začátku musíme definovat podobu chromozomu – může to být například zakódovaný průchod grafem jako posloupnost vrcholů. Důležitou součástí algoritmu je i fitness funkce, která každého jedince ohodnotí. Dále definujeme velikost populace, ukončující podmínky – maximální počet generací nebo přípustnou hodnotu fitness. Když této nebo lepší fitness hodnoty dosáhne některý jedinec, je označen jako postačující řešení – evoluce nemusí pokračovat, pokud již bylo nalezeno řešení.

Počáteční generace se vytváří buď náhodně nebo s užitím již známých řešení problému. Na jedince se aplikují genetické operátory křížení, případně mutace, a nová populace vzniká v závislosti na vhodnosti jedinců.

Křížení jedinců je obvykle implementováno jako prohození částí chromozomů rodičů, přičemž vzniknou dva potomci. Nejčastěji se pak užívá křížení jednobodové, ale použít se dají i křížení vícebodová. Dalším genetickým operátorem je mutace. Mutace jedince se implementuje jako nahrazení části jeho chromozomu náhodně generovanou posloupností – zajišťuje tak diverzitu populace a snižuje riziko uváznutí v lokálním extrému fitness funkce.

Algoritmus evoluce tedy běží v cyklu, v jehož průběhu se vytváří populace potomků z populace rodičů, až dosáhne maximálního počtu generací nebo najde optimální řešení – hodnota fitness funkce některého jedince (kandidátního řešení) se dostane do intervalu správných řešení. Tento cyklus ilustruje obrázek 2.1.

Ohodnocení jedinců populace

Ohodnocení jedinců populace je obvykle časově nejnáročnější částí evolučního algoritmu. Ohodnocení pomocí fitness funkce se provádí nad všemi jedinci v populaci (kromě již ohodnocených jedinců, pokud se takoví v populaci nacházejí). Výpočet fitness funkce je obvykle realizován samostatným podprogramem. Optimalizaci výpočtu fitness funkce se věnuje velká pozornost a i v této práci se věnuji právě optimalizační metodě, která má za úkol snížit výpočetní složitost výpočtů fitness funkce (podkapitoly 2.3.1, 2.3.2).

Selekce

Selekce neboli výběr rodičů pro křížení je jedním z klíčových okamžiků pro běh genetických algoritmů. Selekce může probíhat mezi dvěma jedinci nebo mezi skupinami jedinců. Při selekci jsou upřednostňování jedinci s vyšší fitness hodnotou. Rozlišujeme čtyři základní typy selekce:

Ruletový výběr: Pravděpodobnost výběru jedince závisí přímo úměrně na jeho kvalitě. Fitness hodnoty jedinců se přepočítají tak, že jsou větší než 0, jejich součet je dohromady 1 a přitom jejich poměr zůstane stejný (fitness hodnoty všech jedinců se vynásobí konstantou, která zaručí, že součet fitness hodnot všech jedinců bude roven 1). Každému jedinci přímo úměrně jeho kvalitě přísluší patřičný úsek (interval) na jednotkové úsečce (kruhové výseči rulety). Náhodně vygenerované číslo na intervalu $\langle 0, 1\rangle$ pak určuje, který jedinec se zúčastní reprodukce.

Turnajový výběr: Náhodně je vybráno k jedinců (obvykle $k = 2$). Nejlepší z nich (s nejvyšší hodnotou fitness) je pak vybrán jako rodič.

Pravděpodobnostní výběr: Využívá dvou předchozích selekcí – nejprve jsou dva jedinci vybráni pomocí ruletového výběru, následně se na ně aplikuje turnajová selekce.

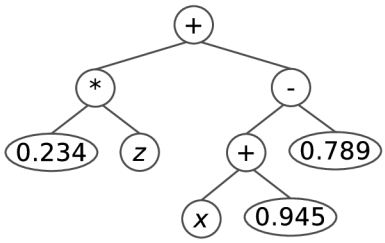
Uspořádaný výběr: Lineární výběr určitého počtu jedinců z populace vyžaduje setříděnou populaci tak, aby nejhorší jedinec měl index 1 a nejlepší index N. Součet indexů všech jedinců je pak $\frac{1}{2}\left(N + N^2\right)$. Náhodně vygenerované číslo na intervalu $\left\langle 1, \frac{1}{2}\left(N + N^2\right)\right\rangle$ pak určuje, který jedinec se zúčastní reprodukce.

2.1.2 Genetické programování

Jak uvádí [14], genetické programování (GP) bylo představeno v 80. letech 20. století Johnem Kozou. Koza navrhl modifikaci genetických algoritmů pro tvorbu programů v rámci symbolické regrese. Cílem není pouze evolučně hledat optimální hodnoty parametrů, které jsou zakódovány v chromozomu, ale automaticky generovat celé programy v určitém programovacím jazyce.

Genetické programování vytváří programy ve formě syntaktických stromů. Tyto struktury mají typicky proměnnou délku. Genetické operátory pracují nad spustitelnými strukturami. Kromě operátorů křížení a mutace existují i pokročilé operátory umožňující generování programů s podprogramy, moduly, atp. V rámci zjištění fitness hodnoty je proveden kód kandidátního programu pro definovanou množinu vstupů a jsou vyhodnoceny získané výsledky.

Kandidátní program může být reprezentován jako syntaktický strom (obrázek 2.2), vývojový diagram nebo v jazyce symbolických instrukcí. Množina terminálů reprezentuje



Obrázek 2.2: Reprezentace kandidátního řešení ve stromovém zakódování.

vstupy do programu, který je vyvíjen pomocí GP, konstanty a funkce bez argumentů s vedlejším účinkem (ve stromové reprezentaci tvoří listy). Množina funkcí může být specifická pro oblast, ve které je GP použito, nebo může být obecná. Je nutné používat *chráněné* varianty některých funkcí, protože některé funkce nemusí být definované pro určité hodnoty. Například při dělení nelze mít dělitel roven nule. Pokud tato situace nastane, musí funkce vrátit nějakou bezpečnou hodnotu (např. hodnotu 1). Používané funkce jsou obvykle standardní aritmetické a logické funkce, konstrukce známé z programovacích jazyků jako cykly, skoky, atp., a funkce z konkrétní aplikační oblasti.

Metody vytváření počáteční populace

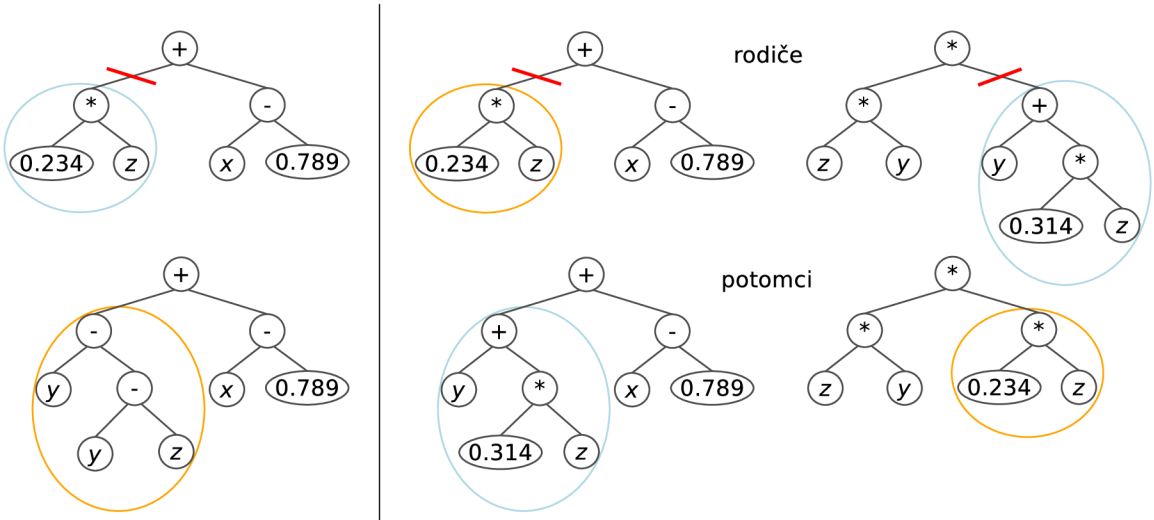
Grow: Při použití této metody se uzly náhodně vybírají z množiny funkcí i terminálů. Jestliže je vybrán terminál, znamená to ukončení dané větve. Tímto způsobem vznikají stromy nepravidelného tvaru.

Full: V této metodě se vybírají prvky pouze z množiny funkcí, dokud není dosaženo maximální možné hloubky stromu. Následně se vyberou terminály, jimiž se větve ukončí.

Ramped Half-and-Half: Na rozdíl od předchozích dvou metod, tato metoda generuje velikostně rozmanité stromy. Na polovinu stromů se použije metoda grow a na druhou polovinu metoda full a to tak, že pokud je např. maximální hloubka stromu 6, pak generuje stromy o maximálních hloubkách 2, 3, 4, 5 a 6.

Genetické operátory

Po náhodném vytvoření populace se pro získání nové populace postupuje stejně jako u genetických algoritmů. Nejprve nastává křížení – zvolí se místo, kde se jednotlivé stromy rozdělí, prohodí se mezi sebou podstromy, čímž vzniknou noví potomci (obrázek 2.3b). Následuje mutace, při níž se určitá část jedince (podstrom) nahradí náhodně vygenerovanou strukturou (podstromem, obrázek 2.3a). Pro aplikaci genetických operátorů se používají parametry jako pravděpodobnost mutace, pravděpodobnost křížení a navíc oproti genetickým algoritmům i parametr pravděpodobnost výběru uzlu pro mutaci. U genetického programování tohoto typu se objevuje problém, že potomci mohou mít větší reprezentace než jejich rodiče. Problém, kdy roste velikost jedinců a jejich fitness se nezlepšuje, se označuje jako *bloat*. V [14] je uvedeno, že velikost jedince obvykle narůstá s počtem iterací lineárně. Možným řešením je penalizace dlouhých řešení ve fitness, uzpůsobení genetických operátorů, atp.



(a) Mutace. (b) Křížení.

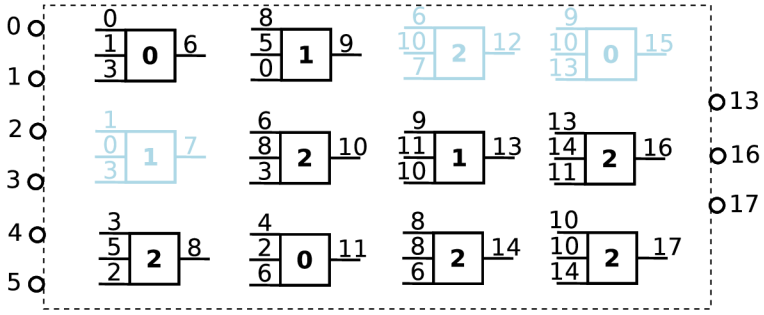
Obrázek 2.3: Stromové genetické programování – genetické operátory.

2.1.3 Kartézské genetické programování

Kartézské genetické programování (CGP) je variantou genetického programování a bylo poprvé uceleně popsáno roku 2000 [6]. Kandidátní řešení je zde reprezentováno pomocí orientovaného acyklického grafu, který je zakódován jako lineární pole celých čísel. Vstupy (terminály) a výstupy uzlů jsou postupně očíslovány. Funkce, které mohou vykonávat uzly, jsou také postupně očíslovány. Genotyp je vyjádřen jako seznam propojení uzlů a jejich funkce. Genotyp se následně mapuje na graf, který reprezentuje vykonání programu.

Na rozdíl od stromového genetického programování CGP nepoužívá tak rozsáhlé populace kandidátních řešení, ani křížení jako primární genetický operátor pro získaní nových kandidátních řešení – CGP používá pouze mutaci. V CGP má genotyp kandidátního řešení stejnou velikost pro všechny jedince v populaci, ale velikost fenotypu se mezi jedinci liší podle toho, kolik aktivních uzlů je na cestě mezi vstupy a výstupy kandidátního programu. Výstup jednoho uzlu je možné připojit na vstupy více uzlů, což stromová hierarchie neumožňuje.

Graf reprezentující genotyp G (kartézský program) je modelován jako pole uzlů umístěných v mřížce o velikost $n_c \times n_r$ (počet řádků $\times$ počet sloupců). Každý z uzlů může mít až n_n vstupních argumentů, nad nimiž realizuje jednu z n_f funkcí z množiny dostupných funkcí F, a jeden výstup. Dále kartézský program obsahuje počet primárních vstupů n_i, počet primárních výstupů n_o a parametr l-back l. Parametr l-back určuje kolik předchozích sloupců (výstupy uzlů z předchozích sloupců) lze propojit s uzlem v aktuálním sloupci. Primární vstup může být připojen ke vstupu uzlu v jakémkoliv sloupci, stejně tak i primární výstup k výstupu uzlu v jakémkoliv sloupci. Kartézský program je tedy definován pomocí devíti parametrů $G, n_i, n_o, n_n, F, n_f, n_r, n_c, l$. Na obrázku 2.4 je předvedeno zakódování kartézského programu (kandidátního řešení). Propojení vstupů uzlu je povoleno pouze s výstupy uzlů v předchozích sloupcích – parametr l-back může nabývat hodnot 1 až n_c (počet sloupců).



Obrázek 2.4: Reprezentace kandidátního řešení s nastavením: 6 primárních vstupů, 3 primární výstupy, 3 vstupy uzlu, 3 funkce uzlu, 3 řádky, 4 sloupce, l-back=2, pro chromozom: 0 1 3 **0** 1 0 4 1 3 5 2 **2** 8 5 0 **1** 6 8 3 **2** 4 2 6 **0** 6 1 0 7 2 9 11 10 **1** 8 8 6 **2** 9 10 13 0 13 14 11 **2** 10 10 14 **2** 13 16 17. Funkce uzlu jsou uvedeny tučně a $F = \{\text{and}_\mathbf{0}, \text{nor}_\mathbf{1}\, \text{mux}_\mathbf{2}\}$.

Zakódování

Chromozom popisující graf sestává z $n_r n_c (n_n + 1) + n_o$ celočíselných hodnot. Každému primárnímu vstupu se přiřadí index z intervalu $\langle 0, n_i - 1 \rangle$, výstupům uzlů se přiřadí indexy počínaje levým horním uzlem $\langle n_i, n_i + n_c n_r - 1 \rangle$ a funkcím, které uzel reprezentuje, se přiřadí indexy z intervalu $\langle 0, n_f - 1 \rangle$. Každý uzel je kódován pomocí $n_n + 1$ celočíselných hodnot – prvních n_n hodnot určuje, ke kterým primárním vstupům nebo výstupům uzlů je připojeno n_n vstupů uzlu, následující hodnota pak určuje funkci, kterou uzel provede nad vstupy a předá na výstup. Na konci chromozomu je n_o hodnot, které značí indexy uzlů, ke kterým jsou připojeny primární výstupy. Tento způsob kódování způsobuje, že:

- Některé uzly nemusí být ve fenotypu použity.
- Některé uzly mohou být použity vícenásobně.
- Některé vstupy uzlů nemusí být použity.
- Některé primární vstupy nemusí být ve fenotypu použity.

Algoritmus

Jediným genetickým operátorem, který CGP využívá, je mutace. Při mutaci se náhodně vybere gen v chromozomu jedince a nahradí se náhodně vygenerovanou hodnotou, která splňuje kritéria – pro gen na konkrétní pozici existuje množina přípustných hodnot. Při mutaci se může změnit funkce uzlu nebo index připojeného vstupu a tím se může změnit i topologie propojení grafu.

Na počátku algoritmu evoluce CGP se náhodně vygeneruje počáteční populace $\lambda + 1$ jedinců. Evoluce pak běží v cyklu, dokud není nalezeno dostatečně kvalitní řešení nebo není dosaženo maximálního počtu populací.

- Ohodnotí se všichni jedinci z populace pomocí fitness funkce.
- Vybere se nejlépe ohodnocený jedinec v populaci. Kontroluje se, zda vybraný jedinec byl rodičem předchozí populace. Pokud ano a v populaci existuje i jiný stejně kvalitní jedinec (má stejnou fitness hodnotu), pak se jako rodič následující generace vybere ten, který nebyl rodičem aktuální populace.

- Vygeneruje se λ jedinců pomocí mutace z nejlepšího jedince.
- Nejlepší jedinec spolu se svými λ potomky tvoří novou populaci.

Vyhodnocení fitness

K vyhodnocení fitness je zapotřebí projít všechna data z trénovací množiny a porovnat referenční výstupy s výstupy kandidátního programu. K vyhodnocení výstupů kandidátního jedince lze přistupovat jedním ze tří způsobů. Prvním způsobem je vyhodnocení všech uzlů pro trénovací vstupy a na závěr předání výstupů – provádí se funkce i u uzlů, které nejsou ve fenotypu. Dalším způsobem může být rekurzivní sestup po fenotypu jedince, od výstupů kandidátního programu po vstupy podobně jako u rekurzivního sestupu syntaktickým stromem. V tomto případě se nevyhodnocují uzly, které nejsou zapojeny do fenotypu, ale mohou se tytéž uzly vyhodnotit vícekrát. Optimálním přístupem je zřejmě kombinace obou předchozích přístupů. Jelikož v trénovací množině bývá mnoho trénovacích konfigurací programu (vstupy a jim odpovídající výstupy), tak se rekurzivním sestupem označí aktivní uzly (obsažené ve fenotypu) a následné vyhodnocování funkcí probíhá pouze nad těmito uzly.

2.1.4 Genetické programování a fitness funkce

Ve většině případech je pro ohodnocování kandidátních řešení použita trénovací množina. Výstupy získané kandidátním programem se porovnají s požadovanými hodnotami – např. může být vypočtena průměrná odchylka mezi získanými a požadovanými hodnotami, která je pak fitness hodnotou.

Hrubá fitness: Udává se v hodnotách přirozených pro daný problém.

Standardizovaná fitness: Převádí hrubou fitness tak, že fitness hodnota je kladné číslo a čím více se blíží k nule, tím kvalitnější kandidátní řešení je.

Přizpůsobená fitness: Jedná se o nejpoužívanější druh fitness. Vypočítá se jako převrácená hodnota součtu jedničky se standardizovanou fitness – tím získáme fitness hodnoty z intervalu $\langle 0, 1\rangle$ (1 pro nejlepšího jedince – standardizovaná hodnota je 0).

Normalizovaná fitness: Vypočítá se jako podíl hrubé fitness jedince a sumy hrubých fitness všech jedinců populace. Hodnota fitness pak leží v intervalu $(0, 1\rangle$, lepší jedinec je ohodnocen vyšší hodnotou než horší a suma normalizované fitness celé populace je rovna jedné.

2.2 Symbolická regrese

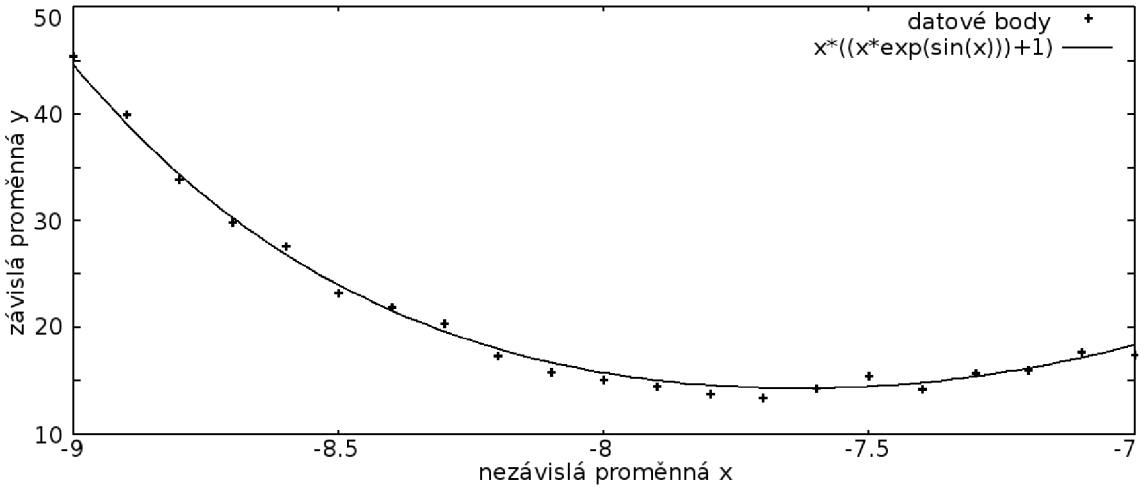
Symbolická regrese je považována za základní úlohu strojového učení. Zabývá se identifikací matematického popisu skryté závislosti experimentálně získaných dat.

Trénovací data v úloze symbolické regrese se typicky sestávají až z desíti tisíců bodů (experimentálně získaných) s předpokladem, že výstupní hodnota závisí na vstupní (závislost měřených hodnot lze vyjádřit funkcí). V tabulce 2.1 je příklad takových dat a na obrázku 2.5 jejich zobrazení.

Již po staletí se vědci snaží nalézt přírodní zákony. Jak je uvedeno v [8], symbolická regrese umožňuje nalézt tyto zákony bez znalostí fyziky, kinematiky, geometrie, atp., na

Nezávislá proměnná x	-9.00	-8.90	-8.80	-8.70	-8.60	...	-7.10	-7.00
Závislá proměnná y	45.45	39.99	33.84	29.87	27.67	...	17.74	17.43

Tabulka 2.1: Příklad trénovacích dat.



Obrázek 2.5: Zobrazení bodů z tabulky 2.1 a jejich proložení.

základě zpracování experimentálně získaných dat. Symbolická regrese se využívá pro popis systémů, které obsahují mezery navzdory velkému množství dat z pozorování, od biologie až po kosmologii.

Jednou z možností výpočtu fitness kandidátních řešení je, že se spočítání střední absolutní odchylky datových bodů, které jsou v trénovací množině dat, a které generuje kandidátní řešení následovně:

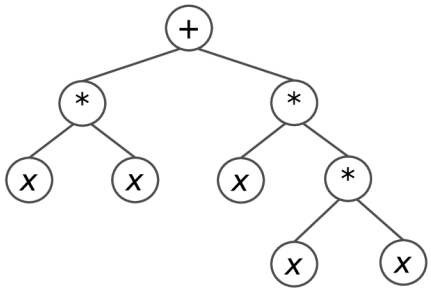
$$\mathrm{SF}(s) \quad = \quad \frac{1}{n}\sum_{i=1}^{n} \left| s\left(x_i\right) - y_i \right|, \tag{2.1}$$

kde s je kandidátní řešení (algebraický výraz), x_i je trénovací vstup funkce a y_i je výstup funkce z trénovací množiny, n je počet vzorků v trénovací množině dat. Čím menší má kandidátní řešení hodnotu fitness, tím je kvalitnější.

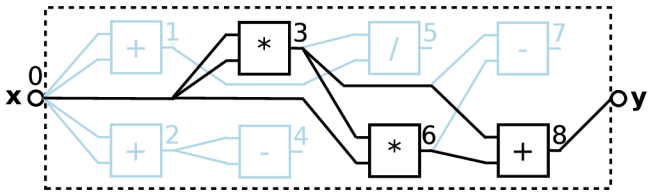
Druhým způsobem, jak určit fitness kandidátního řešení, je pomocí přidělování bodů (skóre, uvedeno v [3]) za správný výsledek kandidátního řešení pro daný datový bod. Jestliže výsledek řešení pro daný datový bod odpovídá funkční hodnotě datového bodu s definovanou chybou σ, tak se k fitness hodnotě přičte 1, v opačném případě 0. Čím větší má kandidátní řešení hodnotu fitness, tím je kvalitnější.

$$\mathrm{SF}(s) \quad = \quad \sum_{i=1}^{n} g\left(s\left(x_i\right)\right), \text{ kde} \tag{2.2}$$

$$g\left(s\left(x_i\right)\right) \quad = \quad \begin{cases} 0 & \text{pro } \left|s\left(x_i\right) - y_i\right| \geq \sigma \\ 1 & \text{pro } \left|s\left(x_i\right) - y_i\right| < \sigma \end{cases} \tag{2.3}$$



(a) Fenotyp $x^2 + x^3$ ve stromové reprezentaci.



(b) Fenotyp $x^2 + x^3$ v CGP.

Obrázek 2.6: Reprezentace kandidátních řešení pro symbolickou regresi.

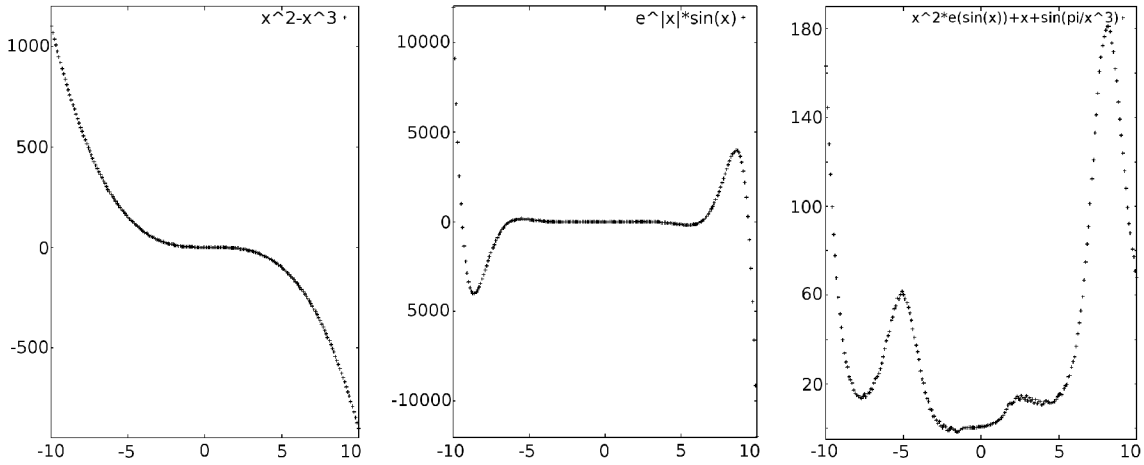
σ je povolená odchylka řešení. Čím větší má kandidátní řešení hodnotu fitness, tím je kvalitnější. Bodový systém může fungovat i jako přidělování trestných bodů – za chybný výpočet datového bodu se přidělí trestný bod – pak se pro správné řešení hledá fitness hodnota blížící se nule.

Zakódování kandidátního řešení

Kandidátním řešením v úloze symbolické regrese je funkce, která pro zadaný vstup přiřadí výstup na základě vytvořeného stromu nebo acyklického grafu. V případě funkce jedné proměnné, jak je uvedeno vztahem 2.4 i 2.6, máme tedy pro stromové GP jeden terminál (x) a pro CGP nám stačí jeden primární vstup (x) a jeden primární výstup (y). V případě funkce více proměnných, jak uvádí vztah 2.5 – funkce dvou proměnných, potřebujeme dva terminály x, z pro stromové GP a dva primární vstupy x a z pro CGP, primární výstup však zůstává pouze jeden.

$$y = f(x) \tag{2.4}$$
$$y = f(x, z) \tag{2.5}$$
$$y_n = f(y_{n-1}) \tag{2.6}$$

Jelikož většina aritmetických operátorů je binárních, postačí nám dva vstupy uzlu pro CGP a pro stromovou reprezentaci binární strom. Unární operace (například unární minus, faktoriál, či absolutní hodnota) je pak provedena pouze nad prvním vstupem. Příklad fenotypu reprezentujícího funkci $y = x^2 + x^3$ pro stromovou reprezentaci je na obrázku 2.6a a pro CGP na obrázku 2.6b.



(a) Průběh funkce 2.7. (b) Průběh funkce 2.8. (c) Průběh funkce 2.9.

Obrázek 2.7: Grafy hledaných vztahů. Na horizontální ose jsou zobrazeny hodnoty x, na vertikální funkční hodnoty $f(x)$.

2.2.1 Testovací úlohy

Jako trénovací množiny je vhodné vytvořit množiny datových bodů reprezentujících funkce s různou složitostí průběhu. Vzorkováním vybraných funkcí získáme data, na kterých potom můžeme testovat kandidátní funkce. V článku [9] používají pro trénování navzorkované funkce 2.7, 2.8, 2.9, jejichž průběhy jsou zobrazeny na obrázcích 2.7a, 2.7b, 2.7c.

$$f(x) = x^2 - x^3 \tag{2.7}$$
$$f(x) = e^{|x|}\sin(x) \tag{2.8}$$
$$f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right) \tag{2.9}$$

2.3 Koevoluce

Podle Janzena [2] můžeme koevoluci definovat jako evoluční změny vlastností jedinců v jedné populaci jako odpověď na vlastnosti jedinců v populaci druhé. Koevoluce druhů může nastat, jestliže jeden druh selektivně utlačuje jiný. Kořisti se vyvíjí vlastnosti, díky kterým může predátorovi snadněji uniknout nebo se bránit, a vlastnosti predátora se mění tak, aby kořist dokázal získat.

Tohoto principu můžeme využít v genetickém programování jako prostředek pro optimalizaci řešení úlohy. Jak uvádí Schmidt a Lipson [9], v koevolučních algoritmech se míra fitness jednoho jedince stává funkcí jiného jedince. Přesněji řečeno, jeden jedinec může ovlivnit relativní výpočet fitness funkce mezi jinými jedinci ve stejné nebo oddělené populaci. Upřesňování fitness funkce a vývoj jedinců mohou změnit průběh evoluce.

Koevoluce je často použita v úlohách, kde nelze jednoznačně předem určit fitness funkci. Můžeme chtít najít řešení, které závisí na jiném řešení. Ze soutěže mezi jedinci založené na koevoluci a zdokonalování úspěšných jedinců pak vyplyne dominantní řešení.

Vyhodnocování fitness funkce jedinců je celkově výpočetně velmi náročné. Souběžný vývoj fitness prediktorů může snížit frekvenci a náročnost výpočtu fitness. Populace kandidátních řešení se vylepšuje pomocí populace fitness prediktorů a zároveň dochází ke snižování výpočetní náročnosti. Populaci fitness prediktorů vylepšuje populace kandidátních řešení pomocí zmenšování intervalu množiny nejvhodnějších fitness hodnot. Koevolucí fitness prediktorů se zabývá část 2.3.2.

2.3.1 Modelování fitness funkce

Modelování fitness funkce je zkoumanou oblastí v evolucí inspirovaném počítání s různými přístupy a výsledky. Je hned několik důvodů pro použití modelování aproximace fitness funkce:

- snížení výpočetní složitosti výpočtů fitness funkce,
- potřeba řešit problém, kde fitness nelze jednoznačně předem určit,
- fitness funkce je hrubá a neumožňuje tak stabilní výpočty,
- řešení často uvázne v lokálním extrému,
- jsou generováni rozdílní jedinci se stejnou fitness.

Metody modelování fitness funkce spadají do oblasti strojového učení a proto se dají využít různé techniky strojového učení jako neuronové sítě, SVM (support vector machine), rozhodovací stromy, bayesovská síť, atp., pro klasifikaci jedinců za účelem vytvoření modelu. Přesnějších modelů pak dosahují moderní metody jako boosting, bagging nebo ensemble learning. Bohužel nelze předem určit, která z metod bude pro řešení konkrétního problému nejefektivnější.

Evoluční techniky vytváření modelu zahrnují dědičnost fitness, napodobení fitness a částečnou evoluci. Použití modelu aproximace fitness funkce s sebou nese i tyto problémy:

Trénování modelu: Fitness modely jako neuronová síť nebo SVM jsou velmi robustní. Při použití pokročilejších metod jako boosting nebo bagging dosáhneme ještě významného rozšíření. Je třeba přesně spočítat velké množství fitness, abychom byli schopni efektivně učit jakýkoliv typ modelu.

Při použití koevoluce můžeme model fitness učit paralelně s řešením problému. V počátcích evoluce tak můžeme použít jen velmi hrubé fitness. V průběhu evoluce se pak vytváří i model fitness.

Úroveň aproximace: Když se vytváří fitness model samostatně, vyžaduje to značnou obecnost řešení, aby model zahrnoval všechna možná řešení z prostoru kandidátních řešení s různou fitness.

Vytváření modelu fitness při koevoluci musí pracovat jen s jedinci v současné populaci. Model proto nemusí pracovat s celým prohledávacím prostorem, ale zahrnovat jen jeho podmnožinu a tedy může být mnohem méně komplexní.

Pokles přesnosti: Ve většině aplikací je použití aproximace fitness na úkor přesnosti fitness. V nejhorším případě se může globální optimum úplně vytratit z prostoru, který pokrývá fitness.

2.3.2 Koevoluce fitness prediktorů

Konvenční evoluční algoritmus se snaží nalézt optimální řešení daného problému. V tomto smyslu je optimální řešení s^* definováno jako:

$$s^* = \arg\max_{s \in S} \text{fitness}(s), \tag{2.10}$$

kde S je je množina všech možných kandidátních řešení problému a fitness(s) je fitness hodnota aktuálního řešení s. V koevolučním algoritmu zaměníme všechny fitness s fitness prediktorem p. V tomto případě je optimální řešení hledáno vyhodnocením funkce prediktoru:

$$s^* = \arg\max_{s \in S} p(s), \tag{2.11}$$

kde p je použitý fitness prediktor.

Při koevoluci vyvíjíme v druhé populaci fitness prediktory, aby funkce p byla co nejpřesnější pro současnou populaci řešení. Ve třetí populaci jsou fitness trainers, které jsou použity pro výpočet, jak se blíží fitness prediktory požadované fitness. Fitness trainers se vybírají z populace řešení periodicky tak, že mají co nejvyšší vzájemnou odlišnost.

Článek [9] uvádí, že optimální řešení pro každou z populací s^* (populace řešení), t^* (populace fitness trainers), p^* (populace fitness prediktorů) je definováno následovně:

$$s^* = \arg\max_{s \in S} p_{\text{best}}(s), \tag{2.12}$$

$$t^* = \arg\max_{s \in S_c} \frac{1}{N} \sum_{p \in P_{cur}} \left(p\left(s\right) - \overline{p\left(s\right)} \right)^2, \tag{2.13}$$

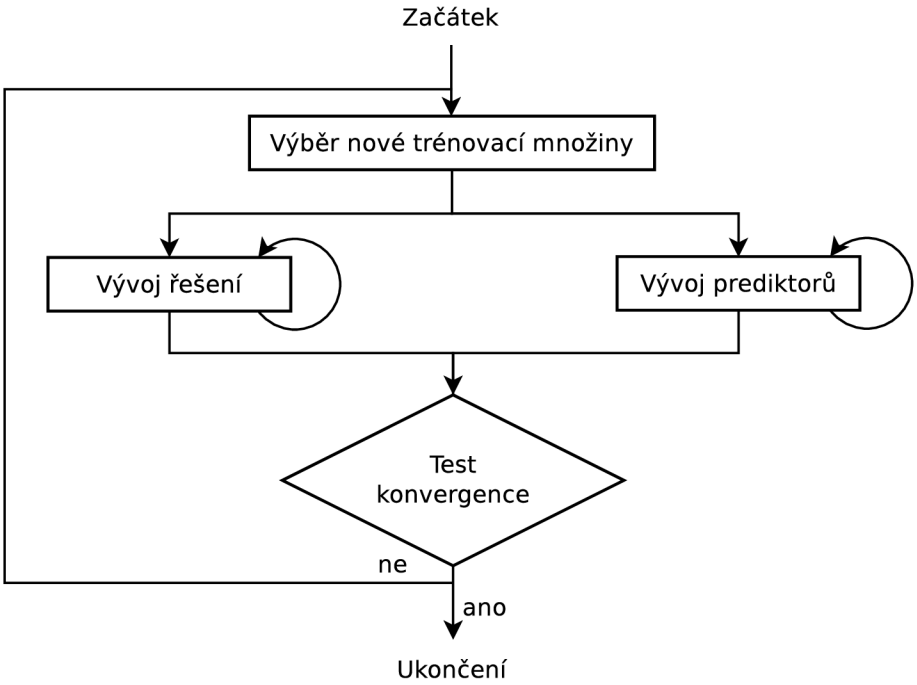
$$p^* = \arg\min_{p \in P} \frac{1}{N} \sum_{t \in T_{cur}} \left| \text{fitness}\left(t\right) - p\left(t\right) \right|, \tag{2.14}$$

kde S je množina všech možných řešení problému, S_c je současná populace řešení, P je množina všech možných fitness prediktorů, P_{cur} je současná populace prediktorů, T_{cur} je současná populace fitness trainers, p_{best} je nejlépe ohodnocený prediktor z P_{cur}, a $p(s)$ je průměrná predikovaná fitness řešení s mezi současnými prediktory. Všechny tři populace se vyvíjí paralelně a jejich výsledky se mění s každou generací.

Populace řešení se vyvíjí podle fitness současného nejlepšího fitness prediktoru. Fitness trainers jsou vybraná řešení z populace řešení, které produkují vetší variabilitu v predikcích mezi populací prediktorů. Populace fitness prediktorů se vyvíjí tak, aby minimalizovala rozdíl mezi pravou fitness funkcí a predikovanými fitness současnou populací.

Algoritmus

Algoritmus prezentovaný v [9] využívá tři populace: populace řešení problémů, populace fitness prediktorů, populace fitness trainers. Princip algoritmu je zobrazen na obrázku 2.8. Na začátku koevoluce se náhodně vygeneruje populace kandidátních řešení a populace prediktorů. Algoritmus vybere jedince z populace řešení pro ohodnocení skutečnou fitness funkcí pro trénování fitness prediktorů. Pak algoritmus vyvíjí populaci řešení s použitím nejlépe ohodnoceného fitness prediktoru a vyvíjí populaci prediktorů s použitím fitness trainers. Nakonec se otestuje konvergence nejlépe ohodnoceného jedince, a pokud je splněna, algoritmus s úspěchem skončí.



Obrázek 2.8: Koevoluce kandidátních řešení a fitness prediktorů.

Výpočet skutečné fitness: Cílem tohoto kroku je vybrat jedince z populace řešení, aby pomohli optimalizovat fitness prediktory na základě současných řešení. Vyberou se jedinci, kteří mají rozdílné fitness hodnoty, aby se zajistily různé hodnoty predikovaných fitness v populaci prediktorů. V tomto algoritmu se uchovávají poslední trainers (kandidátní řešení se svojí fitness). Odstranění starých trainers může urychlit algoritmus, ale může vést k uváznutí v cyklu.

Vývoj populací: Kandidátní řešení a fitness prediktory jsou vyvíjeny paralelně užitím dvou vláken. Fitness trainers se vybírají periodicky ve vlákně obstarávajícím vývoj populace prediktorů.

Vlákno vývoje populace řešení začíná náhodným generováním počáteční populace kandidátních řešení. Hlavní smyčka pak vyvíjí tuto populaci. Nejlépe ohodnocený fitness prediktor je pak použit pro ohodnocení generace potomků kandidátních řešení. Nakonec je otestována konvergence nejlépe ohodnoceného jedince z populace řešení.

Vlákno vývoje populace prediktorů začíná náhodným generováním fitness prediktorů. Hlavní smyčka pak vyvíjí prediktory a periodicky přidává nové trainers do populace fitness trainers. Fitness každého prediktoru se počítá tak, že se spočítá průměr absolutních chyb mezi fitness predikcí a skutečnou fitness hodnotou každého fitness traineru.

Protože evoluce fitness prediktorů nepotřebuje tolik výpočetního výkonu jako evoluce řešení, běží mnohem rychleji. Pro redukci tohoto problému a zpomalení vývoje je ve smyčce vlákna vyvíjejícího populaci prediktorů přidáno čekání (delay).

Nové fitness trainers se z populace řešení vybírají periodicky. Fitness trainers jsou řešení, které optimalizují predikci fitness prediktorů.

Test konvergence: Test konvergence určuje, jestli algoritmus skončí, testováním nejlépe ohodnoceného kandidátní řešení. Pokud v tomto kroku řešení nekonvergovalo, přidají se nové trainers do populace trainers a koevoluce pokračuje. Jinak je vráceno nejlepší nalezené řešení a algoritmus se ukončí.

2.4 Koevoluce a symbolická regrese

Koevoluce trénovacích příkladů v úloze symbolické regrese je v literatuře popsána mnoha pracemi. Poslední výzkum využívá souběžné koevoluce trénovacích příkladů a využití odchylek a podobá se metodě strojového učení boosting. Velmi málo prací se však zabývá predikcí fitness nebo modelováním v symbolické regresi. V experimentech v článku [9] se koevoluje podmnožina trénovacích dat příkladů, které aproximují míru fitness (oproti kompletnímu použití trénovacích dat).

2.4.1 Fitness prediktor

Trénovací data v úloze symbolické regrese typicky sestávají ze stovek až tisíců bodů (zobrazení, experimentálně získaných) s předpokladem, že výstupní hodnota závisí na vstupní (závislost měřených hodnot lze vyjádřit funkcí). Při použití fitness prediktoru je prediktor fitness malou podmnožinou těchto bodů. Namísto výpočtu skutečné celkové fitness kandidátních řešení se určí imaginární fitness kandidátních řešení vypočtením odchylky pouze hrstky datových bodů, které jsou ve fitness prediktoru. Predikovaná fitness se spočítá následovně:

$$\mathrm{PSF}(s) = \frac{1}{n} \sum_{i=1}^{n} \left| s\left(x_i\right) - y_i \right|, \tag{2.15}$$

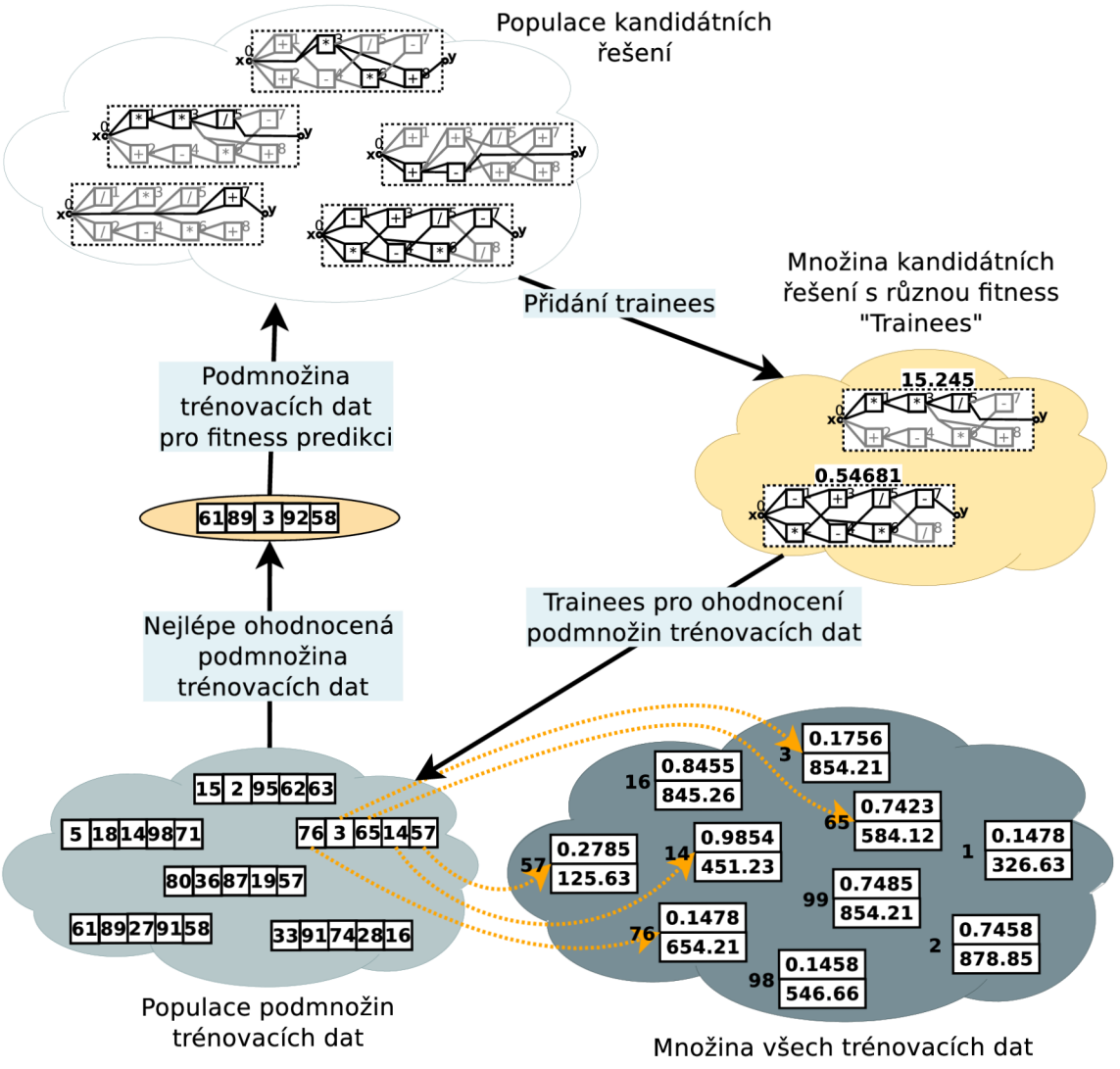
kde s je kandidátní řešení (algebraický výraz), x_i je trénovací vstup funkce a y_i výstup funkce z trénovací množiny, které jsou indexované prediktorem, n je počet vzorků v prediktoru.

V experimentech výše zmíněného článku se jako evoluční algoritmus využívá genetické programování. Vývoj fitness prediktorů se zajišťuje jednobodovým křížením tak, že se náhodně vybere bod křížení dvou rodičů. Mutace fitness prediktoru se pak provádí náhodnou změnou indexu.

2.4.2 Velikost fitness prediktoru

Mezi celkovým výkonem a velikostí prediktoru (podmnožiny trénovacích dat) je podstatná závislost. Použití malého počtu vzorků ve fitness prediktoru dovoluje využít více generací při zachování výpočetního výkonu za cenu snížení přesnosti predikce. Příkladem mohou být vzorky bodů hledané funkce $f(x) = e^{|x|} \sin(x)$. Tato funkce je nelineární a má dvě aproximační minima, která mohou výrazně ztížit hledání správného řešení. Konvergence řešení je pak lepší při použití většího počtu vzorků.

Když má fitness prediktor pouze dva vzorky, výpočet fitness má velmi nízkou váhu a evoluční proces vyžaduje mnohem více generací. Zatímco větší podmnožina vzorků je dostatečně přesná, počet potřebných generací se příliš neredukuje. Lze předpokládat, že pro funkce se složitějším průběhem bude potřeba více vzorků v prediktorech pro dosažení konvergence.



Obrázek 2.9: Populace pro koevoluci.

Kapitola 3

Návrh řešení symbolické regrese

V této kapitole se zabývám návrhem systému, který bude umožňovat řešení symbolické regrese pomocí kartézského genetického programování a koevoluce. Článek [9] pojednává o koevoluci prediktorů fitness a kandidátních řešení, jejichž evoluce je založena na principu stromového genetického programování. Tato práce se však zabývá návrhem a implementací evoluce kandidátních řešení založené na principu kartézského genetického programování.

Kartézský program je obecnější variantou orientovaného acyklického grafu než strom. Rozdíly v přístupu mezi stromovým genetickým programováním a kartézským genetickým programováním jsou popsány v kapitole 2.1. V této kapitole uvádím i několik odlišných přístupů ke koevoluci prediktorů fitness, než jak jsou uvedeny v článku o koevoluci prediktorů fitness [9]. Na základě článku [9] je v části 7.2 porovnána výkonnost koevoluce s CGP s koevolucí se stromovým GP.

Cílem této práce je porovnání výkonnosti řešení symbolické regrese s použitím koevoluce a bez ní. Bude tedy implementován program, který řeší symbolickou regresi bez použití koevoluce a pomocí experimentů bude nalezeno jeho nejvhodnější nastavení. Toto nastavení bude dále použito při experimentech s koevolucí, kde se budou měnit již jen parametry nastavení koevoluce.

Při návrhu programu je třeba řešit několik dílčích problémů. Jením z nich je paralelizace zpracování koevoluce. Tímto tématem se zabývám v části 3.1. Další části této kapitoly se pak zabývají návrhem kandidátního řešení a trainers 3.2 a prediktorů fitness 3.3. V části 3.3.1 jsou ukázány přístupy k ohodnocení prediktorů fitness. Část 3.4 pojednává o vyhodnocení fitness kandidátního řešení. Dále je popsána evoluce kandidátních řešení bez použití koevoluce 3.5 a s použitím koevoluce 3.6.

3.1 Návrh paralelizace

V článku [9] uvádí příklad paralelizace využitím dvou vláken. Jedno vlákno se stará o evoluci kandidátních řešení, druhé pak o evoluci prediktorů fitness, o nahrazování trainers a ohodnocení nových trainers skutečnou fitness. Máme zde tedy tři základní úlohy:

- evoluce kandidátních řešení,
- evoluce prediktorů fitness,
- vytvoření nových trainers a výpočet jejich skutečné fitness.

Tyto tři úlohy lze vykonávat sekvenčně, částečně paralelně nebo plně paralelně. V případě využití vláken, lze použít jedno, dvě nebo tři vlákna.

V případě sekvenčního přístupu probíhá evoluce kandidátních řešení, přičemž periodicky jednou za daný počet iterací evolučního cyklu kandidátních řešení dojde k jednomu iteračnímu kroku evoluce prediktorů. Rovněž jsou periodicky nahrazovány trainers.

Plně paralelní přístup odděluje všechny tři výše zmíněné úlohy. Vlákno obstarávající evoluci kandidátních řešení, které běží nejrychleji, periodicky posílá signál dalším dvěma vláknům, aby pokračovala v další iteraci svého cyklu.

V této práci jsem zvolila částečně paralelní přístup, avšak mírně odlišný od přístupu uvedeného v článku [9]. Pro koevoluci jsou použita dvě vlákna. První vlákno se stará o koevoluci kandidátních řešení a na základě svých výsledků vybírá nové trainers z populace kandidátních řešení. Část trainers je však v tomto vlákně generována náhodně, aby byla zajištěna různorodost fitness hodnot jedinců z množiny trainers. První vlákno periodicky posílá signál druhému vláknu, které na základě přijetí signálu provede další iteraci. Druhé vlákno se stará o evoluci prediktorů fitness. Pokud byly trainers změněny, načte je a ohodnotí je skutečnou fitness. Dále je používá pro ohodnocení jedinců z populace prediktorů.

Jelikož jsou použita dvě vlákna, je třeba použít synchronizační prostředky. K vyloučení vícenásobného přístupu ke sdíleným proměnným slouží binární semafory (*mutual exclusion*). Protože druhé vlákno pokračuje v práci na základě signálu od prvního vlákna, je rovněž třeba zařadit podmíněnou proměnnou (*condition variable*), která slouží pro pozastavení vlákna a čekání na splnění podmínky.

3.2 Kandidátní řešení

Chromozom kandidátního řešení sestává z uzlů, jejichž počet je roven součinu počtu řádků a počtu sloupců v mřížce genotypu a jednoho uzlu navíc – pro výstup. Každý z uzlů pak obsahuje tři hodnoty:

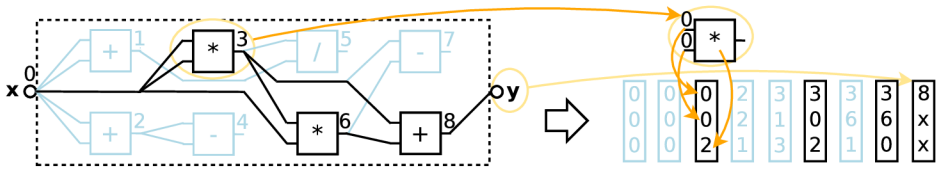
- index uzlu připojeného na první vstup uzlu,
- index uzlu připojeného na druhý vstup uzlu,
- číselné označení operace, kterou uzel nad vstupy vyčísluje.

U posledního uzlu se používá pouze hodnota indexu uzlu připojeného na první vstup uzlu. Tato hodnota označuje, ke kterému vnitřnímu uzlu je připojen primární výstup kandidátního řešení. Kódování chromozomu kandidátního řešení je znázorněno na obrázku 3.1.

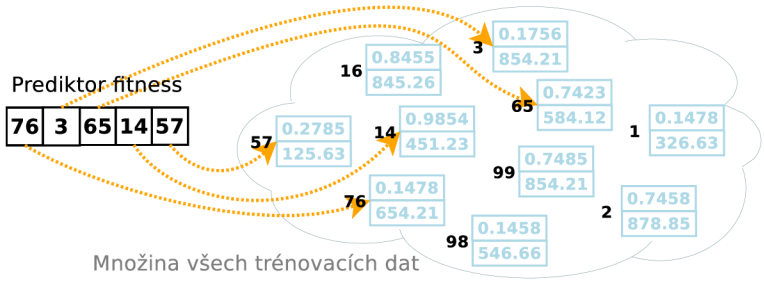
Jelikož fenotyp řešení neobsahuje všechny uzly genotypu, je vhodné ke každému uzlu v chromozomu přiřadit informaci o tom, zda je či není aktivní. Tato informace urychlí výpočet fitness, neboť se nemusí počítat funkční hodnota všech uzlů v genotypu, ale pouze těch obsažených ve fenotypu. V případě, že uzel je aktivní, je na tomto místě rozlišeno, zda provádí požadovanou funkci nad vstupy, nebo generuje konstantu. Generování konstant je popsáno v části 3.4.3.

3.2.1 Trainers

Trainers je množina kandidátních řešení ohodnocená skutečnou fitness, která slouží pro ohodnocení fitness prediktorů. Vybrané trainers musí obsahovat kandidátní řešení s různou fitness, aby bylo možné prediktory co nejpřesněji ohodnotit. Z toho důvodu dělím trainers na dvě poloviny. Jedna polovina je plněna při evoluci kandidátních řešení nejlepšími vždy různými kandidátními řešeními. V druhé polovině se obměňují náhodně generované



Obrázek 3.1: Chromozom kandidátního řešení.



Obrázek 3.2: Chromozom prediktoru fitness.

trainers pro zajištění rozmanitosti množiny trainers. Jedná se o množinu sdílenou evolucí kandidátních řešení s evolucí prediktorů fitness.

3.3 Prediktory fitness

Jak je uvedeno v části 2.4.1, fitness prediktor je podmnožinou všech datových bodů z trénovací množiny (zobrazeno na obrázku 3.2). Chromozom fitness prediktoru obsahuje určitý počet ukazatelů na datové body (optimálním počtem se zabývám v kapitole 7). Struktura datových bodů je blíže popsána v implementační části 4.2.

3.3.1 Ohodnocení prediktoru

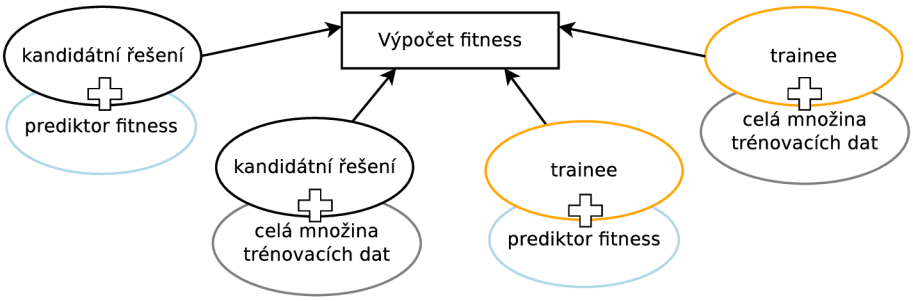
K ohodnocení prediktoru lze přistupovat podobným způsobem jako při ohodnocení kandidátního řešení v úloze symbolické regrese.

V prvním případě lze spočítat absolutní odchylku řešení ohodnocených skutečnou fitness (ohodnocení dané vztahem 2.1) a řešení ohodnocených prediktorem (ohodnocení dané vztahem 2.15) následovně:

$$\mathrm{PF}(p) \quad = \quad \frac{1}{o}\sum_{j=1}^{o} \left|\mathrm{SF}(s_j) - \mathrm{PSF}(s_j)\right|, \tag{3.1}$$

kde PF je fitness prediktoru p, o je počet trainers, $\mathrm{SF}(s_j)$ je skutečná fitness řešení s_j a $\mathrm{PSF}(s_j)$ je fitness řešení s_j predikovaná prediktorem p.

Druhým způsobem, jak určit fitness prediktoru, je pomocí přidělování bodů (skóre) za výsledek predikované fitness (ohodnocení dané vztahem 2.15) odpovídající skutečné fitness



Obrázek 3.3: Varianty ohodnocení fitness.

kandidátního řešení (ohodnocení dané vztahem 2.1) – s povolenou odchylkou.

$$\mathrm{PF}(p) = \sum_{i=1}^{o} g\left(s_j\right), \text{ kde} \tag{3.2}$$

$$g\left(s_j\right) = \begin{cases} 0 & \text{pro } |\mathrm{SF}(s_j) - \mathrm{PSF}(s_j)| \geq \sigma \\ 1 & \text{pro } |\mathrm{SF}(s_j) - \mathrm{PSF}(s_j)| < \sigma, \end{cases} \tag{3.3}$$

kde σ je povolená odchylka predikce fitness a o je počet trainers.

Nejlépe ohodnocený prediktor je používán pro predikci fitness při evoluci kandidátních řešení. Evoluce prediktorů jej přepisuje, zatímco evoluce kandidátních řešení jej čte.

3.4 Vyhodnocení fitness kandidátního řešení

Při koevoluci je třeba vyhodnocovat fitness kandidátního řešení při několika různých konfiguracích. Základním způsobem je ohodnocení kandidátního řešení všemi datovými body z trénovací množiny. Tato konfigurace slouží pro ohodnocení:

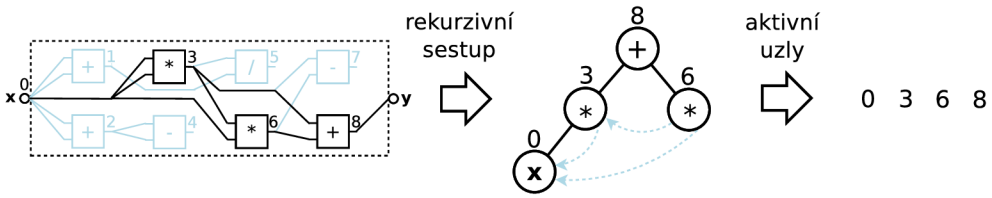
- kandidátního řešení z množiny trainers skutečnou fitness,
- skutečnou fitness kandidátního řešení, které prediktor určil jako výsledné řešení,
- kandidátního řešení při výpočtu bez použití koevoluce.

Další konfigurací je ohodnocení kandidátního řešení pomocí prediktoru fitness, které slouží k ohodnocení:

- kandidátního řešení při evoluci kandidátních řešení,
- prediktoru pomocí kandidátních řešení z množiny trainers.

Zobrazení variant výpočtu fitness je na obrázku 3.3. Výpočet fitness by měl být univerzální pro všechny typy vyhodnocení, které koevoluce vyžaduje, a proto je funkce pro výpočet fitness kandidátního řešení navržena komplexně. Na základě vstupních parametrů provede ohodnocení kandidátního řešení buď pomocí všech datových bodů z trénovací množiny nebo pouze pomocí malé podmnožiny těchto bodů určené prediktorem.

Jak je uvedeno v části 2.1.3, k vyhodnocení fitness kartézského programu lze přistoupit několika způsoby. Pro tuto úlohu jsem zvolila přístup, který spočívá v označení aktivních



Obrázek 3.4: Označení aktivních uzlů.

uzlů (obsažených ve fenotypu) rekurzivním sestupem a následném postupném vyhodnocení aktivních uzlů nad datovými body z trénovací množiny.

V kapitole 2.2 byly uvedeny dva přístupy k výpočtu fitness v úloze symbolické regrese. Z tohoto důvodu jsem zvolila možnost, že kandidátní řešení si bude uchovávat hodnotu založenou na výpočtu absolutní odchylky řešení i na výpočtu skóre.

V případě koevoluce je třeba mít možnost porovnat vypočtenou skutečnou fitness s fitness predikovanou. Proto přidávám hodnotu, která vyjadřuje počet zásahů (skóre) vůči počtu ohodnocených datových bodů. Jestliže při výpočtu skutečné fitness zasáhne do povoleného intervalu například 40 datových bodů z 200 (20 %), odpovídá to pak zásahu 2 bodů z prediktoru o velikosti 10 datových bodů.

3.4.1 Označení aktivních uzlů

Rekurzivní sestup začíná od výstupního uzlu, na který je připojen jemu předcházející vnitřní uzel. Podle funkce vnitřního uzlu se sestupuje na jeho jeden (pro unární funkce) nebo oba vstupy (binární funkce). Může nastat i případ, kdy je vnitřní uzel označen jako koncový a generuje konstantu (o generování konstant více v části 3.4.3) a průchod větví je ukončen. Jestliže se při rekurzivním sestupu narazí na uzel, který již byl označen jako aktivní, průchod větví se ukončí neboť předchůdci tohoto uzlu již jsou označeni. K ukončení průchodu větví dále dojde, když se sestoupí na primární vstup kartézského programu. Ilustrace označení aktivních uzlů je na obrázku 3.4.

3.4.2 Vyhodnocení datového bodu

První aktivní vnitřní uzel kartézského programu pracuje pouze nad primárními vstupy – nezávislými proměnnými danými datovým bodem z trénovací množiny. Následující aktivní uzly pak pracují s hodnotami danými předchozími aktivními uzly nebo primárními vstupy. Při vyhodnocení datového bodu z trénovací množiny dochází k postupnému vyhodnocování aktivních uzlů, až se dojde k primárnímu výstupu kartézského programu. Na primárním výstupu se na závěr nachází funkční hodnota daná kandidátním řešením, které je reprezentováno kartézským programem. Funkční hodnota daná kandidátním řešením je porovnána se skutečnou funkční hodnotou danou datovým bodem z trénovací množiny – závislou proměnnou. Odchylka těchto funkčních hodnot určuje, zda se zvýší fitness hodnota kandidátního řešení daná pomocí skóre (vztahem 2.2), případně se přičte k absolutní odchylce řešení (dané vztahem 2.1).

(a) Generování jedničky. (b) Generování nuly.

Obrázek 3.5: Ukázka převodu aktivního uzlu na uzel generující konstantu.

3.4.3 Generování konstant

Ke generování konstanty dochází v několika případech, a to v závislosti na vstupních hodnotách uzlu.

Ke generování konstanty nezávislé na vstupu dochází při odčítání nebo dělení totožných vstupů funkce ($x - x = 0$, $\frac{x}{x} = 1$). Tato situace nastane, jestliže index uzlu připojeného na první vstup je roven indexu uzlu připojeného na druhý vstup. Pokud je takový uzel nalezen, není třeba zkoumat jeho vstupní uzly, neboť vyčíslení uzlu je za všech podmínek totožné. Ukázka změny uzlu při generování konstant nezávislých na vstupu je na obrázku 3.5.

Existují případy, kdy funkční hodnota uzlu nad vstupními hodnotami nemůže být vyčíslena. To nastane, pokud je hodnota vstupu funkce mimo definiční obor (dělení nulou, logaritmus záporného čísla, atp.). Jestliže vyčíslení uzlu vrátí nepovolenou hodnotu (nekonečno, NaN), pak tomuto uzlu může být přiřazena konstanta nezávislá na jeho vstupech, například π.

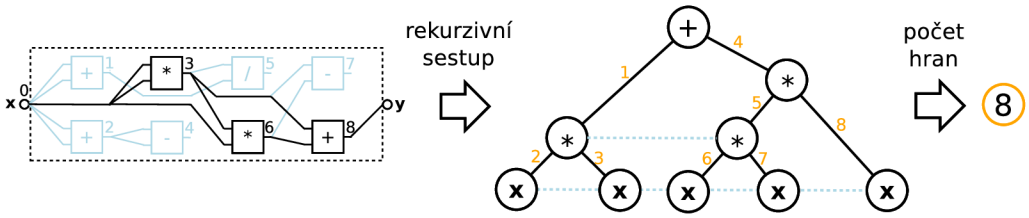
Nepovolená operace jako dělení nulou může vzniknout v kandidátním řešení nezávisle na primárních vstupech, protože konstanta nula se generuje při odčítání stejných hodnot. Může ale nastat i případ, kdy k dělení nulou dojde až po vyčíslení určitého počtu datových bodů, a to pokud se na primárním vstupu objeví nula – jako nezávislá proměnná. Tuto skutečnost je třeba řešit poznačením, že uzel generuje konstantu, a navíc musí kandidátní řešení znovu ohodnotit všechny datové body. Kdyby se v průběhu ohodnocování kandidátního řešení pozměnil výpočet kandidátního řešení, neodpovídala by jeho fitness hodnota skutečnému stavu řešení. Jedna část by tak byla ohodnocena podle starého významu kandidátního řešení a druhá část podle významu změněného generovanou konstantou.

3.4.4 Délka cesty řešení

Aby bylo možné hledat jednodušší matematické závislosti, je třeba mít informaci o velikosti řešení – vztahu. Vzhledem k tomu, že i v průběhu výpočtu fitness může dojít ke změně uzlu na uzel generující konstantu a tím ztratit závislost na vstupech, dochází k určení délky cesty až po ohodnocení všech datových bodů. Zjištění délky cesty řešení probíhá téměř stejným způsobem jako označení aktivních uzlů 3.4.1 s rozdílem, že když při rekurzivním sestupu narazí na uzel, který již byl označen jako aktivní, průchod větví se neukončí. Průchod větví končí až v případě, kdy uzel generuje konstantu nebo je primárním vstupem kartézského programu. Nalezení délky cesty řešení je demonstrováno obrázkem 3.6.

3.5 Evoluce kandidátních řešení bez koevoluce

Řešení bez koevoluce a s použitím koevoluce je implementováno současně a jednotlivé rozdíly jsou odděleny využitím podmíněného překladu pomocí příkazů C-preprocesoru. Podmíněný překlad je vázán na existenci symbolické konstanty `COEVOLUTION`. Program, který



Obrázek 3.6: Délka cesty řešení.

řeší symbolickou regresi, obsahuje kusy kódu, které patří pouze řešení bez využití koevoluce nebo které patří pouze řešení s využitím koevoluce a pak společné části kódu. Díky tomuto přístupu současné implementace řešení s koevolucí a bez koevoluce není třeba při ladění udržovat dva různé kódy pro společné části programu.

Program bez koevoluce tak pouze načte datové body z množiny trénovacích dat do paměti, inicializuje kandidátní řešení a pak provádí evoluci na principu kartézského genetického programování popsaném v části 2.1.3. Při překladu je využíváno pouze modulu pro načtení trénovacích dat, operace s kandidátními řešeními a výpočet fitness kandidátního řešení – bez použití prediktorů. Prediktory fitness, množina trainers, vlákna, ani žádné operace nad nimi v tomto případě nejsou zahrnuty při překladu.

3.6 Koevoluce

Koevoluce v úloze symbolické regrese sestává ze dvou dílčích evolucí. Evoluce populace kandidátních řešení funguje na principu kartézského genetického programování a obstarává ji jedno vlákno, zatímco evoluce populace prediktorů fitness funguje na principu genetického algoritmu a obstarává ji druhé vlákno. Koevoluce, tedy vývoj účelných vlastností na základě změny vlastností jiné populace, probíhá tak, že si populace mezi sebou vyměňují jedince k ohodnocení. Jedinci z jedné populace sloužící k ohodnocení druhé populace se předávají jako sdílené proměnné. Na rozdíl od postupu uvedeného v článku [9], negeneruji nové trainers náhodně periodicky ve vlákně obstarávajícím evoluci prediktorů. Nové trainers jsou generovány v průběhu evoluce kandidátních řešení.

Na počátku koevoluce je třeba inicializovat všechny populace, množiny a proměnné podílející se na koevoluci. Podle nastavených parametrů se náhodně vygeneruje populace kandidátních řešení, pomocí níž se určí první trainers. Inicializace trainers probíhá tak, že prochází populaci čerstvě inicializovaných kandidátních řešení a do trainers se vybírají kandidátní řešení s různou skutečnou fitness. Pokud dojdou jedinci v populaci kandidátních řešení, jsou zbylé trainers generovány náhodně, tak aby měly různou fitness. Při inicializaci trainers dochází rovněž k inicializaci mutexu pro vyloučení vícenásobného přístupu k trainers.

Prediktory se inicializují generováním náhodných čísel v rozsahu počtu datových bodů z trénovací množiny tak, aby se v chromozomu jedince každý datový bod vyskytoval nejvýše jednou. Následně dojde k inicializaci a zamknutí mutexu pro vyloučení vícenásobného přístupu k nejlépe ohodnocenému prediktoru. Aby mohla evoluce kandidátních řešení používat prediktor k ohodnocení jedinců, musí nejdřív evoluce prediktorů v první generaci určit nejlépe ohodnocený prediktor. Po průběhu první iterace evolučního cyklu prediktorů fitness se již proměnná s nejlépe ohodnoceným prediktorem odemkne.

3.6.1 Evoluce kandidátních řešení

Při první iteraci evolučního cyklu kandidátních řešení vlákno čeká na odemknutí mutexu proměnné s nejlépe ohodnoceným prediktorem. Nejlépe ohodnocený prediktor načte a dále jej používá pro ohodnocení jedinců. K načtení nového prediktoru dochází periodicky. Po té, co vlákno evoluce kandidátních řešení načte nový prediktor, pošle signál vláknu evoluce prediktorů fitness, že může pokračovat další iterací a tak nachystat nový prediktor ke čtení. Tím je zajištěno řízené zpomalení evoluce prediktorů fitness, aby nový prediktor byl nalezen pouze čas od času a tím zbylo více výpočetních prostředků na evoluci kandidátních řešení.

Následuje ohodnocení jedinců současné generace pomocí prediktoru. Pokud je nalezen jedinec s lepší predikovanou fitness než doposud, je zařazen mezi nové trainers. Výběr rodiče je možný podle různých kritérií. K dispozici je predikovaná fitness hodnota daná absolutní odchylkou (vztahem 2.15), fitness hodnota určující poměrné skóre a velikost kandidátního řešení. S upřednostněním kritérií pro výběr rodiče byly provedeny experimenty a výsledky jsou popsány v kapitole 6.2. Pravidlem zůstává, že pokud mají dva jedinci v populaci stejnou nejlepší fitness, je jako rodič vybrán ten, který nebyl rodičem současné generace.

Nová generace, jak je tomu u kartézského genetického programování, vzniká z nejlépe ohodnoceného jedince a vytvořením λ jedinců pomocí mutace nejlépe ohodnoceného jedince. S počtem mutací v nově vznikajícím jedinci experimentuji tak, že počet mutací je náhodný v rozmezí $\langle 1, mut\_max \rangle$. Při hledání vhodného nastavení evoluce CGP zkoumám i parametr mut_max.

Jestliže některé kandidátní řešení podle ohodnocení fitness prediktorem spadá do intervalu povolené odchylky od hledaného řešení, je proveden výpočet skutečné fitness řešení. Pokud i skutečná fitness hodnota spadá do povoleného intervalu, je nalezeno řešení a koevoluce je ukončena.

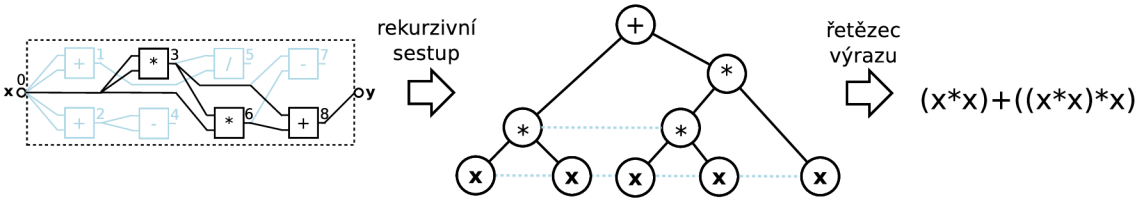
Pokud kandidátní řešení ohodnocené prediktorem fitness spadá do intervalu s povolenou odchylkou a jeho skutečná fitness ne, koevoluce pokračuje. Je poslán signál pro nalezení nového prediktoru (současný prediktor ohodnotil řešení jako správné, ale podle skutečné fitness správné není). I když predikovaná fitness stále spadá do intervalu povolených řešení, je nové ohodnocení skutečnou fitness provedeno až po určitém počtu iterací evolučního cyklu. Tím se sníží frekvence ohodnocování skutečnou fitness a dá se čas evoluci, aby mohla kandidátní řešení upravit s pomocí prediktoru.

Nahrazení trainers

Periodicky se provádí výběr nových trainers, které pak nahradí používané trainers. Nové trainers z poloviny sestávají z nejlépe (různě) ohodnocených jedinců z evoluce kandidátních řešení a z druhé poloviny z náhodně generovaných řešení s různou fitness. V článku [9] uvádí, že nové prediktory generují periodicky při evoluci prediktorů. Moje řešení ale umožňuje, aby se prediktory naučily ohodnotit správně řešení, které aktuálně evoluce kandidátních řešení generuje.

3.6.2 Evoluce prediktorů

Na počátku iterace evolučního cyklu prediktorů fitness jsou načteny trainers ze sdílené proměnné, pokud byly změněny. Následně jsou ohodnoceny pomocí skutečné fitness. K ohodnocení prediktorů dochází tak, že pro každý prediktor se vyhodnotí fitness každého jedince z množiny trainers. Na základě odchylky predikované fitness a skutečné fitness jedinců



Obrázek 3.7: Generování řetězce výrazu.

z množiny trainers se vypočítá fitness hodnota prediktoru. Vztah pro výpočet fitness prediktoru je uveden v části 3.3.1.

V populaci prediktorů fitness je vždy navíc uchováván jedinec, který měl nejlepší fitness v předchozí generaci. Při změně trainers se může změnit jeho fitness hodnota (může se i snížit) a nemusí pak již být nejvhodnějším kandidátem pro predikci fitness kandidátních řešení.

Nejlépe ohodnocený prediktor je uložen ve sdílené proměnné, kterou používá evoluce kandidátních řešení k ohodnocení jedinců.

Vytvoření nové generace prediktorů využívá turnajové selekce rodičů a následného křížení a mutace potomků. Ze současné generace jsou vybráni náhodně dva jedinci, lepší z nich se stává prvním rodičem potomků, výběr druhého rodiče probíhá stejně. Potomci jsou vytvořeni pomocí jednobodového křížení a jejich chromozom je mutován se zadanou pravděpodobností. U nově vzniklého potomka je provedena kontrola, zda se indexy datových bodů v chromozomu neopakují. Pokud tomu tak je, pak je takový gen mutován. Tato kontrola zabrání snížení počtu datových bodů v prediktoru.

Evoluce prediktorů fitness pokračuje do doby, než algoritmus dostane signál od vlákna obstarávajícího evoluci kandidátních řešení, že požadované řešení již bylo nalezeno, a končí koevoluce.

Variabilita prediktorů

Při nízkém počtu jedinců v populaci prediktorů se vyskytl problém, že populace po určitém počtu iterací přestala být rozmanitá a ke změnám jedinců docházelo převážně jen na základě mutací, neboť všichni jedinci si byli velmi podobní (elitismus). Z důvodu zachování rozmanitosti populace je nejhůře ohodnocený jedinec odstraněn a nahrazen náhodným mutantem, který se pak podílí na reprodukci.

3.7 Výpis řetězce řešení

Sestavení řetězce reprezentujícího řešení z chromozomu probíhá jako průchod abstraktním syntaktickým stromem. Při tomto průchodu je orientovaný acyklický graf abstraktně rozbalen na strom a rekurzivním sestupem (průchodem post order) je sestaven řetězec. Tento postup je ilustrován obrázkem 3.7.

3.8 Návrh struktury souboru s trénovacími daty

Struktura souboru s trénovacími daty je navrhována pro co nejúčelnější zpracování. Proto první hodnota na začátku textového souboru značí počet datových bodů v trénovací množině a druhá hodnota, oddělená bílým znakem[1], počet nezávislých proměnných. Následují vždy nezávislé proměnné a závislá proměnná pro daný datový bod oddělené bílými znaky. Obsah souboru pro pět datových bodů s jednou nezávislou proměnnou pak může vypadat takto:

```
 5   1                          počet datových bodů a počet nezávislých proměnných
-2  12                                                        první datový bod
-1   2                                                        druhý datový bod
 0   3.53084e-28                                              třetí datový bod
 1   1.88738e-14                                              čtvrtý datový bod
 2  -4                                                        pátý datový bod
```

U ilustrovaného souboru je od druhého řádku vždy nejdříve nezávislá hodnota, několik mezer a pak závislá hodnota daného datového bodu. Další datový bod je na novém řádku. V případě, že by soubor obsahoval méně datových bodů než je uvedeno v záhlaví, program detekuje chybu. Pokud je datových bodů více, než je uvedeno, přebývající datové body jsou ignorovány.

[1] "C"a "POSIX"definují tyto bílé znaky: mezera, form-feed (`'\f'`), nový řádek (`'\n'`), posun kurzoru na začátek řádku (`'\r'`), horizontální tabulátor (`'\t'`) a vertikální tabulátor (`'\v'`).

Kapitola 4

Implementace symbolické regrese

Program řešící symbolickou regresi pomocí kartézského genetického programování s užitím koevoluce a bez koevoluce je implementován v jazyce C. Pro překlad a sestavení programu používám překladač GNU GCC a program GNU make.

Při implementaci řešení s užitím koevoluce a bez koevoluce jsem využila podmíněného překladu pomocí příkazů C-preprocesoru tak, jak je popsáno v knize [1]. Zdrojový program je označením určitých částí rozdělen na části sloužící pouze pro řešení bez koevoluce, pouze pro řešení s užitím koevoluce a na společné části. Je tedy možné z jedné sady zdrojových souborů sestavit dva různé programy, které vzájemně neobsahují nadbytečné části pro danou variantu řešení. Toto řešení je výhodné, neboť při vývoji a údržbě není třeba udržovat dvě sady zdrojových souborů a společné části programu lze měnit současně.

Implementovaný podmíněný překlad funguje na základě využití existence symbolické konstanty `COEVOLUTION`. Jelikož řešení bez užití koevoluce nepotřebuje využít všechny implementované moduly, jsou ve skriptu Makefile definovány dvě verze překladu. Detaily překladu a spuštění těchto programů se zabývá část 4.7.

Program je rozdělen do několika dílčích modulů. Jádro programu řídí vlákna pro jednotlivé populace, komunikaci mezi nimi, průběh evolučních cyklů populací a celkovou koevoluci. Funkce jádra programu je popsána v části 4.1. Definice struktury reprezentující kandidátní řešení a potřebné informace o kandidátním řešení (jako například jeho hodnotu fitness) a operace nad touto strukturou se nachází v modulu popsaném v části 4.3. Další moduly jednotlivě řeší definici struktur a operace nad trainers, prediktory fitness a trénovacími daty (v částech 4.2, 4.4, 4.5). Popis modulu pro univerzální výpočet fitness nad všemi variantami uvedenými v části 3.4 a pomocné operace pro výpočet fitness je uveden v části 4.6.

4.1 Jádro programu

Jádro programu obstarává základní běh řešení symbolické regrese. Řídí populace a jejich evoluce, které jádro programu zpracovává paralelně pomocí vláken. Tento model paralelního programování funguje na principu paralelního provádění nezávislých funkcí. Každé vlákno má vyhrazen svůj vlastní zásobník, na kterém má uloženy lokální proměnné, parametry a návratové adresy aktivních funkcí. Každé vlákno může vyvolat kteroukoli funkci programu a používat kteroukoli globální proměnnou. Skupina vláken v jednom adresovém prostoru pak tvoří proces. Systémové zdroje jsou přidělovány na úrovni procesů, takže všechna vlákna sdílí deskriptory otevřených souborů, nastavení zpracování signálů, zámky souborů, a tak podobně.

Model s užitím vláken, jak uvádí [5], je rychlejší než paralelní programování založené na principu spouštění paralelních procesů, neboť ty vyžadují přepínání kontextu mezi nezávislými procesy s různými adresovými prostory. Pro paralelní programování je tedy mnohem výhodnější použití vláken, kdy je více paralelních procesů soustředěno ve stejném adresovém prostoru a mohou tak přímo mezi sebou komunikovat sdílenou pamětí a přepínání kontextu nevyžaduje přepínaní adresového prostoru.

Pro implementaci s použitím vláken využívám knihovnu pro GNU/Linux podle standardu POSIX (pthreads). Všechny funkce a datové typy jsou deklarovány v `pthread.h`. Funkce nejsou ve standardní C knihovně, ale v knihovně libpthread, takže je potřeba při překladu linkovat s parametrem `-lpthread`.

V procesu obstarávajícím koevoluci je v hlavním vlákně obsluhována evoluce kandidátních řešení. Evoluci prediktorů zajišťuje druhé vlákno spuštěné po počáteční inicializaci. Po nalezení řešení při evoluci kandidátních řešení je druhé vlákno ukončeno a jeho stav ukončení převezme první vlákno.

Tato dvě vlákna sdílí dvě globální proměnné, nejlépe ohodnocený prediktor a množinu trainers, a proměnné zajišťující synchronizaci. Každá ze sdílených globálních proměnných je chráněna binárním semaforem (*mutual exclusion*) pro vzájemné vyloučení přístupu k proměnné. Jelikož evoluce prediktorů běží pomaleji než evoluce kandidátních řešení, používám pro pozastavení vlákna s evolucí prediktorů podmíněnou proměnnou (*condition variable*). Vlákno pro evoluci prediktoru v průběhu každé iterace svého evolučního cyklu čeká na signál, po jehož přijetí proběhne další iterace. Tato podmíněná proměnná je svázána se vzájemným vyloučením operací nad nejlépe ohodnoceným prediktorem. Aby kritická sekce operací nad sdílenými proměnnými byla co nejkratší, je uvnitř sdílená proměnná vždy pouze načtena do lokální proměnné daného vlákna, nebo naopak je lokální proměnná daného vlákna uložena do sdílené proměnné.

4.1.1 Řešení bez koevoluce

Řešení bez užití koevoluce však neobsahuje výše uvedenou paralelizaci výpočtu. Při řešení se využívá pouze jedna populace, a to populace kandidátních řešení, která je obsluhována sekvenčně. Algoritmus řešení symbolické regrese s užitím kartézského genetického programování obsahuje tyto kroky (kroky společné pro oba přístupy, s koevolucí i bez koevoluce, jsou *zvýrazněny*):

1. *Načtení trénovacích dat ze souboru a uložení do paměti.*
2. *Inicializace populace kandidátních řešení.*
3. Ohodnocení všech jedinců v populaci pomocí celé množiny trénovacích dat.
4. *Výběr nejlépe ohodnoceného jedince.*
5. Ověření konvergence:
   (a) Fitness nejlépe ohodnoceného jedince spadá do intervalu povolených řešení – evoluce ukončena a vypsáno řešení.
   (b) Fitness nejlépe ohodnoceného jedince nespadá do intervalu povolených řešení – evoluce pokračuje.
6. *Vytvoření nové generace a pokračování krokem 3.*

4.1.2 Řešení s koevolucí

Řešení symbolické regrese s užitím koevoluce lze rozdělit na tři základní části. První část inicializuje a ukončuje koevoluci a další dvě části obsluhují jednotlivé evoluční cykly pro kandidátní řešení a prediktory.

Inicializace a ukončení koevoluce

1. *Načtení trénovacích dat ze souboru a uložení do paměti.*
2. *Inicializace populace kandidátních řešení.*
3. Inicializace populace trainers.
4. Inicializace populace prediktorů fitness a uzamknutí mutexu pro nejlépe ohodnocený prediktor fitness.
5. **Spuštění vlákna pro evoluci prediktorů fitness.**
6. **Průběh cyklu pro evoluci kandidátních řešení.**
7. Po skončení cyklu pro evoluci kandidátních řešení poslání signálu o ukončení vláknu s evolucí prediktorů fitness.
8. Spojení vláken a ukončení koevoluce.

Evoluce prediktorů fitness

1. Načtení trainers ze sdílené proměnné do lokální proměnné.
2. Ohodnocení všech jedinců z populace prediktorů. Každý prediktor je ohodnocen pomocí celé množiny trainers.
3. Nahrazení nejhůře ohodnoceného prediktoru náhodně generovaným mutantem.
4. Uložení nejlépe ohodnoceného prediktoru do sdílené proměnné.
   (a) V první iteraci cyklu se pouze uloží nejlépe ohodnocený prediktor a následně se odemkne mutex.
   (b) V dalších iteracích se vždy čeká na podmínku (*condition variable*) vázanou na sdílenou proměnnou nejlépe ohodnocený prediktor. Ve chvíli, kdy je signál přijat, uloží se nový nejlépe ohodnocený prediktor.
   (c) Po přijetí signálu se kontroluje, zda již není konec koevoluce (řešení nalezeno). Pokud ano, vlákno ukončí svoji činnost.
5. Vytvoření nové generace prediktorů fitness a pokračování krokem 1.

Evoluce kandidátních řešení

1. Vždy po určitém počtu iterací evolučního cyklu dojde k načtení nejlépe ohodnoceného prediktoru ze sdílené proměnné do lokální proměnné a poslání signálu druhému vláknu.

2. Ohodnocení všech jedinců z populace kandidátních řešení pomocí nejlépe ohodnoceného prediktoru.

3. *Výběr nejlépe ohodnoceného jedince.*

4. Ověření konvergence:

   (a) Predikovaná fitness nejlépe ohodnoceného jedince spadá do intervalu povolených řešení – vypočtení skutečné fitness.

      i. Skutečná fitness nejlépe ohodnoceného jedince spadá do intervalu povolených řešení – evoluce ukončena a vypsáno řešení.

      ii. Skutečná fitness nejlépe ohodnoceného jedince nespadá do intervalu povolených řešení – evoluce pokračuje. K dalšímu ověření skutečné fitness dojde až po určitém počtu iterací, přestože predikovaná fitness stále spadá do intervalu povolených řešení.

   (b) Predikovaná fitness nejlépe ohodnoceného jedince nespadá do intervalu povolených řešení – evoluce pokračuje.

5. Pokud má nejlépe ohodnocený jedinec lepší fitness než předchozí nejlépe ohodnocený jedinec, je zařazen mezi nové trainers.

6. Jednou za čas se vygeneruje náhodná polovina trainers a nové trainers se uloží do sdílené proměnné.

7. *Vytvoření nové generace kandidátních řešení a pokračování krokem 1.*

4.1.3 Záznamy pro statistiky

V průběhu koevoluce, resp. řešení bez užití koevoluce, se zaznamenávají některé hodnoty, které pak slouží pro zpracování statistiky. Sleduje se například konvergence fitness kandidátních řešení či prediktorů fitness, a tyto hodnoty jsou uchovávány v poli, které se při ukončení koevoluce přepíše do souboru. Pro jednotlivé nastavení koevoluce nebo řešení bez koevoluce tak lze sledovat změny fitness nejlépe ohodnocených jedinců v průběhu generací.

Při nalezení řešení pak jádro programu vypíše řetězec řešení ve formátu pro gnuplot[1], což je vhodné pro ověření správnosti nalezeného řešení. Dále jsou vypsány aktuální nastavení koevoluce, generace, ve které bylo řešení nalezeno, či počet datových bodů, které bylo nutné vyčíslit pomocí kandidátních řešení v průběhu koevoluce, a doba běhu programu. V případě výpočtu bez koevoluce, který je řešen sekvenčně, stačí zjistit dobu běhu jednoho vlákna procesu. Při použití koevoluce je třeba vzít v úvahu dobu výpočtu jednotlivých vláken, případně jejich součet. Tyto informace umožňuje zjistit funkce jádra operačního systému pro zjištění využití zdrojů `getrusage()` specifikovaná podle standardu POSIX.1-2001. Pro využití této funkce je nutné připojit knihovny `sys/time.h` a `sys/resource.h`. Voláním

[1] Gnuplot je interaktivní program pro generování dvoudimenzionálních a trojdimenzionálních grafů funkcí či dat. Program je distribuován pod svobodnou licencí.

funkce `getrusage()` na konci fuknce vlákna obstarávajícího evoluci kandidátních řešení lze zjistit dobu využití zdrojů volajícího vlákna (`RUSAGE_THREAD`) i součet využití zdrojů za všechna vlákna v procesu (`RUSAGE_SELF`).

4.1.4 Generování pseudonáhodných čísel

Při inicializaci populací i dalších operací v průběhu evolucí je nutné využívat náhodně generované hodnoty. Pro inicializaci generátoru pseudonáhodných čísel je možné použít funkci `time()` z knihovny `time.h`, ale toto řešení je nevhodné při dávkovém zpracování testů. V tomto případě se inicializuje několik běhů stejnou hodnotou, pokud jsou spuštěny v průběhu téže sekundy.

Při testování na víceprocesorovém systému, kde testované procesy běží současně, je nutné použít inicializační funkci, která umožňuje vyšší rozlišení času. Proto jsem použila funkci `gettimeofday()` z knihovny `sys/time.h`, která vrátí čas rozlišitelný v mikrosekundách od začátku dne, což je dostačující rozlišení proto, aby generátor pseudonáhodných čísel byl inicializován vždy různou hodnotou. Nevýhodou je, že se čísla mohou opakovat každých 24 hodin. Při testování jsem však na problém opakování inicializujícího čísla nenarazila a proto ji považuji za dostačující.

4.2 Datové body

Úvodním problémem, který musí program řešit, je načtení trénovacích datových bodů. Trénovací data jsou uložena v souboru, jehož struktura je definována v části 3.8. Takto definovaná struktura umožňuje jednoduché načtení trénovacích datových bodů a uložení do paměti. Pro uložení datových bodů do paměti slouží struktura `sDataSet`, která uchovává celkový počet datových bodů, počet nezávislých proměnných každého bodu a pole s jednotlivými datovými body. Každý datový bod je rovněž struktura obsahující nezávislé proměnné a závislou proměnnou ve formátu `double`. Modul pro obsluhu datových bodů obsahuje jedinou funkci, která slouží pro načtení množiny datových bodů ze souboru do proměnné typu `sDataSet`. Pro proměnnou s množinou trénovacích dat je na začátku běhu programu dynamicky alokována paměť a znovu měněna (uvolněna) je až v závěru.

4.3 Kandidátní řešení

Jak již bylo zmíněno v návrhu, kandidátní řešení si s sebou kromě chromozomu musí nést i informace, jakou má hodnotu fitness či délku cesty řešení. Tyto informace se používají při výběru nejlépe ohodnoceného jedince a zároveň rodiče další generace. Ve struktuře reprezentující kandidátní řešení `sIndividalCGP` je dvourozměrné pole reprezentující chromozom (popsáno v části 3.2), fitness hodnota kandidátního řešení založená na absolutní odchylce trénovacích datových bodů (vztahem 2.1), fitness hodnota určená jako skóre (vztahem 2.2), pak poměrné skóre vzhledem k počtu trénovacích bodů, pomocí kterých došlo k ohodnocení kandidátního řešení, a délka cesty řešení.

Modul, který obstarává obsluhu operací nad proměnnou typu `sIndividalCGP` a jejich populací, obsahuje několik základních funkcí. Při inicializaci se využívají funkce pro vytvoření populace jedinců a náhodnou inicializaci jedince, kterou využívá i inicializace a nahrazení trainers. Nachází se zde i funkce pro vypsání chromozomu, kopírování chromozomu, převod chromozomu na řetězec reprezentující matematický výraz a funkce pro nalezení rodiče, vytvoření nové generace a mutaci jedince. Výběr rodiče je implementován

na základě tří kritérií, jejichž pořadí a užitečnost zkoumám v rámci testování symbolické regrese pomocí kartézského genetického programování bez užití koevoluce. Zkoumanými kritérii na vliv rychlosti konvergence ke správnému řešení jsou fitness hodnota skóre, fitness hodnota určená absolutní odchylkou a délka cesty kandidátního řešení.

4.4 Trainers

Struktura, která reprezentuje množinu trainers obsahuje pole jedinců kandidátních řešení typu `sIndividualCGP` a také proměnnou typu `pthread_mutex_t`, která zaručuje výhradní přístup k trainers, neboť množina trainers je sdílená proměnná. Modul pro práci s trainers obsahuje funkci pro inicializaci trainers a funkce pro uzamknutí a odemknutí mutexu.

4.5 Prediktory fitness

V modulu pro práci s prediktory fitness jsou deklarovány dvě základní struktury. Struktura `sIndividualPredictor` reprezentuje prediktor fitness s chromozomem a různě spočítanými hodnotami fitness, jak je navrženo v části 3.3.

Druhá struktura deklarovaná v modulu pro práci s prediktory je nejlépe ohodnocený prediktor `sBestPredictor`, která obsahuje jednoho jedince z populace prediktorů a to proměnnou typu `sIndividualPredictor`. Jelikož nejlépe ohodnocený prediktor je sdílená proměnná, musí obsahovat proměnnou typu `pthread_mutex_t`, která zaručuje výhradní přístup k nejlépe ohodnocenému prediktoru a podmíněnou proměnnou typu `pthread_cond_t`, která slouží ke zpomalení evoluce prediktorů.

Pro práci s prediktory jsou implementovány funkce, které inicializují počáteční populaci prediktorů, náhodně vytvoří jedince z populace prediktorů podle počtu dostupných trénovacích bodů tak, aby se žádný z trénovacích bodů v prediktoru neopakoval.

Funkce pro vyčíslení fitness prediktoru funguje tak, že pomocí prediktoru jsou postupně ohodnoceni jedinci z množiny kandidátních řešení. Aby se predikovaná fitness nezapisovala přímo do daného traineru, kde se nachází vypočtená skutečná fitness, je trainer zkopírován pouze pro účel ohodnocení prediktorem. Následně je porovnána predikovaná fitness s fitness skutečnou. Z rozdílu skutečné a predikované fitness se pak počítá fitness hodnota prediktoru.

Modul rovněž obsahuje funkce pro reprodukci a vytvoření nové generace pomocí turnajové selekce, jednobodového křížení a mutace chromozomu, pro vypsání chromozomu prediktoru a pro práci s podmíněnou proměnnou zajišťující výhradní přístup k nejlépe ohodnocenému prediktoru.

4.6 Výpočet fitness

Jak je uvedeno v části 3.4, výpočet fitness kandidátního řešení obsahuje několik dílčích problémů. Modul obsahuje jedinou externě přístupnou funkci, která slouží pro vyhodnocení fitness. Funkce přijímá jako vstupní parametry ukazatel na strukturu kandidátního řešení, které ohodnotí na základě jeho chromozomu a ohodnocení uloží do atributů pro hodnoty fitness, dále přijímá ukazatel na celou množinu datových bodů a ukazatel na prediktor. Pokud ukazatel na prediktor je `NULL`, dojde k ohodnocení kandidátního řešení pomocí celé množiny trénovacích datových bodů. Takto je zajištěno, že stejná funkce může ohodnotit kandidátní řešení pomocí prediktoru, ale i pomocí celé množiny trénovacích bodů. Postup vyhodnocení fitness je popsán v části 3.4.2.

Pro efektivnější výpočet fitness označením aktivních uzlů a zjištění délky cesty řešení jsou implementovány funkce pracující na principu průchodu grafem (stromem) metodou post order. Při označení aktivních uzlů v případě orientovaného acyklického grafu to umožňuje přerušit průchod, pokud je navštíven uzel, který již byl označen, protože všichni jeho následovníci jsou již také označeni.

4.7 Kompilace a spuštění

Program, který řeší symbolickou regresi pomocí kartézského genetického programování, se skládá z třinácti dílčích souborů. Šest dvojic představuje dílčí moduly - knihovnu deklarací externě dostupných funkcí a datových typů a soubor s definicemi funkcí modulu. Makefile pak řídí překlady jednotlivých modulů a sestavení programu.

V Makefile jsou jako parametry překladu modulů předávány i definice konstant, které určují nastavení parametrů koevoluce. Tím je zajištěna možnost dávkového zpracování obsluhovaného skriptem při testování různých nastavení koevoluce. Testovacímu skriptu se věnuje část 6.1.

Kompilace: K přeložení programu je potřeba překladač GNU GCC a program GNU make. Pomocí Makefile je oddělen překlad řešení s užitím koevoluce a řešení bez užití koevoluce. Kompilace programu, který řeší symbolickou regresi, se vyvolá příkazem:

`make`	*kompilace řešení s užitím koevoluce*
`make nocoe`	*kompilace řešení bez koevoluce*

Spuštění: Programům, které řeší symbolickou regresi ať s užitím koevoluce nebo bez ní, může být předán jeden parametr, a to název souboru s trénovacími daty. V případě, že tento parametr není zadán, hledá se soubor s názvem `data.dta`. Samotné spuštění programu se na unixových systémech provede příkazy:

`./coevolution funkce1.dta`	
`./coevolution`	*spuštění řešení s užitím koevoluce*
`./solution funkce1.dta`	
`./solution`	*spuštění řešení bez koevoluce*

Výstupy: Program před ukončením vypíše dva výstupní řetězce. První řetězec obsahuje nalezené řešení a informace o jeho nalezení jako generaci, ve které bylo řešení nalezeno, počet vyhodnocení datových bodů či dobu výpočtu řešení. Tento řetězec je vypsán na standardní výstup. Druhý řetězec obsahuje záznamy fitness hodnot v průběhu evoluce kandidátních řešení. Tento řetězec je vždy připsán do souboru, jehož název se skládá z názvu souboru s trénovacími daty a parametrů nastavení koevoluce či jen evoluce bez užití koevoluce. Oba z výstupních řetězců obsahují jen jeden řádek s informacemi oddělenými mezerami, což umožňuje jednoduché dávkové zpracování výstupních dat pro vytvoření statistik.

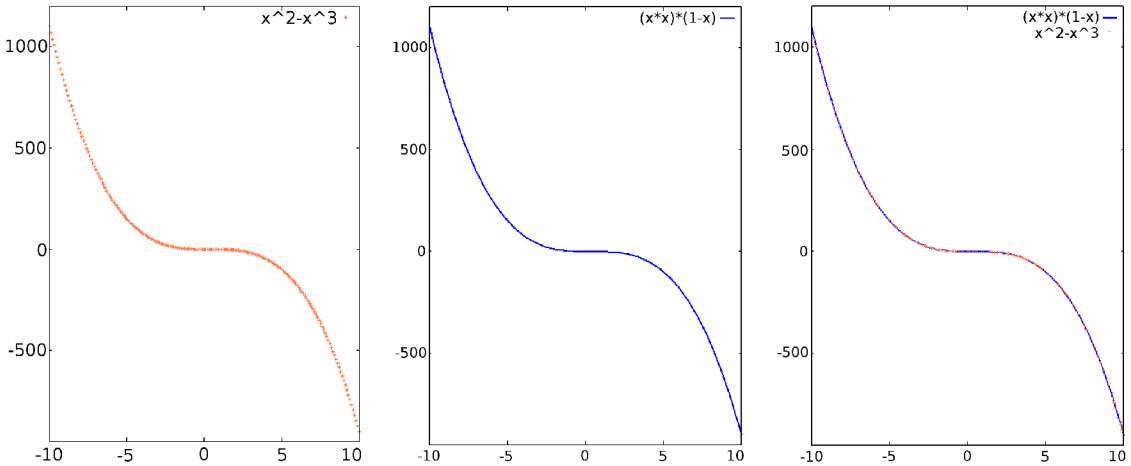
Kapitola 5

Ověření funkčnosti programu na zadaných úlohách

Ověření správné funkčnosti programu pro symbolickou regresi proběhlo porovnáním zobrazení trénovacích dat a následně funkcí, které symbolická regrese našla jako řešení, pomocí programu gnuplot. Trénovací data byla vytvořena vzorkováním funkce na zadaném intervalu. Šířka intervalů a počet vzorků odpovídá intervalům v článku [9], aby bylo následně možné porovnat výsledky koevoluce s kartézským genetickým programováním s koevolucí s genetickým programováním uvedeným v článku o koevoluci prediktorů fitness.

Ověření správnosti nalezení funkce $f(x) = x^2 - x^3$

Napřed jsem testovala funkci s jednoduchým klesajícím průběhem $f(x) = x^2 - x^3$. Trénovací data byla vytvořena pomocí 201 vzorků z intervalu $\langle -10, 10 \rangle$. Takto vytvořené datové body jsou zobrazeny na obrázku 5.1a. Implementované řešení symbolické regrese našlo stejný vztah v různé podobě zápisu, nejčastěji jako `(x*x)*(1-x)`. Průběh nalezené funkce je na obrázku 5.1b. Porovnání řešení je na obrázku 5.1c, přičemž všechny body z trénovací množiny leží přímo v průběhu nalezené funkce.

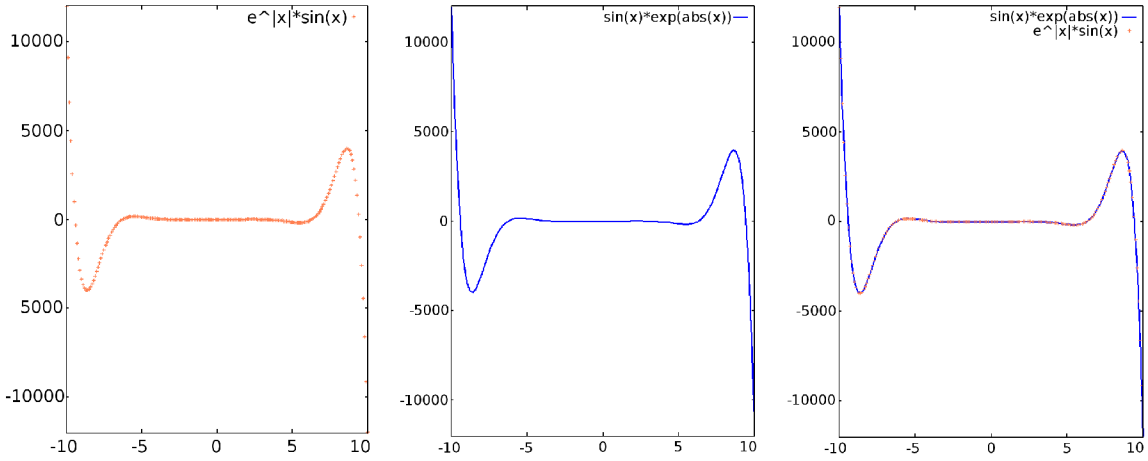


(a) Trénovací body. (b) Nalezený vztah. (c) Porovnání.

Obrázek 5.1: Zobrazení pro porovnání funkce $f(x) = x^2 - x^3$.

Ověření správnosti nalezení funkce $f(x) = e^{|x|}\sin(x)$

Následoval test funkce s několika lokálními extrémy $f(x) = e^{|x|}\sin(x)$. Trénovací data byla vytvořena pomocí 201 vzorků z intervalu $\langle -10, 10\rangle$. Takto vytvořené datové body jsou zobrazeny na obrázku 5.2a. Implementované řešení symbolické regrese našlo tentýž vztah v různé podobě zápisu, nejčastěji jako `sin(x)*exp(abs(x))`. Průběh nalezené funkce je na obrázku 5.2b. Porovnání řešení je na obrázku 5.2c, přičemž všechny body z trénovací množiny leží přímo v průběhu nalezené funkce.



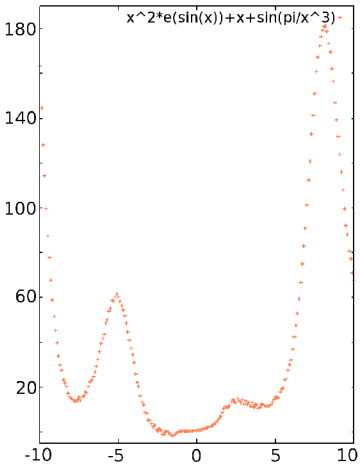
(a) Trénovací body. (b) Nalezený vztah. (c) Porovnání.

Obrázek 5.2: Zobrazení pro porovnání funkce $f(x) = e^{|x|}\sin(x)$.

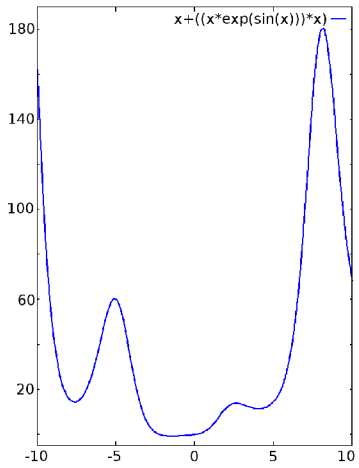
Ověření správnosti nalezení funkce $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$

Pro test funkce $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$ byla trénovací data vytvořena také pomocí 201 vzorků z intervalu $\langle -10, 10\rangle$. Takto vytvořené datové body jsou vyobrazeny na obrázku 5.3a. Implementované řešení symbolické regrese našlo vztah v různé podobě zápisu, nejčastěji jako `x*((x*exp(sin(x)))+1)`, který se však liší od původního vztahu. Nalezený výraz odpovídá funkci $f(x) = x^2 \cdot e^{(\sin(x))} + x$. Je patrné, že oproti původní funkci došlo k zanedbání posledního členu $\sin\left(\frac{\pi}{x^3}\right)$, což je způsobené vlivem povolené odchylky nalezeného řešení a nastavením hledání jednodušších vztahů. Změny funkčních hodnot, které poslední člen generuje, jsou menší než povolená odchylka datového bodu. Průběh nalezené funkce je na obrázku 5.3b.

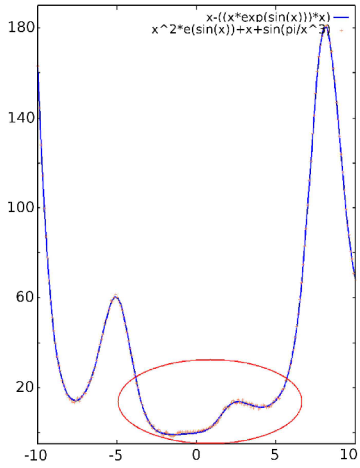
Na obrázku 5.3c, kde je porovnání řešení, je vyznačena oblast, kde při symbolické regresi došlo k patrnému „vyhlazení“ průběhu. Detail této oblasti je zobrazen na obrázku 5.3d. Se zvýšením přesnosti hledaného řešení (snížením maximální povolené odchylky) se v případě řešení s užitím koevoluce mnohonásobně zvýší doba potřebná k nalezení řešení. V případě řešení bez koevoluce se řešení nepodaří nalézt v průběhu osmi milionů generací (testováno pro 50 běhů optimálního nastavení). Evoluce pro osm milionů generací se zjištěným optimálním nastavením evoluce CGP (zjištěním optimálního nastavení se zabývám v kapitole o testování 7) trvá v řádech hodin – průměrně 9 hodin na procesoru Intel® Xeon® CPU X5675 @ 3.07 GHz.



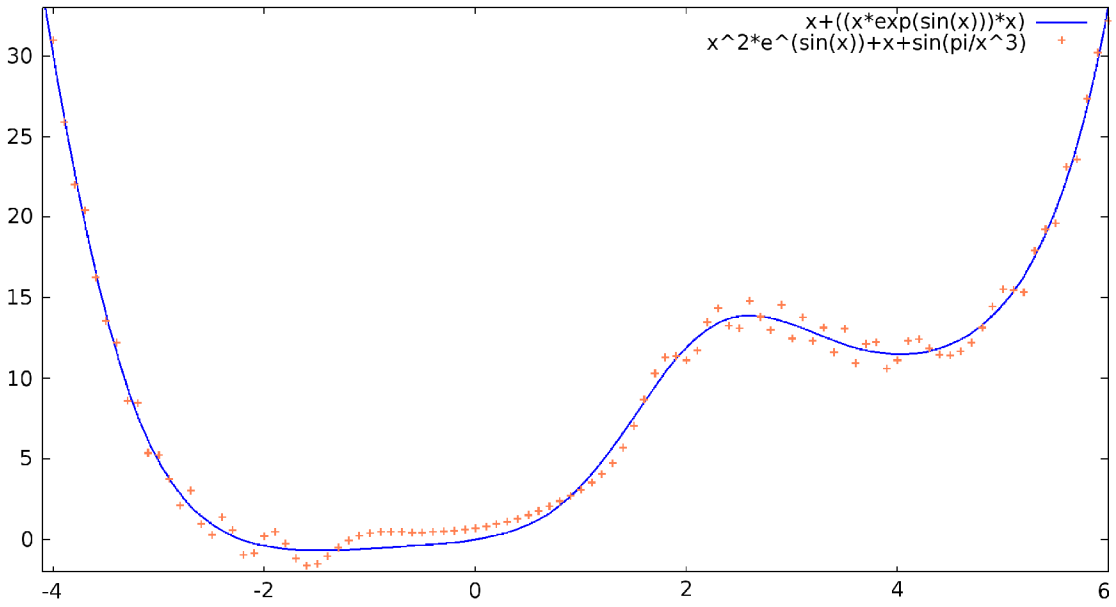
(a) Trénovací body.



(b) Nalezený vztah.



(c) Porovnání.



(d) Vyhlazení části průběhu funkce – detail obrázku 5.3c

Obrázek 5.3: Zobrazení pro porovnání funkce $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$.

Kapitola 6

Testování implementovaných programů

Testy probíhaly automaticky pomocí testovacích skriptů napsaných pro interpret Bash. Popis těchto skriptů je v části 6.1. Pro testování jsem použila server se dvěma procesory procesorem Intel® Xeon® X5675 s operačním systémem Linux. Tyto procesory umožňují současný běh 24 vláken, což je vhodné pro paralelní zpracování testů.

Pro každé nastavení evoluce či koevoluce proběhlo 50 testů nad všemi třemi úlohami zmíněnými v kapitole 5. Pro testování koevoluce bylo nejprve třeba najít co nejvýhodnější nastavení hledání řešení pomocí kartézského genetického programování. V části 6.3 se zabývám hledáním tohoto nastavení řešení bez užití koevoluce. Nalezené co nejvýhodnější nastavení evoluce kandidátních řešení je pak použito jako výchozí pro řešení s koevolucí a je hledáno pouze co nejvýhodnější nastavení koevoluce vzhledem ke statickému nastavení evoluce kandidátních řešení. Hledání co nejvýhodnějšího nastavení s užitím koevoluce je popsáno v části 6.4.

6.1 Skript pro testování

Jelikož server, na kterém byla symbolická regrese testována, umožňoval současný běh až 24 vláken, byl testovací systém v jazyce interpretu Bash navržen tak, aby možnost paralelního zpracování využil. Pro testování jsem napsala tři skripty:

Inicializace testování: Pomocí prvního ze skriptů je vytvořen pracovní soubor, který obsahuje právě tolik řádků, kolik má být běhů v průběhu nastaveného testování. Na každém řádku jsou informace pro nastavení daného běhu, jako jméno souboru s trénovacími daty a nastavení parametrů evoluce či koevoluce. Pokud jsou testována tři různá nastavení na třech funkcích, pro každé nastavení 50 běhů, obsahuje soubor 450 řádků.

Inicializace vláken pro testování: Tento skript spustí tolik testovacích vláken, kolik je mu zadáno parametrem. Na testovacím systému pro řešení bez užití koevoluce, které využívá pouze jedno vlákno, je možné spustit až 24 testovacích vláken. Řešení s koevolucí využívá dvě vlákna, a proto jsem omezila počet spuštěných testovacích vláken maximálně na 12.

Testování: Samotný skript, který zajišťuje spuštění jednoho běhu, funguje následovně:

1. Načte řádek z pracovního souboru a tento řádek smaže. Pro čtení a mazání v pracovním souboru je zajištěn výhradní přístup pomocí operací souborového systému, neboť testovacích vláken může být spuštěno současně několik.
2. Podle nastavení uvedeného v načteném řádku spustí program pro řešení symbolické regrese a výstup programu uloží do souboru pojmenovaného podle nastavení.
3. Po dokončení testu pokračuje bodem 1, dokud pracovní soubor není prázdný.

6.2 Způsob výpočtu fitness

V předchozích kapitolách byly zmíněny dva způsoby výpočtu fitness, jejichž efektivnost byla zkoumána v rámci testování řešení bez užití koevoluce. Při co nejvýhodnějším nastavení evoluce kandidátních řešení bylo zjištěno:

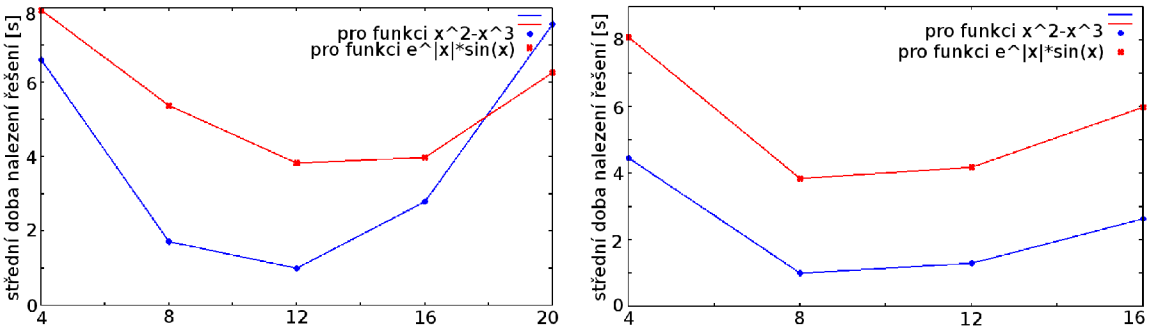
Střední absolutní odchylka: Při použití výpočtu fitness kandidátního řešení pomocí určení střední absolutní odchylky dané vztahem 2.1 se v souhrnu pro všechny testovací úlohy objevila pouze 19,3% úspěšnost. Pro zbylé běhy se pro tyto testovací úlohy objevilo uváznutí v lokálním extrému. V průběhu osmi milionů generací se nepodařilo přiblížit fitness řešení, která by splnila podmínku minimální povolené odchylky.

Skóre: Výpočtem fitness kandidátních řešení pomocí skóre, daným vztahem 2.2, se podařilo dosáhnout, v souhrnu za všechny tři testovací úlohy, úspěšnosti nalezení řešení 94 %. Zvýšení úspěšnosti nalezení řešení bylo pro mě dostačujícím faktorem pro výběr způsobu určení fitness založeném na výpočtu skóre kandidátního řešení. Tento způsob dále využívám při testování řešení s koevolucí i bez koevoluce.

6.3 Nastavení řešení bez koevoluce

Jako výchozí nastavení evoluce kandidátních řešení bez použití koevoluce jsem zvolila jeden řádek uzlů kartézského programu, parametr L-back rovný počtu sloupců uzlů v kartézském programu a maximálně 8 milionů generací, po které se hledá řešení. Pokud je nalezeno postačující řešení dříve než za 8 milionů generací, evoluce se ukončí a je vypsáno nalezené řešení. Pokud po 8 milionech generací není nalezeno postačující řešení, evoluce se ukončí a je vypsáno nejlépe ohodnocené kandidátní řešení. Parametr L-back nastavený na počet sloupců a jeden řádek kartézského programu umožňují maximální propojení acyklického orientovaného grafu. Při tomto nastavení mohou být vstupy uzlů připojeny k výstupu libovolného předcházejícího uzlu. Každý uzel má dva vstupy i_1 a i_2 a vykonává nad nimi jednu z operací: $i_1 + i_2$, $i_1 - i_2$, $i_1 \cdot i_2$, $\frac{i_1}{i_2}$, $i_1 + i_2$, $\sin(i_1)$, $\cos(i_1)$, e^{i_1}, $\log(i_1)$, $|i_1|$.

Nastavení parametrů počet sloupců kartézského programu, počet mutací na nově vznikajícího potomka a počet jedinců v populaci kandidátních řešení jsou testovány, aby bylo nalezeno co nejvíce výhodné nastavení. Pro každý parametr jsem zvolila několik možných hodnot a při testování se zkoušely jejich kombinace.



(a) Velikost populace kandidátních řešení.

(b) Maximální počet mutací.

Obrázek 6.1: Zobrazení závislostí testovaných parametrů na střední době nalezení postačujícího řešení pro funkci $f(x) = x^2 - x^3$ (modře) a funkci $f(x) = e^{|x|}\sin(x)$ (červeně) při co nejvýhodnějším nastavení ostatních parametrů.

Velikost chromozomu kandidátního řešení: Jelikož je počet řádků nastaven na jeden, je zkoumán pouze počet sloupců v kartézském programu. Zkoumaný počet sloupců byl postupně nastaven na 16, 24, 32, 48, 64, 96 a 128.

Většinou byla rozhodujícím kritériem pro výběr co nejvýhodnějšího nastavení daného parametru doba výpočtu potřebná pro nalezení postačujícího řešení. V případě funkce $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$ však nastávaly případy, kdy řešení nebylo nalezeno v průběhu osmi milionů generací. Proto je při výběru délky chromozomu rozhodující nastavení, při kterém k řešení konvergoval největší počet běhů. Nejvíce konvergovalo ke správnému řešení 82 % běhů pro tuto funkci při nastavení 32 uzlů v chromozomu. Při velikosti chromozomu větším než 32 už v případě funkce $f(x) = x^2 - x^3$ i funkce $f(x) = e^{|x|}\sin(x)$ výrazně rostla průměrná doba potřebná k nalezení řešení, zatímco při menším počtu uzlů v chromozomu pouze mírně klesala.

Počet jedinců v populaci kandidátních řešení: Při hledání co nejvýhodnějšího počtu jedinců v populaci řešení byl počet jedinců postupně nastaven na 4, 8, 12, 16 a 20. Zjišťovala se závislost velikosti populace na středním čase (medián z 50 běhů) potřebném k nalezení řešení. V případě všech funkcí mělo nejlepší střední čas 12 jedinců v populaci kandidátních řešení. Na obrázku 6.1a[1] je zobrazen průběh pro funkci $f(x) = x^2 - x^3$ (modře) a funkci $f(x) = e^{|x|}\sin(x)$ (červeně) při co nejvíce výhodném nastavení ostatních parametrů.

Maximální počet mutací na nově vznikajícího potomka: Při hledání co nejvýhodnějšího maximálního počtu mutací na nově vznikajícího potomka byl maximální počet mutací postupně nastaven na 4, 8, 12 a 16. Zjištění co nejvýhodnějšího nastavení maximálního počtu mutací probíhalo sledováním závislosti maximálního počtu mutací na středním čase (medián z 50 běhů) potřebném k nalezení řešení. V případě všech funkcí mělo nejlepší střední čas 8 mutací na jedince pro 32 uzlů v chromozomu jedince. Na obrázku 6.1b je zobrazen průběh pro funkci 1 (modře) a funkci 2 (červeně) při co nejvíce výhodném nastavení ostatních parametrů.

[1] Na tomto a následujících grafech (obrázky 6.1a, 6.1b, 6.2a, 6.2b, 6.2c, 6.4b, 6.4a) jsou zobrazeny diskrétní hodnoty, které jsou spojeny čarami pro znázornění vývoje trendů sledovaných parametrů vzhledem ke střední době nalezení postačujícího řešení.

Na základě těchto testů jsem jako co nejvýhodnější nastavení řešení bez koevoluce vybrala následující hodnoty parametrů:

- 12 jedinců v populaci kandidátních řešení,
- 32 uzlů (1 řádek a 32 sloupců) v chromozomu kandidátního řešení,
- maximálně 8 mutací na nově vznikajícího potomka.

6.4 Nastavení řešení s koevolucí

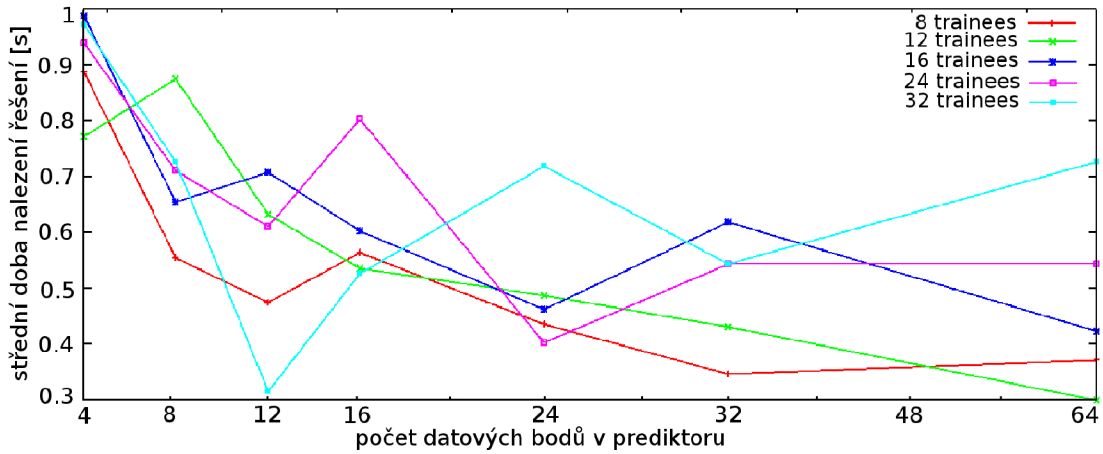
Jako výchozí nastavení hledání řešení symbolické regrese s použitím koevoluce jsem zvolila co nejvýhodnější nastavení evoluce kandidátních řešení (popsáno v části 6.3), zpomalení evoluce prediktorů 100krát (jedna iterace evolučního cyklu prediktorů za 100 iterací evolučního cyklu kandidátních řešení) a vytvoření nových trainers jednou za 500 iterací evolučního cyklu kandidátních řešení.

Při reprodukci populace prediktorů jsou vždy pro turnajový výběr vybráni dva jedinci, lepší z nich se stává rodičem. Vždy ze dvou vybraných rodičů jsou vytvořeni dva potomci pomocí jednobodového křížení. S pravděpodobností $0,2$ je nově vzniklému potomkovi náhodně mutován jeden gen. Populace prediktorů je nahrazena potomky, ale zůstává i nejlépe ohodnocený prediktor z minulé generace.

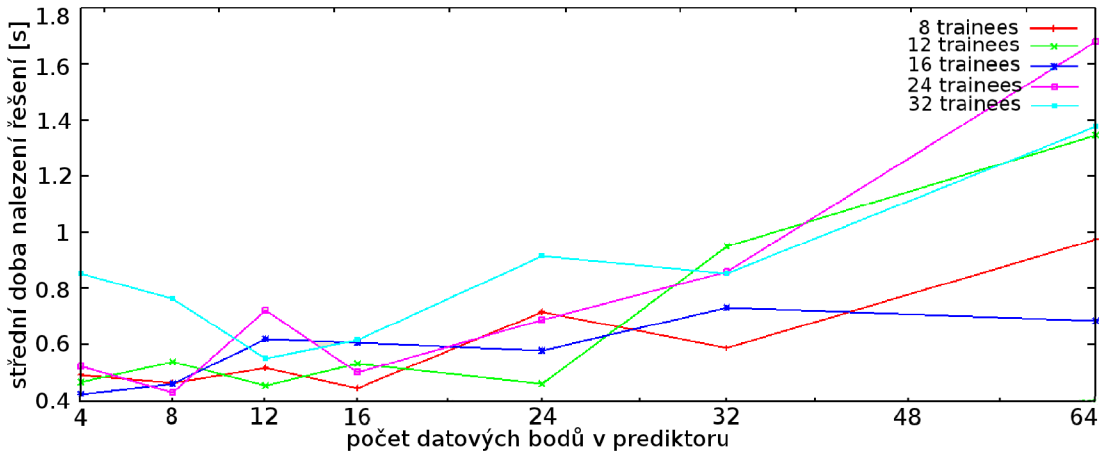
Při zjištění co nejvýhodnějšího nastavení řešení s užitím koevoluce bylo experimentováno s počtem jedinců v množině trainers, počtem jedinců v populaci prediktorů a velikostí prediktoru. Pro každý parametr jsem zvolila několik možných hodnot a při testování se zkoušely jejich kombinace.

Počet jedinců v množině trainers: Množina trainers obsahuje kandidátní řešení. Tato kandidátní řešení musí každý prediktor ohodnotit, aby bylo možné určit jeho fitness. Pro experimenty jsem zvolila 8, 12, 16, 24 a 32 jedinců v množině trainers. Zvýšení počtu trainers zvýší počet ohodnocení datových bodů pro každou generaci prediktorů, současně však zvýší přesnost ohodnocení prediktoru a tím vhodný prediktor může být nalezen v menším počtu generací. Na obrázcích 6.2 je vidět, že pro velikost chromozomu prediktoru 16 je v případě obrázku 6.2a nejvhodnějších 12 jedinců v množině trainers, v případě dalších dvou obrázků 6.2b a 6.2c je to 8 jedinců v množině trainers.

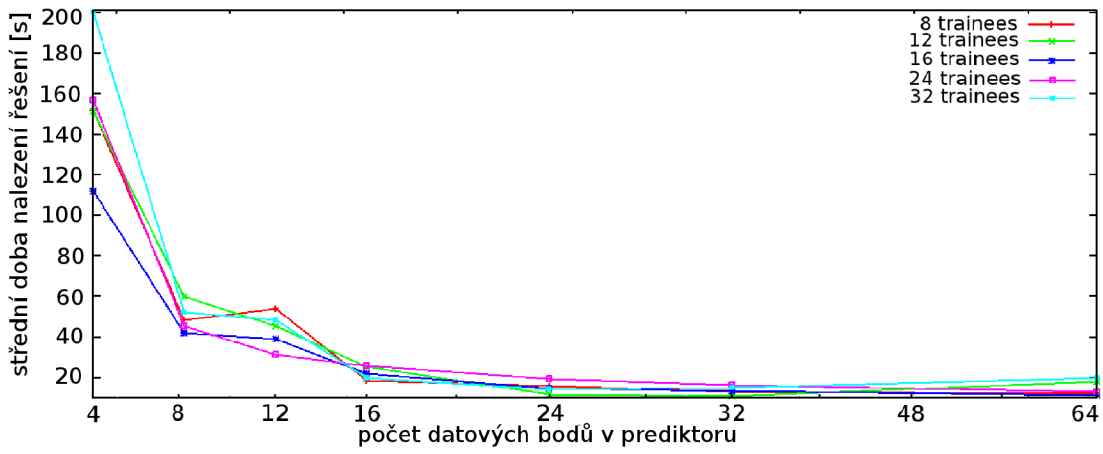
Velikost chromozomu prediktoru fitness: Při hledání co nejvýhodnější velikosti chromozomu prediktoru fitness byla velikost prediktoru postupně nastavena na 4, 8, 12, 16, 24, 32 a 64 datových bodů. Jak je vidět z obrázků 6.2, vhodná velikost prediktoru závisí na složitosti průběhu hledané funkce. Tyto obrázky znázorňují závislost velikosti prediktoru na střední době potřebné k nalezení řešení. Na obrázku 6.2b je vidět, že od velikosti 16 datových bodů již doba výpočtu roste, zatímco na obrázku 6.2c od velikosti 16 datových bodů stále mírně klesá. V případě méně složité funkce stačí pro vhodný prediktor méně datových bodů, které charakterizují funkci. V případě funkce se složitějším průběhem může zvětšení velikosti prediktoru pomoci nalézt prediktor s jehož využitím lze získat správné řešení funkce za menší počet generací, který se může promítnout i ve snížení doby výpočtu.



(a) $f(x) = x^2 - x^3$.



(b) $f(x) = e^{|x|}\sin(x)$.



(c) $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$.

Obrázek 6.2: Zobrazení závislosti velikosti prediktoru na střední době, z 50 běhů programu, potřebné k nalezení řešení pro různé velikosti populace trainers.

Počet jedinců v populaci prediktorů: Při zjišťování co nejvýhodnějšího počtu jedinců v populaci prediktorů fitness jsem pro experimenty zvolila 8, 12, 16, 24, 32, 48, 64, 96 a 128 jedinců. Při testování jsem zjistila, že každá z řešených úloh má toto nastavení různé. Nízký počet jedinců v populaci prediktorů způsobuje menší rozmanitost prediktorů, a proto nalezení vhodného prediktoru trvá více generací. Pokud má hledaná funkce jednodušší průběh, stačí méně jedinců v populaci prediktorů. V takovém případě existuje větší množství vhodných podmnožin trénovacích datových bodů. V případě funkce se složitým průběhem je prediktorů vystihujících průběh funkce podstatně méně, a proto evoluci trvá déle takový prediktor najít. Snížení počtu generací, za které je vhodný prediktor možné nalézt, lze dosáhnout zvýšením počtu jedinců v populaci prediktorů. Průměrem ze všech běhů pro všechny testované funkce vyšlo nejlépe 32 jedinců v populaci prediktorů fitness. Dále 32 jedinců v populaci kandidátních řešení bylo nejlepší variantou pro druhou testovací funkci $f(x) = e^{|x|}\sin(x)$. Funkce $f(x) = x^2 - x^3$ má jednodušší průběh a stačilo jí pouze 8 jedinců v populaci prediktorů, zatímco funkce $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$ se složitým průběhem našla řešení symbolické regrese v nejkratším čase při 64 jedincích v populaci prediktorů. Dále je použito 32 jedinců v populaci prediktorů při zobrazování vlastností řešení a zkoumání ostatních nastavení. Mým cílem je najít jedno kompromisní nastavení tak, aby pomocí tohoto nastavení se našlo řešení všech tří funkcí v co nejnižším čase. V kapitole 7 pak uvádím doporučení pro vývoj práce směrem k dynamickému hledání nejvýhodnějších nastavení souběžně s průběhem koevoluce.

Na základě těchto experimentů jsem pro řešení s koevolucí vybrala následující hodnoty parametrů:

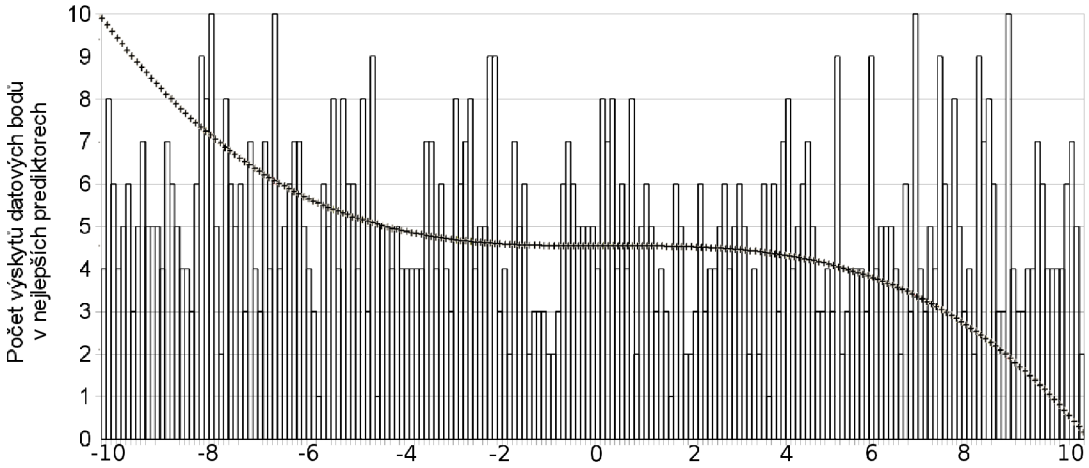
- 8 jedinců v populaci trainers,
- 16 datových bodů v prediktoru,
- 32 jedinců v populaci prediktorů.

6.5 Četnost datových bodů v nejlepších prediktorech

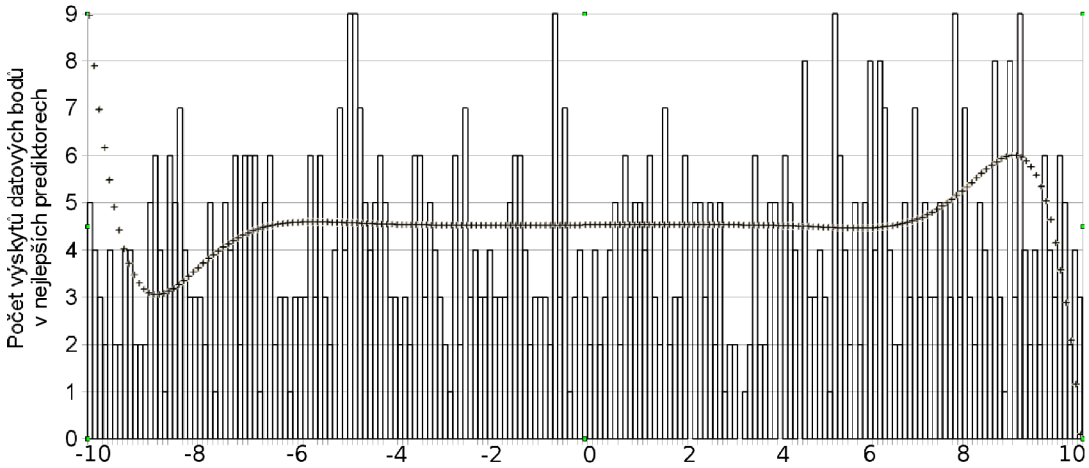
Při experimentování jsem zkoumala i složení prediktorů, které pomohly nalezení postačujícího řešení. Výběr datových bodů v prediktoru je důležitý pro správné ohodnocení kandidátních řešení. Podmnožin datových bodů, které charakterizují hledanou funkci, je více. Vybrané podmnožiny v nejlépe ohodnocených prediktorech často vystihují průběh funkce pomocí datových bodů vyskytujících se v okolí lokálních extrémů či inflexních bodů. Datové body vybrané v prediktoru nejsou většinou vzájemně sousedící, ale jsou rozloženy po celém průběhu trénovacích datových bodů.

Při testování jsem používala trénovací datové množiny o 201 datových bodech, z nichž bylo vždy 16 vybráno pro prediktor fitness.

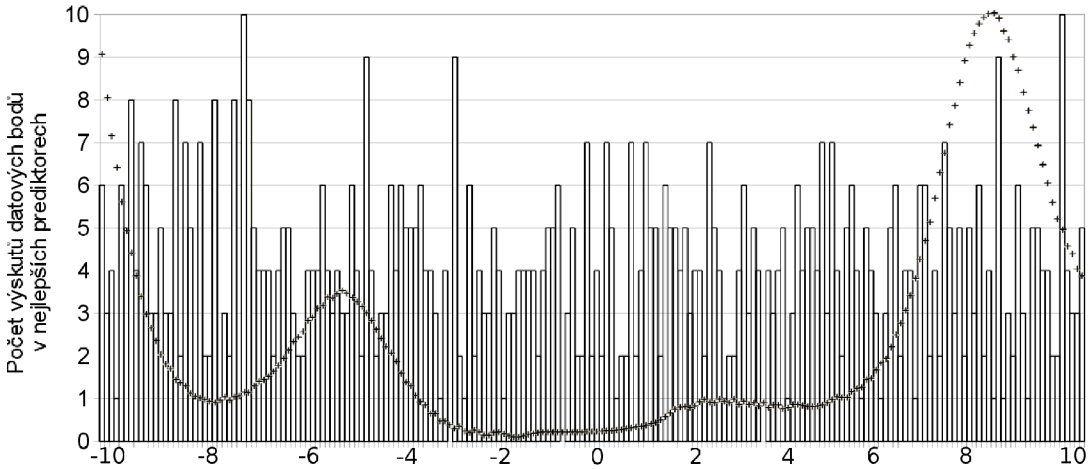
Na obrázcích 6.3 je zobrazena četnost jednotlivých datových bodů z prediktorů, pomocí kterých se podařilo najít řešení symbolické regrese, za 50 běhů programu pro každou z testovaných funkcí.



(a) $f(x) = x^2 - x^3$.

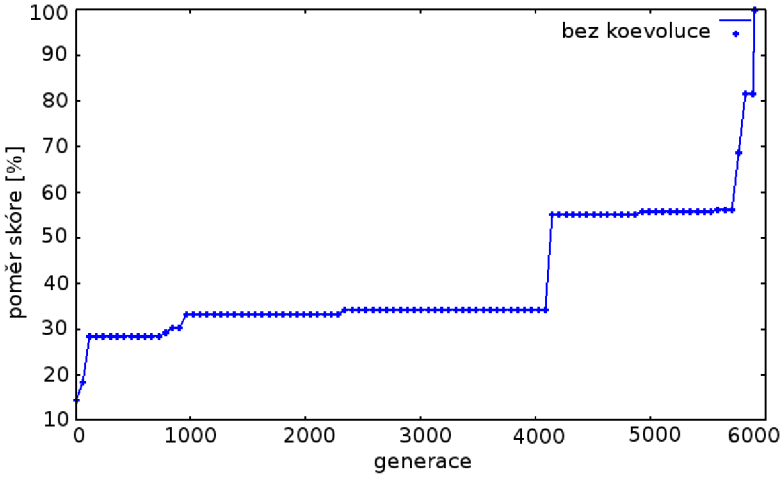


(b) $f(x) = e^{|x|} \sin(x)$.

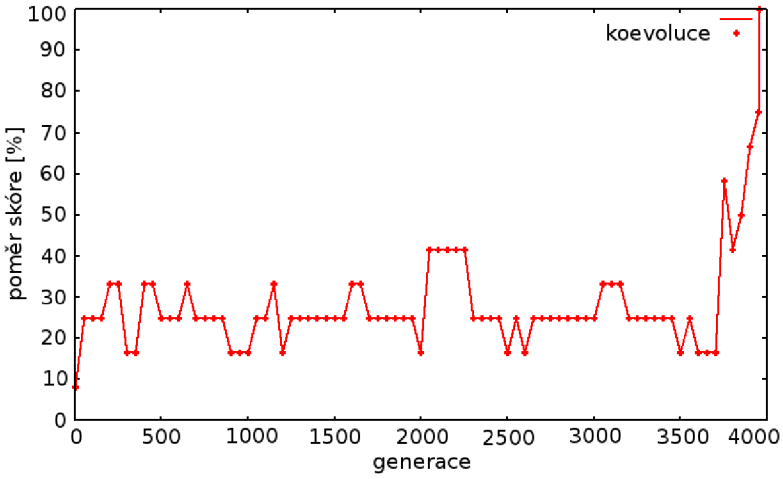


(c) $f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$.

Obrázek 6.3: Zobrazení četnosti datových bodů v prediktorech, pomocí kterých se podařilo nalézt postačující řešení symbolické regrese, za 50 běhů pro každou testovanou funkci. Z 200 datových bodů bylo do prediktoru vybráno vždy 16 datových bodů. V grafech je pro názornost ilustrativně zobrazen průběh datových bodů.



(a) Řešení bez koevoluce.

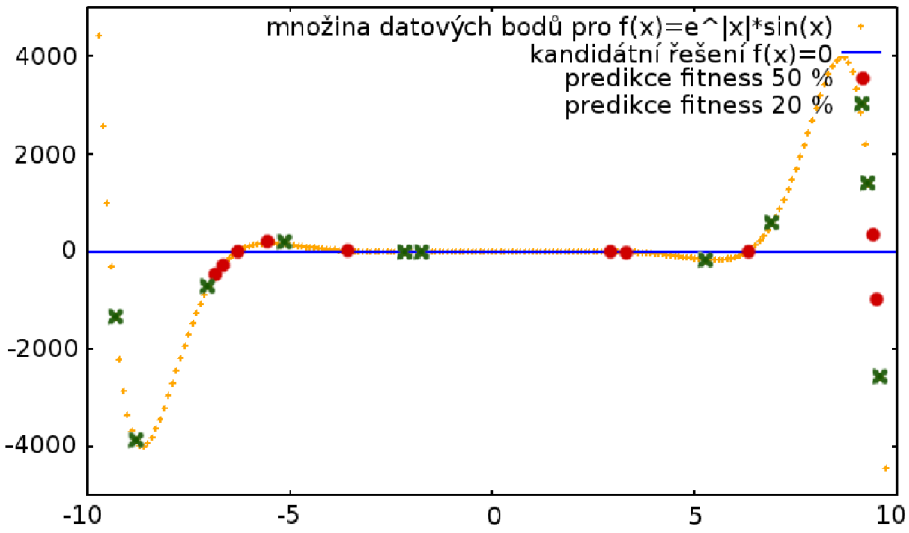


(b) Řešení s užitím koevoluce.

Obrázek 6.4: Zobrazení fitness nejlépe ohodnoceného kandidátního řešení v průběhu evoluce za jeden běh programu.

6.6 Vývoj fitness

Vývoj fitness kandidátních řešení v průběhu koevoluce není rostoucí v celém intervalu, jak je tomu u řešení bez koevoluce – obrázek 6.4a. Z obrázku 6.4b je vidět, že fitness v průběhu evoluce roste a klesá v závislosti na změně prediktorů. Na obrázku 6.5 je ilustrován případ, kdy dojde k poklesu fitness. Kandidátní řešení jsou ohodnocována pomocí prediktoru, který nemusí správně charakterizovat průběh funkce, a tím se mohou upřednostňovat i nevhodná řešení. Při zvolení nového prediktoru jsou pak tato kandidátní řešení ohodnocena pomocí jiné podmnožiny datových bodů, a tak jeho fitness může klesnout.



Obrázek 6.5: Ilustrace různého ohodnocení téhož kandidátního řešení různými prediktory. Červeným kolečkem jsou vyznačeny datové body v prediktoru predikujícím fitness kandidátního řešení 50 %, zeleným křížkem jsou označeny datové body z prediktoru predikujícího fitness kandidátního řešení 20 %.

Kapitola 7

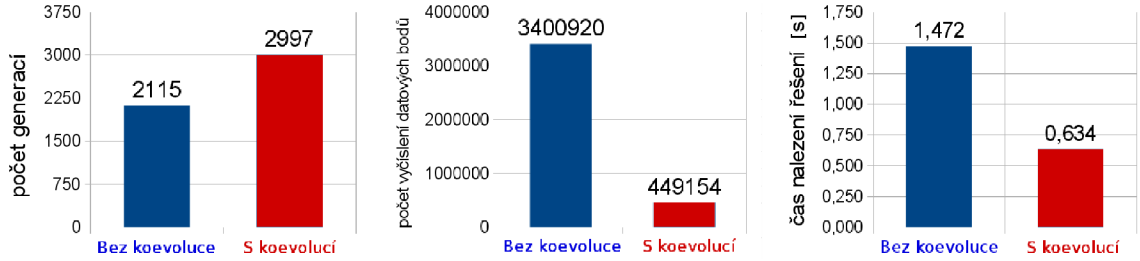
Souhrn výsledků

Optimální nastavení parametrů koevoluce závisí na konkrétním řešeném problému. Nastavení parametrů uvedená v této práci všechna konvergovala k nalezení správného řešení, avšak některá nastavení konvergovala výrazně pomaleji. Pro řešení složitějších úloh pro symbolickou regresi bych doporučovala tuto práci doplnit o dynamické hledání nejvýhodnějších nastavení souběžně s koevolucí. Tato optimalizace by mohla automatizovat nalezení vhodného nastavení souběžně s hledáním řešení.

Pro porovnání řešení s užitím koevoluce a bez koevoluce jsem vybrala jedno nastavení koevoluce a jedno nastavení evoluce kandidátních řešení pro všechny tři testované funkce:

- 12 jedinců v populaci kandidátních řešení,
- 32 uzlů (1 řádek a 32 sloupců) v chromozomu kandidátního řešení,
- maximálně 8 mutací na nově vznikajícího potomka,
- 12 jedinců v populaci trainers,
- 12 datových bodů v prediktoru,
- 32 jedinců v populaci prediktorů.

Porovnání pro funkci $f(x) = x^2 - x^3$: Na obrázku 7.1 lze vidět, že při použití koevoluce se navýšil počet potřebných generací k nalezení řešení, ale přesto se výrazně snížil počet ohodnocení datových bodů z trénovací množiny i čas potřebný k nalezení řešení.



Obrázek 7.1: Porovnání řešení bez koevoluce a s koevolucí funkce $f(x) = x^2 - x^3$ pro medián z 50 běhů.

Porovnání pro funkci $f(x) = e^{|x|}\sin(x)$: Na obrázku 7.2 je porovnání pro tuto funkci. Pomocí koevoluce se snížil počet generací a s tím i počet ohodnocení datových bodů z trénovací množiny i čas potřebný k nalezení řešení. Vybrané nastavení koevoluce nejvíce pomohlo urychlit nalezení řešení právě pro tuto funkci.

Obrázek 7.2: Porovnání řešení bez koevoluce a s koevolucí funkce $f(x) = e^{|x|}\sin(x)$ pro medián z 50 běhů.

Porovnání pro funkci $f(x) = x^2e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$: Porovnání výsledků koevoluce s výsledky bez koevoluce je na obrázku 7.3. S koevolucí se zvýšil počet potřebných generací, ale snížil se počet ohodnocení datových bodů z trénovací množiny a tím i čas potřebný k nalezení řešení. Navíc úspěšnost nalezení řešení při použití koevoluce byla 100%, zatímco při řešení bez koevoluce pouze 82%.

Obrázek 7.3: Porovnání řešení bez koevoluce a s koevolucí $f(x) = x^2e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$ pro medián z 50 běhů.

Koevolucí se tedy ve všech třech případech podařilo snížit frekvenci výpočtu fitness a současně i dobu potřebnou k nalezení řešení. Nejlépe se s tímto nastavením podařilo optimalizovat hledání funkce $f(x) = e^{|x|}\sin(x)$. Zbylé dvě funkce při jiném nastavení, než jsem vybrala, dosahovali i lepších výsledků. Pro funkci $f(x) = x^2e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right)$ při nastavení 12 jedinců v množině trainers, 32 datových bodů v prediktoru a 32 jedinců v populaci prediktorů byla střední doba potřebná k nalezení řešení 10,85 s. Funkce $f(x) = x^2 - x^3$ dosáhla střední doby potřebné k nalezení řešení 0,35 s při nastavení 8 jedinců v množině trainers, 32 datových bodů v prediktoru a 32 jedinců v populaci prediktorů.

7.1 Chování koevoluce

Na míru zrychlení řešení s užitím koevoluce mají vliv všechny tři sledované parametry:

Velikost prediktoru: S rostoucí velikostí prediktorů se zvyšuje přesnost predikce, ale roste počet ohodnocení datových bodů za jednu generaci.

Počet prediktorů: S vyšším počtem prediktorů je zajištěna genetická rozmanitost v populaci prediktorů a tím se urychlí nalezení vhodného prediktoru. Nižší počet prediktorů znamená menší počet ohodnocení datových bodů v průběhu evoluce prediktorů, ale pro některé úlohy je to dostačující. Pro zajištění genetické variability je v mém programu implementováno nahrazení nejhoršího jedince z populace prediktorů náhodným mutantem.

Počet trainers: S vyšším počtem trainers se zvyšuje přesnost určení fitness prediktoru, ale rozdíly jsou poměrně malé. Proto stačí menší počet trainers, čímž se zajistí menší množství ohodnocení datových bodů v průběhu ohodnocení prediktorů.

7.2 Porovnání s metodou pomocí stromového genetického programování

V článku [9] o koevoluci prediktorů fitness a genetického programování se nachází porovnatelné informace pro uvedenou testovací funkci $f(x) = e^{|x|}\sin(x)$. Z uvedených grafů lze vyčíst přibližné hodnoty pro průměrný počet generací potřebný k nalezení řešení a příslušný počet ohodnocení datových bodů. Pro nastavení velikosti prediktoru na 8 datových bodů je počet generací přibližně 1000 a počet vyčíslení datových bodů přibližně $5 \cdot 10^6$. Ve výše uvedeném článku používají pro genetické programování populaci kandidátních řešení o velikosti 128 jedinců. V mém řešení je nastaveno 12 jedinců z populace kandidátních řešení, protože kartézské genetické programování používá menší populace. Při nastavení velikosti prediktoru 8 datových bodů je pomocí mého programu řešení nalezeno průměrně za 2735 generací, ale počet potřebných datových bodů k vyčíslení je jen $3 \cdot 10^5$.

Kapitola 8

Závěr

Výsledkem této práce je funkční program pro řešení symbolické regrese implementovaný pro použití v prostředí operačního systému Linux. Řešení symbolické regrese funguje na principu kartézského genetického programování a koevoluce. Pomocí koevoluce se podařilo snížit počet vyhodnocení datových bodů a tím i celkovou dobu výpočtu oproti řešení bez užití koevoluce. Doba potřebná k nalezení řešení symbolické regrese je závislá na několika parametrech nastavení koevoluce. Tyto parametry byly předmětem testování s výsledkem, že nejvýhodnější nastavení těchto parametrů je pro každou z testovaných funkcí odlišné. Při porovnání stejného nastavení pro všechny testovací funkce bylo pomocí koevoluce dosaženo minimálně 2,3násobného zrychlení oproti řešení bez koevoluce, při co nejvýhodnějším nastavení téměř 11násobného zrychlení.

Pro ohodnocení kandidátních řešení jsem použila dva způsoby vyhodnocení fitness. Prvním způsobem bylo spočítání střední absolutní chyby bodů, které jsou v trénovací množině dat, a které generuje kandidátní řešení. Tento způsob výpočtu fitness se ukázal jako nevhodný, protože pouze 19,3 % běhů programu při použití bez koevoluce našlo pro testované funkce řešení. Druhým způsobem výpočtu ohodnocení kandidátního řešení bylo ohodnocení pomocí přidělování bodů (skóre) za správný výsledek kandidátního řešení pro datový bod s povolenou odchylkou. Tento způsob dosáhl úspěšnosti nalezení řešení v 94 % bez použití koevoluce. S použitím koevoluce se zvedla úspěšnost nalezení řešení na 100 % pro všechny tři testované funkce.

Pro řešení složitějších úloh by bylo vhodné rozšířit práci o dynamické hledání optimálního nastavení souběžně s koevolucí. Tato optimalizace by mohla pomoci automatizovat nalezení vhodného nastavení souběžně s hledáním řešení.

Pro porovnání koevoluce prediktorů fitness s kartézským genetickým programováním a koevoluce prediktorů fitness se stromovým genetickým programováním by bylo vhodné implementovat modul pro řešení symbolické regrese pomocí stromového genetického programování, který by fungoval společně s moduly pro koevoluci prediktorů fitness.

Při zpracování tohoto tématu jsem si zopakovala a nastudovala modely evolucí inspirovaných výpočetních metod a řešení symbolické regrese a s některými z nich jsem se seznámila i prakticky při implementaci.

Prezentace této práce se umístila jako první v kategorii magisterských projektů Inteligentní systémy na konferenci a soutěži STUDENT EEICT 2011.

Literatura

[1] Herout, P.: *Učebnice jazyka C – 2. díl.* KOPP, 2004, ISBN 80-7232-221-4.

[2] Janzen, D.: When is it coevolution? *Evolution*, ročník 34, č. 3, 1980: s. 611–612, ISSN 0014-3820.

[3] Koza, J.: *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems.* Citeseer, 1990.

[4] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* STU, 2000, ISBN 800-227-1377-5.

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

[6] Miller, J. M.; Thomson, P.: Cartesian Genetic Programming. *In Proc. of the Third European Conference on Genetic Programming (EuroGP2000)*, LNCS 1820, Springer, 2000: s. 121–132.

[7] Satola, R.: Genetické algoritmy. *Aplikované evoluční algoritmy – EVO, přednášky k předmětu*, FIT VUT v Brně, 2004.

[8] Schmidt, M.; Lipson, H.: Distilling Free-Form Natural Laws from Experimental Data. *SCIENCE*, ročník 324, 2009: s. 81–85.

[9] Schmidt, M. D.; Lipson, H.: Coevolution of Fitness Predictors. *IEEE Transactions on Evolutionary Computation*, ročník 12, č. 6, 2008: s. 736–749, ISSN 1089-778X.

[10] Schwarz, J.: Návrh a ladění genetických algoritmů. *Aplikované evoluční algoritmy – EVO, přednášky k předmětu*, FIT VUT v Brně, 2008.

[11] Sekanina, L.: Genetické programování. *Aplikované evoluční algoritmy – EVO, přednášky k předmětu*, FIT VUT v Brně, 2009.

[12] Sekanina, L.: Evoluční design. *Biologií inspirované počítače – BIN, přednášky k předmětu*, FIT VUT v Brně, 2010.

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

[14] Zelinka, I.; Oplatková, Z.; Šeda, M.; aj.: *Evoluční výpočetní techniky – principy a aplikace.* BEN - technická literatura, 2009, ISBN 978-80-7300-218-3.

Příloha A

Obsah CD

Písemná zpráva

Písemná zpráva ve formátu pdf je na přiloženém CD v adresáři `pisemna_zprava/` a její zdrojový text pro LaTeX v `pisemna_zprava/src`. Grafy, diagramy a jiné obrázky použité ve zprávě jsou v `pisemna_zprava/pictures`.

Program řešící symbolickou regresi

Programy řešící symbolickou regresi jsou v adresáři `SR/`. Tento adresář obsahuje i soubor `README`, ve kterém se nachází stručný popis programů a návod na použití. Zdrojové kódy programu, který řeší symbolickou regresi, jsou v `SR/src/`.

Trénovací data

Použitá trénovací data pro všechny tři použité funkce jsou spolu s generátorem trénovacích dat umístěna v adresáři `data/`. Stručný popis a návod na použití se nachází tamtéž v souboru `README`

Testovací systém

Testovací skripty se nachází včetně návodu na použití v adresáři `test/`. Návod na použití testovacích skriptů se nachází v tomto adresáři v souboru `README`. Výstupy provedených testů jsou uloženy v adresáři `test/outputs/` a pojmenovány podle nastavení evoluce či koevoluce při testování.