VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOEVOLUČNÍ ALGORITMUS PRO ÚLOHY ZALOŽENÉ NA TESTU

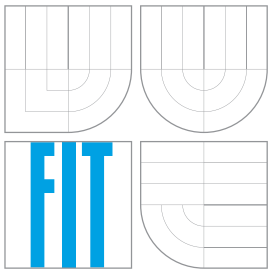
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ HULVA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOEVOLUČNÍ ALGORITMUS PRO ÚLOHY ZALOŽENÉ NA TESTU

COEVOLUTIONARY ALGORITHM FOR TEST-BASED PROBLEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ HULVA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAELA ŠIKULOVÁ

BRNO 2014

Abstrakt

Tato práce se zabývá využitím koevoluce při řešení symbolické regrese. Symbolická regrese se používá pro zjištění matematického vztahu, který aproximuje naměřená data. Lze ji provádět pomocí genetického programování - metody ze skupiny evolučních algoritmů inspirovaných evolučními procesy v přírodě. Koevoluce pracuje s několika vzájemně působícími evolučními procesy. V této práci je popsán návrh a implementace aplikace, která dokáže provádět symbolickou regresi pomocí koevoluce pro úlohy založené na testu. Testy jsou generovány novou metodou, která umožňuje dynamicky měnit počet trénovacích vektorů potřebných k ohodnocení kandidátních řešení. Funkčnost aplikace byla ověřena na pěti testovacích úlohách. Výsledky byly porovnány s koevoluční metodou pracující s fixním počtem trénovacích vektorů. U tří úloh nalezla nová metoda řešení požadované kvality během menšího počtu generací, většinou ale bylo potřeba provést více vyčíslení trénovacích vektorů.

Abstract

This thesis deals with the usage of coevolution in the task of symbolic regression. Symbolic regression is used for obtaining mathematical formula which approximates the measured data. It can be executed by genetic programming - a method from the category of evolutionary algorithms that is inspired by natural evolutionary processes. Coevolution works with multiple evolutionary processes that are running simultaneously and influencing each other. This work deals with the design and implementation of the application which performs symbolic regression using coevolution on test-based problems. The test set was generated by a new method, which allows to adjust its size dynamically. Functionality of the application was verified on a set of five test tasks. The results were compared with a coevolution algorithm with a fixed-sized test set. In three cases the new method needed lesser number of generations to find a solution of a desired quality, however, in most cases more data-point evaluations were required.

Klíčová slova

Koevoluční algoritmy, symbolická regrese, evoluční algoritmy, kartézské genetické programování

Keywords

Coevolutionary algorithms, symbolic regression, evolutionary algorithms, cartesian genetic programming

Citace

Jiří Hulva: Koevoluční algoritmus pro úlohy založené na testu, diplomová práce, Brno, FIT VUT v Brně, 2014

Koevoluční algoritmus pro úlohy založené na testu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michaely Šikulové

.....
Jiří Hulva
27. května 2014

Poděkování

Timto bych rád poděkoval své vedoucí Michaele Šikulové za její vstřícnost a bezmeznou trpělivost u konzultací a také za spoustu cenných připomínek, rad a doporučení.

Obsah

Kapitola 1

Úvod

Snaha o tvorbu co nejefektivnějších algoritmů je stejně stará jako informační technologie samy. I když dostupný výpočetní výkon každým rokem roste exponenciálním tempem, stále existují problémy, jejichž řešení hrubou silou je jednoduše nemyslitelné - prostor možných řešení, který je třeba prohledat, je příliš obrovský. Zde přichází ke slovu nejrůznější heuristické metody, které chytře pátrají po řešení, zaměřují svou pozornost do slibných oblastí a s trochou štěstí naleznou řešení mnohem rychleji. Mezi ty úspěšné patří skupina evolučních algoritmů, jejichž mechanismy vycházejí z teorie evoluce druhů. Evoluční algoritmy doznaly velkého rozmachu v posledních desetiletích a byly s úspěchem použity u mnoha úloh z oblasti optimalizace, prohledávání anebo také inženýrského návrhu. V poslední době probíhá v této oblasti rozsáhlý výzkum. Vznikají nové techniky a vylepšení, díky kterým evoluční algoritmy pracují efektivněji a mohou být použity i u úloh, se kterými měly dříve problémy. Mezi tato rozšíření patří i koevoluční algoritmy.

Cílem této práce je seznámit se s problematikou genetického programování, koevolučních algoritmů a symbolické regrese. Dále bude navržena a implementována aplikace, která bude schopna řešit problém symbolické regrese pomocí koevolučního algoritmu pro úlohy založené na testu. Testy pro ohodnocení kandidátních řešení v úloze symbolické regrese budou generovány novou, dosud nevyzkoušenou metodou - pomocí evolučního mechanismu založeného na principu symbolické regrese. Aplikace bude otestována na sadě vybraných úloh. Na základě dosažených výsledků bude tento nový přístup porovnán se standardně používanou metodou.

Kapitola 2 je úvodem do problematiky evolučních algoritmů. Jsou zde vysvětleny základní pojmy a principy, které jsou společné všem dále uvedeným technikám spadajícím do skupiny evolučních algoritmů. Část 2.1 se blíže věnuje *genetickému programování*, evoluční technice používané mimo jiné i při řešení symbolické regrese. Je zde objasněna podstata genetického programování, její výhody, ale i problémy, se kterými se potýká. Část 2.1.1 se zabývá *Kartézským genetickým programováním*, které z klasického genetického programování vychází. Podkapitola 2.2 je o koevoluci. Jsou zde uvedeny situace, za kterých je výhodné koevoluci použít, a dále jsou zde rozebrány rozličné typy koevolučních algoritmů. V části 2.2.1 se blíže věnuji problematice predikce fitness, což je typ úlohy, na jejíž řešení bude v této práci koevoluce použita. Podkapitola 2.3 popisuje symbolickou regresi, porovnává ji s jinými metodami regresní analýzy a přibližuje dále metody řešení symbolické regrese pomocí koevoluce. V části 2.3.3 je popis standardního koevolučního algoritmu pro úlohy založené na testu.

V kapitole 3 je představen návrh koevolučního algoritmu. Jsou zde blíže popsány jednotlivé části algoritmu, zejména pak rozdíly oproti schématu popsanému v části 2.3.3. Nej-

rozsáhlejší částí je podkapitola 3.6. Pojednává o novém způsobu evoluce prediktorů fitness. Je zde popsán navržený princip predikce a jeho výhody i možné nevýhody oproti dosavadní implementaci.

Podrobnosti o implementaci jednotlivých částí navrženého algoritmu jsou uvedeny v kapitole 4, včetně popisu použitých technologií.

Kapitola 5 obsahuje popis testů, pomocí nichž byly měřeny vlastnosti nové metody. Je zde uvedena množina úloh, na nichž testy probíhaly, a popsáno použité nastavení algoritmu. V části 5.4 jsou shrnuty výsledky experimentů dosažených pomocí koevoluce a uvedeno srovnání s výsledky bez koevoluce. Část 5.5 uvádí porovnání s výsledky dosaženými pomocí odlišné metody predikce fitness.

Kapitola 6 je závěrečným shrnutím dosažených výsledků.

Tato diplomová práce navazuje na semestrální projekt, v rámci něhož byla zpracována teoretická část práce v kapitole 2 a také navrženy a implementovány základní části koevolučního algoritmu.

Kapitola 2

Evoluční algoritmy

Jako evoluční algoritmy označujeme celou skupinu stochastických prohledávacích algoritmů, jejichž základní principy vycházejí z pozorovaných evolučních procesů v přírodě a z teorií neodarwinismu. Výzkum v této oblasti započal již v polovině dvacátého století. Jejich vývoj probíhal mnohdy odděleně, nezávisle na sobě v různých odvětvích vědy a průmyslu, a tak vzniklo hned několik proudů. Jmenujme například evoluční programování, genetické algoritmy, genetické programování a evoluční strategie. Pojem evoluční algoritmy vznikl až v 90. letech, aby všechny tyto směry zastřešil a sjednotil terminologii. Od počátku byly používány především u optimalizačních úloh - pro nalezení ideálních hodnot pevně daného počtu parametrů, kde se ukázaly jako vysoce účinné. V posledních letech se experimentuje s jejich použitím při návrhu nových součástek a programů, o čemž bude řeč později v kapitole o genetickém programování 2.1.

Evoluční algoritmy jsou charakteristické tím, že pracují s populací (množinou) *kandidátních řešení*. Každé z kandidátních řešení (neboli také jedinec) je zakódováno pomocí *genotypu* - typicky je to vektor hodnot (genů) - a představuje jedno z možných řešení zkoumané úlohy. Souvisejícím pojmem je *fenotyp*, což je synonymum pro kandidátní řešení problému, které můžeme sestavit podle dat obsažených v genotypu. Počáteční generaci je možné vytvořit buď zcela náhodně, anebo heuristicky - například tak, aby byli jedinci na začátku rovnoměrně rozprostřeni v prostoru možných řešení, nebo pomocí dalších znalostí o dané úloze odhadneme, jakých hodnot by řešení mohlo nabývat, a do těchto oblastí umístíme počáteční jedince. Další možností je použití již dříve získaných řešení problému jako jedinců pro počáteční populaci.

Jakmile máme počáteční populaci, může začít prohledávání. To probíhá v iteracích. V každé iteraci jsou všichni jedinci ve stávající generaci ohodnoceni a na základě tohoto ohodnocení (tzv. *fitness hodnoty*) je vybrána podmnožina jedinců, kteří se stanou „rodiči“. Z rodičů jsou pak pomocí genetických operátorů vytvořeni potomci. Rodiče a potomci dohromady tvoří novou generaci. Tento proces se opakuje tak dlouho, dokud nejsou splněny ukončující podmínky - například pokud je dosažen maximální počet iterací, ohodnocení jedinců dosáhlo požadované úrovně, anebo se nejlepší dosažená hodnota fitness po daný časový interval již nijak dále nezlepšila.

Důležitým pojmem je takzvaná *fitness*. Jedná se o hodnotu, která udává, jak kvalitní dané řešení je. Fitness lze definovat jako funkci f:

$$f : G \mapsto \mathbb{R} \tag{2.1}$$

G je množina všech genotypů. Zpravidla je otestováno chování kandidátního řešení zakódovaného genotypem na množině trénovacích dat a podle toho, jak dobře si vedlo, se mu

přidělí reálné číslo - fitness hodnota. Výpočet fitness hodnoty bývá často nejnáročnější částí algoritmu a na jeho optimalizaci se tato práce dále zaměří.

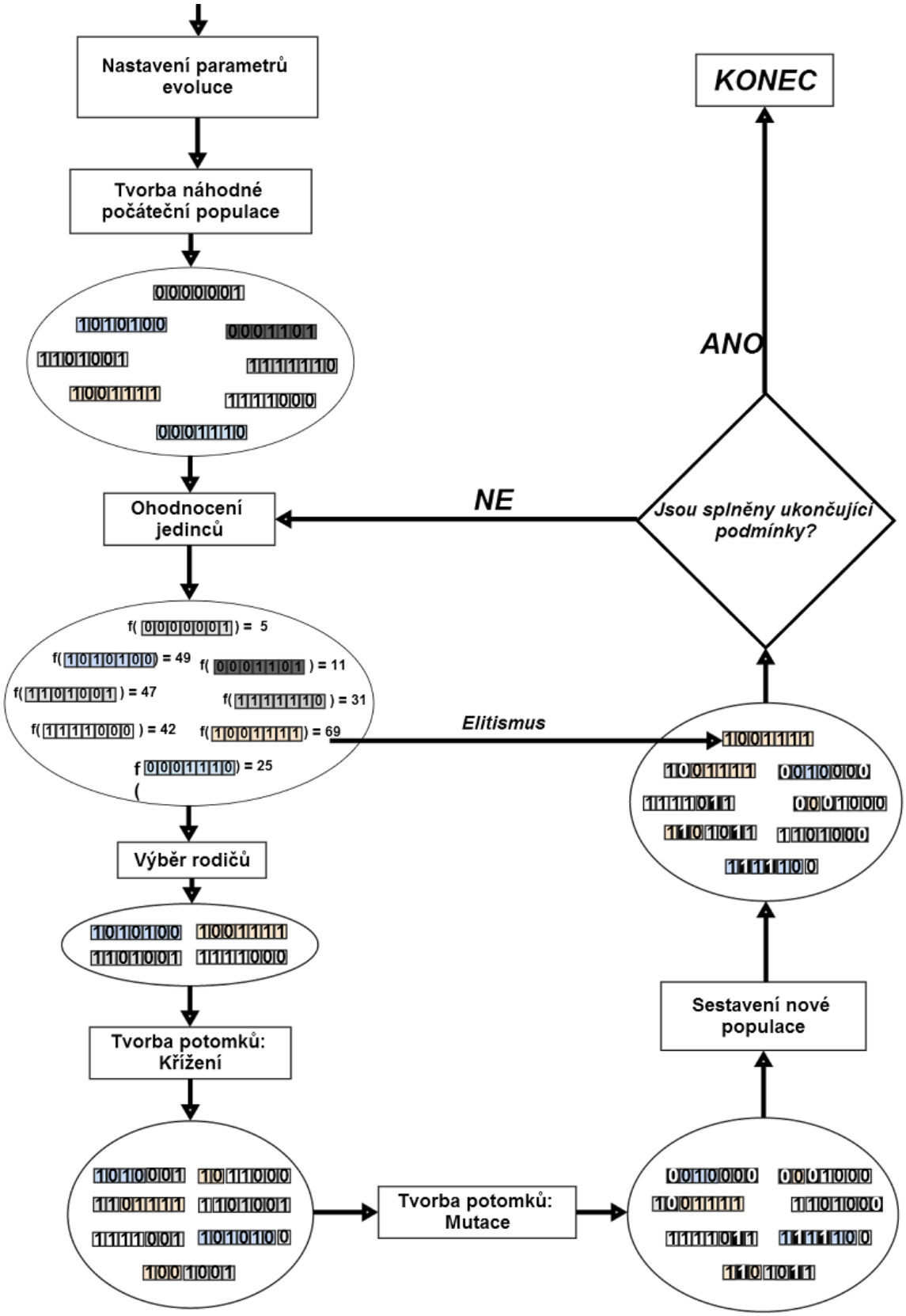
V duchu hesla „silnější přežije“ se na složení další generace podílejí především jedinci s vyšší fitness. Vzniká tak selekční tlak, kdy se proces prohledávání soustřeďuje do slibnějších oblastí stavového prostoru a průměrná fitness populace roste. Volbu jedinců je možné implementovat například pomocí tzv. ruletového výběru, kdy na hodnotě fitness závisí pravděpodobnost, že bude daný jedinec vybrán jako „rodič“. Další možností je turnajový výběr, kdy opakovaně náhodně zvolíme několik jedinců a rodičem se stane ten s nejvyšší fitness. Obě zmíněné metody umožňují, aby se do další generace dostaly i geny hůře ohodnocených jedinců, byť jejich šance jsou o poznání horší. To je velice důležité, protože to pomáhá udržet různorodost jedinců v populaci. V opačném případě by si jedinci začali být brzy příliš podobní a kombinací jejich genů bychom již nedokázali získat nějaké opravdu nové kandidátní řešení, pouze kopie těch stávajících. Proces evoluce by se tím v podstatě zastavil a prohledávání by uvázlo v lokálním optimu. Některé algoritmy přesto uchovávají alespoň část nejlepších nalezených řešení. Stanovený počet jedinců s nejvyššími hodnotami fitness má v takových případech zaručen postup do další generace. Tomuto principu se říká *elitismus*.

Jakmile jsou vybráni „rodiče“, můžeme z nich vytvořit novou generaci. Zde si evoluční algoritmy opět berou příklad z mechanismů vývoje živých organismů - nejběžněji používané operátory pro manipulaci s genotypy a tvorbu nových kandidátních řešení jsou křížení a mutace. Samotný mechanismus těchto operátorů se liší nejen mezi různými druhy evolučních algoritmů, ale také podle typu zkoumané úlohy a způsobu, jakým je kandidátní řešení v genotypu zakódováno.

Křížení: Inspiruje se v pohlavním rozmnožování živých organismů. Tento operátor vytvoří genotyp potomka kombinací částí genotypů jeho rodičů. V ideálním případě by tak potomek mohl zdědit dobré vlastnosti obou rodičů.

Mutace: Pracuje pouze s jediným jedincem. Náhodně změní hodnotu jednoho nebo více jeho genů. V populaci tak vytváří zcela nové náhodné motivy, které by nemohly vzniknout pouze křížením, a umožňuje tak evolučnímu algoritmu prozkoumat i jinak nedosažitelné oblasti prohledávaného prostoru řešení. Také přispívá k rozmanitosti populace. Jakmile si totiž začnou být jednotlivé genotypy navzájem příliš podobné, z populace se ztratí potřebná diverzita a hrozí, že evoluční proces uvázne v lokálním extrému. Operátor mutace se uplatňuje s určitou mírou pravděpodobností. Ta by neměla být příliš malá, aby se mutace vůbec nějak projevila, ale ani příliš vysoká. Vývoj populace by se tak stal nestabilní, přestal by konvergovat k nejlepšímu řešení a stalo by se z něj de facto náhodné prohledávání.

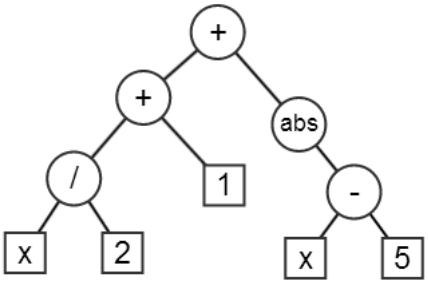
Na obrázku 2.1 je zobrazeno schéma evolučního algoritmu. V tomto ukázkovém příkladu má genotyp podobu vektoru binárních hodnot a je uplatněn elitismus pro nejlepšího jedince.



Obrázek 2.1: Schéma evolučního algoritmu.

2.1 Genetické programování

Genetické programování je podoborem evolučních algoritmů, který vznikl koncem 80. let. Nejvíce se výzkumu v této oblasti věnoval John Koza. Princip této metody se drží schématu představeného u evolučních algoritmů. Na začátku se náhodně vygeneruje počáteční populace a následuje cyklus, kdy se ze stávající generace vyberou nejlépe ohodnocení jedinci, ze kterých se příslušnými evolučními operátory vytvoří nová generace. Hlavním rozdílem oproti dřívějším typům evolučních algoritmů je fakt, že genetické programování neslouží pouze k nalezení optimálních parametrů, ale přímo vyvíjí celé spustitelné programy, a to tak, aby co nejpřesněji prováděly specifikovanou činnost. Genetické programování bylo s úspěchem uplatněno při symbolické regresi, u návrhu nových obvodů a součástek a také v oblasti umělé inteligence. Genotyp jedince může nabývat různé velikosti a reprezentuje zde spustitelný program, nejčastěji v podobě stromové struktury. Příklad zakódování fenotypu lze vidět na obrázku 2.2. Listy tohoto stromu - terminály - mohou být vstupy programu (tzv. primární vstupy), konstanty, anebo funkce bez parametrů. Uzly pak reprezentují funkce s určitým počtem vstupních parametrů. U funkcí je důležitý požadavek uzavřenosti - každá funkce musí být schopna jako vstup přijmout jakýkoli terminál nebo výstup libovolné funkce. Proto se zpravidla počítá s hodnotami v podobě desetinných čísel a používají se tzv. *chráněné varianty* funkcí. Množinu funkcí a terminálů je nutné specifikovat ještě před započetím evolučního procesu. Výpočet odezvy jedince, uvažujeme-li stromovou reprezentaci, spočívá v postorder-průchodu stromem a provedení všech funkcí v jeho uzlech. Výstupní hodnota pak odpovídá výstupu funkce v kořeni stromu.



Obrázek 2.2: Stromová reprezentace kandidátního řešení $y = \frac{x}{2} + 1 + |x - 5|$.

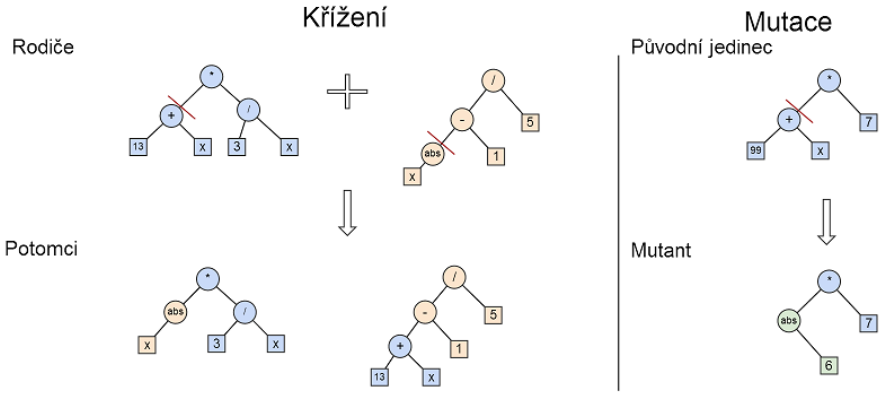
Pro určování fitness máme zpravidla definovánu množinu vstupních hodnot v, na které chceme funkci jedince otestovat. Ke každé vstupní hodnotě x_i máme přiřazen očekávaný výstup y_i. Tato dvojice se nazývá *trénovací vektor*. Fitness jedince lze pak definovat například jako sumu druhých mocnin rozdílů skutečných a očekávaných výstupů takto:

$$f = \sum_{i=1}^{N} (s(x_i) - y_i)^2 \tag{2.2}$$

kde y_i je výstup kandidátního programu s při vstupní hodnotě x_i [7]. Evoluční proces se snaží tuto hodnotu minimalizovat. Volba které trénovací vektory při ohodnocení použít a jaký má být jejich počet má zásadní vliv na to, zda dokáže algoritmus nalézt řešení požadované kvality. Jedna z možností jak tento problém řešit je nastíněna v části 2.2 o koevolučních algoritmech.

Genetické operátory

Proces výběru rodičů a vytváření nového jedince probíhá u genetického programování obdobně jako u genetických algoritmů. Opět jsou použity operátory křížení a mutace, upravené tak, aby pracovaly se stromovou reprezentací genotypu. Při křížení je u obou rodičů náhodně vybrán jeden uzel. Tyto uzly společně s podstromy, které na ně navazují, se poté prohodí a tím vzniknou dva noví jedinci. Operátor mutace pracuje obdobně - začne náhodným výběrem jednoho z uzlů. Podstrom, který na něj navazuje, je následně nahrazen novou náhodně vygenerovanou strukturou (obrázek 2.3). Kromě křížení a mutace zde mohou existovat i jiné specifické operátory, například pro tvorbu podprogramů, iteračních smyček, nebo rekurze.



Obrázek 2.3: Operátory křížení a mutace u GP.

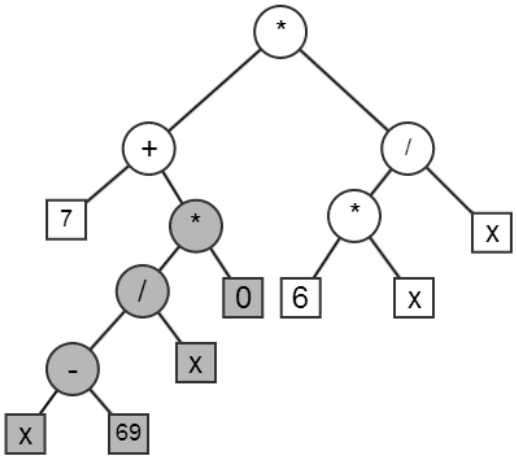
Bloat

U genetického programování se setkáváme s jevem zvaným *bloat*. Jde o to, že velikost kandidátních programů bývá sice zpočátku konstantní, v určité chvíli se začnou zvětšovat. Tento růst exponenciálně zrychluje, aniž by se nějak významně zlepšovala hodnota fitness. Ve struktuře programu se začínají objevovat části kódu, které neprovádějí žádnou užitečnou činnost - takzvané *introny*. Může jít například o výraz $a = a - 0$. Obrázek 2.4 zobrazuje stromovou reprezentaci programu, kde intronem je podstrom $\frac{x-69}{x} \cdot 0$.

Bloat se často považuje za negativní jev. S tím, jak postupně roste délka kandidátních programů, stává se jejich ohodnocení čím dál tím výpočetně náročnějším. Nejjednodušším a nejčastěji používaným protiopatřením je přičtení penalizace za délku programu k hodnotě fitness - metoda známá jako *Parsimony pressure* [4]. Dalším řešením může být přímo detekce a eliminace takovýchto částí kódu. Na druhou stranu existují teorie, které tvrdí, že introny mohou mít důležitou roli při ochraně užitečného kódu před vlivem křížení a mutace - úplně stejně jako jejich protějšky z oblasti biologie. Čím je program delší a čím více intronů obsahuje, tím menší je pravděpodobnost, že genetické operátory při svém náhodném výběru uzlů zasáhnou právě dovnitř důležitého bloku užitečného kódu a tím jej rozbijí [3].

2.1.1 Kartézské genetické programování

Kartézské genetické programování se dá považovat za variantu klasického genetického programování (GP), z něhož svými základními principy vychází. Jeho autorem je Julian Miller,



Obrázek 2.4: Jedinec postižený bloatem. Intron je označen šedě.

který ho v roce 1999 poprvé uceleně publikoval. Také zde se kandidátní řešení skládá ze vzájemně propojených uzlů. Tentokrát jsou však uzly uspořádány v obdélníkové mřížce, jejíž sloupce budeme označovat jako vrstvy.

Před započetím evolučního procesu se určí několik parametrů, které zůstanou v jeho průběhu neměnné:

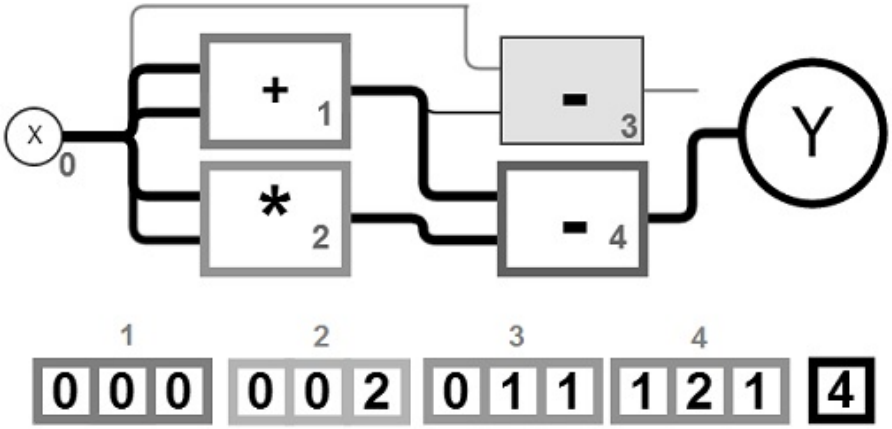
- rozměry mřížky uzlů – počet vrstev n_c a šířka vrstev n_r
- počet primárních vstupů jedince n_{in}
- počet primárních vstupů jedince n_{out}
- počet vstupů každého uzlu u_{in}
- množina funkcí, které mohou jednotlivé uzly vykonávat
- hodnota L ($L - back$), která ovlivňuje možné propojení sítě

Stejně jako u GP, i zde má každý uzel několik vstupů a jednu z několika možných funkcí, pomocí které ze vstupních hodnot vypočítá svůj výstup. Vstupy uzlu mohou být napojeny buď na některý z primárních vstupů, nebo na výstupy uzlů z předchozích L vrstev. Kandidátní řešení tak tvoří acyklický orientovaný graf. Výstup některých uzlů může být použit i vícekrát, zatímco jiné nemusí být využity vůbec. Uzly, které jsou součástí výsledného fenotypu, nazýváme *aktivní*, naproti tomu ty, které jeho součástí nejsou, a tudíž se nijak nepodílí na funkčnosti jedince a jeho fitness hodnotě, se nazývají *neaktivní*.

Zakódování

Skutečnost, že rozměry mřížky uzlů jsou již předem pevně určené a stejné pro všechny jedince, značně usnadňuje zakódování struktury těchto jedinců do jejich genotypu. Ten má podobu vektoru celočíselných hodnot. Uzly mají přiděleny indexy, které korespondují s jejich pozicí v mřížce. Do genotypu jsou pak uzly zapsány postupně za sebou, každý

má pro sebe vyhrazeno $u_{in} + 1$ genů. Zde jsou zaznamenány indexy těch uzlů (popřípadě primárních vstupů), které tvoří jeho vstupy. Za nimi následuje index funkce tohoto uzlu. Na konci genotypu je pak n_{out} genů, označujících indexy uzlů, které jsou považovány za primární výstupy. Možné zakódování genotypu lze vidět na obrázku 2.5.



Obrázek 2.5: Zakódování fenotypu do genotypu.

Fitness hodnota

Výpočet fitness je zpravidla časově nejnáročnější částí evolučního procesu. Postup je v principu stejný jako u klasických GP: Mějme množinu trénovacích dat obsahující vektor vstupů a vektor správných výstupů. Pro každý prvek trénovací množiny se pak kontroluje, na kolik se výstup kandidátního programu liší od správného výstupu. Na primární vstupy jsou přivedeny testované vstupní hodnoty a následně jsou vypočítány hodnoty jednotlivých uzlů. Uzly se vyhodnocují postupně od nulté vrstvy nahoru, dokud nejsou vyhodnoceny všechny uzly označené jako primární výstupy. Samozřejmě není nutné vypočítávat hodnotu všech uzlů. Důležité jsou pouze ty, které jsou aktivní, tedy ty, které patří do orientovaného grafu tvořícího fenotyp řešení. Proto je vhodné ještě před začátkem testování provést optimalizaci, při které se uzly grafu postupně prochází pozpátku od primárních výstupů, a označují se ty z nich, které jsou součástí fenotypu.

Evoluční proces

Algoritmus nejprve vygeneruje náhodnou počáteční populaci. Její velikost je většinou relativně malá, zpravidla to bývá pět jedinců. Zjistí se fitness všech jedinců a ten, který ji má nejvyšší, se stává rodičem. Novou generaci pak tvoří rodič a jeho náhodně zmutované varianty. Tento krok opakujeme tak dlouho, dokud nejsou splněny ukončující podmínky - například fitness dosáhla požadované úrovně, nebo se její hodnota již delší dobu nezlepšila.

Mutace

Při tvorbě nového jedince se v CGP obvykle používá pouze operátor mutace. Náhodně se přepíše hodnota jednoho či několika genů v genotypu. Tím se může změnit funkce konkrétního uzlu, zapojení uzlů za sebou, či to, které uzly jsou primárními výstupy jedince. Aktivní uzly se mohou stát neaktivními a naopak. Pokud mutace proběhne na neaktivním

uzlu, nemá žádný vliv na fenotyp jedince, přesto je důležitá. Existuje pravidlo, že v případě, kdy má v populaci nejlepší fitness více jedinců zároveň, vybere se jako rodič ten, který nebyl rodičem minulé generace. I když pak může mít nový rodič stejný fenotyp jako ten předchozí, mutace na neaktivních uzlech se přenesou na novou generaci a pomalu se v genotypu hromadí. Neaktivní uzly se tak v průběhu vývoje postupně mění a vytvářejí celé nové struktury. Pak už stačí jen drobná mutace, kdy se náhodný podgraf původně neaktivních uzlů stane součástí fenotypu a jedinec získá zcela odlišné chování. Pokud to povede ke zlepšení fitness, může se stát rodičem a tato změna fenotypu se přenese na další generace. Mutace aktivních uzlů se takto hromadit nemohou, protože ve většině případů má osamocená mutace negativní vliv na fitness jedince, a evoluční proces ji tudíž zahodí. Proto je vhodné nastavit parametry evoluce tak, aby měly v průběhu procesu kandidátní programy k dispozici dostatečné množství redundantních uzlů [7].

Křížení se v klasickém CGP téměř nepoužívá, neboť se ukázalo, že jeho použití v obvyklých úlohách pro CGP nezlepšovalo efektivitu algoritmu a mělo spíše negativní vliv. Na druhou stranu proběhly pokusy s modifikovanou variantou CGP pracující s odlišnou reprezentací genotypu, kde je použití křížení smysluplné a v některých případech dokázalo značně urychlit prohledávání stavového prostoru, jak je uvedeno v [1].

2.2 Koevoluční algoritmy

Použití evolučních algoritmů se osvědčilo u mnoha prohledávacích a optimalizačních problémů, kde značně předčily do té doby tradičně používané heuristické metody. Existují problémy, u kterých evoluční algoritmy nejsou tak efektivní, selhávají, anebo není znám způsob, jak by je bylo možné přímo použít.

- Situace, kdy hledáme optimální nastavení několika vzájemně působících komponent. Prohledávaný prostor je v takovém případě kartézským součinem veškerých možných stavů všech komponent. Jeho velikost může být obrovská, v extrémních případech i nekonečná. Proto je zapotřebí mechanismus, který by zaměřil prohledávání na relevantní oblasti.

- Úlohy, u kterých nelze jednoznačně stanovit jednoznačnou fitness jedince, protože ta závisí na měnících se okolnostech. Tak tomu může být kupříkladu u evolučního vývoje herních strategií.

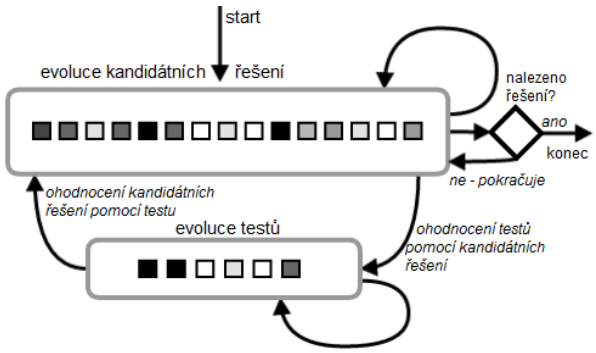
- Problémy, u kterých je výpočet fitness jedince výpočetně příliš náročný, protože trénovací sada je velice rozsáhlá. Výpočet by se mohl značně urychlit, pokud by se podařilo nalézt malou podmnožinu vstupů, pomocí které by šla fitness dostatečně přesně aproximovat.

Tyto a podobné problémy pomáhá řešit metoda zvaná *koevoluce*. Jde o přirozené rozšíření evolučních algoritmů. Opět vychází z přírodních principů, kde se více druhů vyvíjí současně, navzájem na sebe působí tím, že si vzájemně utváří prostředí, ve kterém žijí a adaptují se na toto prostředí. Příkladem budiž dravec a kořist. Dravec se postupně vyvíjí stále účinnější techniky lovu kořisti a tím nutí kořist, aby se zdokonalovala v metodách, jak dravci uniknout a naopak dravec se adaptuje na nové schopnosti kořisti. Koevoluční algoritmus může pracovat s jednou nebo i více populacemi, jejichž evoluce probíhá paralelně. Zásadní rozdíl oproti evolučním algoritmům je způsob, jakým je zde chápána fitness. Jedinci zde není přiřazena nějaká absolutní, stálá, neměnná hodnota - *objektivní fitness*. Jeho fitness se odvozuje podle toho, jak interaguje s jinými jedinci. Změna jednoho jedince tak může ovlivnit

fitness jiných. Tato *subjektivní fitness* je proto relativní a značně proměnlivá v čase podle složení populace. Použitím koevoluce je možné dosáhnout stabilnějšího průběhu evoluce a znatelně rychlejší konvergence fitness. Podle [9] a [8] dochází u koevoluce také mnohem méně často k uváznutí v lokálním extrému.

Koevoluční algoritmy lze rozdělit do několika skupin podle typu úlohy a počtu populací, se kterými pracují:

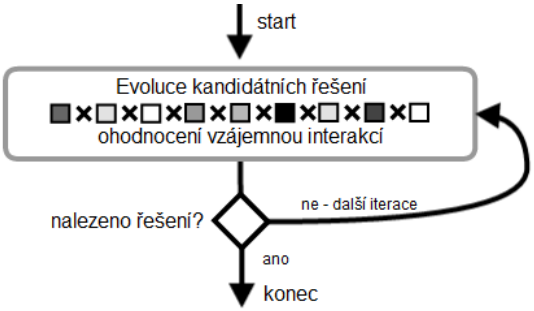
Dvě populace a úlohy založené na testu: Úlohy založené na testu jsou typem úloh, u nichž se k ohodnocení řešení používá sada trénovacích vektorů - takzvaný *test*. Kvalita kandidátního řešení se odvodí z jeho interakce s jednotlivými testy. Za kandidátní řešení se považují pouze jedinci první populace. Jedinci z druhé populace jsou testy a slouží k ohodnocení kandidátních řešení - vybírají z množiny trénovacích vektorů jen menší podmnožinu trénovacích vektorů, pomocí kterých se odhadne fitness kandidátních řešení. To je výhodné v situacích, kdy je množina trénovacích vektorů příliš velká a je zapotřebí zrychlit evoluční proces. Evoluce obou populací může probíhat odděleně, každá svým vlastním tempem. Kandidátní řešení se vyvíjejí tak, aby co nejlépe obstály při testování současnou generací testů. Jedinci z populace testů se přitom snaží vybrat sadu trénovacích vektorů tak, aby co nejlépe pomohla zvyšovat fitness kandidátních řešení (obrázek 2.6).



Obrázek 2.6: Koevoluce u úloh založených na testu.

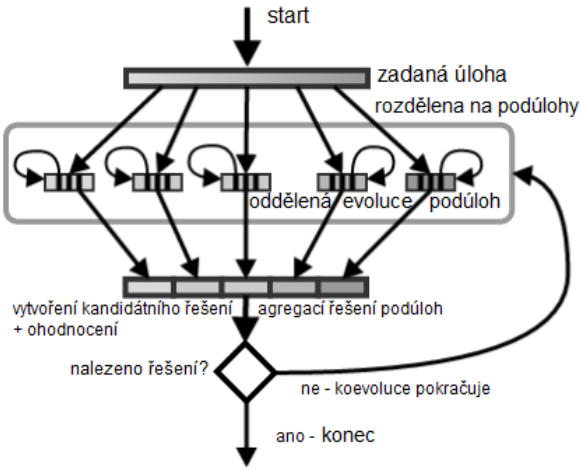
Jediná populace u úloh založených na testu: U koevolučních algoritmů pracujících jen s jedinou populací slouží jedinci dvěma účelům. Jednak jsou to kandidátní řešení zkoumané úlohy, ale zároveň jsou použiti při ohodnocení ostatních jedinců. Může se jednat například o evoluční vývoj herní umělé inteligence. Každý jedinec představuje určitou herní strategii a jeho fitness hodnota se určí podle toho, jak dobře si povede při sehrání hry proti ostatním jedincům. Případně se lze počítání fitness zcela vyhnout, pokud bude vzájemná interakce jedinců plnit úlohu turnajového výběru rodičů, jak je uvedeno v [2]. Do další generace postoupí ten, který ve hře zvítězí (obrázek 2.7).

Více populací u kompozičních problémů: Někdy je možné složitější úlohy rozdělit na jednodušší podproblémy a ty řešit zvlášť. V takové situaci se uplatní koevoluční algoritmus s větším počtem populací. Každá populace představuje jeden podproblém a jejich evoluce probíhá odděleně. Aby bylo možné provést ohodnocení, musí se jedinci z jednotlivých částí spojit dohromady do celkového kandidátního řešení. Koevoluce tak často mívá kooperativní charakter, kdy spolu oddělené druhy navzájem spolupracují na společném řešení úlohy. Nemusí tomu tak ale být vždy - jedinci spolu mohou



Obrázek 2.7: Koevoluce s jedinou populací.

napříč populacemi soupeřit o sdílené zdroje - například u evolučního návrhu budov to můžou být peníze, prostor a podobně (obrázek 2.8).



Obrázek 2.8: Koevoluce u kompozičních úloh.

Archiv

Koevoluční algoritmy často používají kromě populací ještě jiný typ kolekce jedinců, kterému se běžně říká *archiv*. Do archivu se během evolučního procesu postupně ukládají nejlepší jedinci z různých generací, takže jej můžeme považovat za jakousi formu paměti algoritmu. Tím pádem archiv na konci obsahuje celkové řešení úlohy, anebo dokonce sám může být celkovým řešením, pokud je úkolem sestavit množinu nejlépe ohodnocených jedinců. Archiv ale nemusí sloužit pouze jako jakýsi zásobník nejlepších dosud nalezených řešení. Bývá obvyklé, že obsah archivu přímo ovlivňuje samotný koevoluční proces. Může se tak dít například u ohodnocení jedinců. Příkladem může být skupina metod zvaných *Síň slávy*, kam jsou ti nejúspěšnější nalezení jedinci vkládáni do archivu a při ohodnocování dalších generací se část fitness jedinců určí podle toho, jak interagují s vybranými jedinci z archivu. Při řešení úloh, kde je použito více druhů populací, může mít každá populace svůj vlastní archiv. Většina takto založených koevolučních metod pak ve svém algoritmu obsahuje UPDATE krok, kde se posuzuje, zda některé z jedinců současné populace vložit do archivu, a kteří to budou. Stejně tak se hodnotí stávající jedinci v archivu, zda některé z nich nevyřadit. Jak je uvedeno v [5], často je možné získat algoritmus pro řešení určité úlohy tak, že správně

implementujeme tuto UPDATE metodu.

2.2.1 Predikce fitness

Pojmem *Predikce fitness* nazýváme postup, při kterém u evolučního algoritmu nepočítáme přesnou fitness jedinců, ale místo toho ji pouze přibližně odhadujeme technikami, které se samy postupně vyvíjejí v průběhu evolučního procesu. Toto je výhodné zejména v situacích, kdy je přesný výpočet fitness příliš časově náročný - ať už z důvodů obrovského rozsahu trénovací sady, komplexní simulace, anebo protože je ohodnocení potřeba provést pomocí fyzického experimentu. Existuje několik různých přístupů, jak je možné predikci fitness provádět [6]:

Strojové učení: V závislosti na podobě kandidátních řešení si lze vybrat z mnoha typů algoritmů strojového učení. Jmenujme například neuronové sítě, SVM, rozhodovací stromy, Bayessovské sítě a další. Všechny tyto metody lze natrénovat tak, aby pro vstupní vektor genotypu jedince vracely odhad jeho fitness hodnoty. Hlavní nevýhodou je fakt, že je často velice těžké odhadnout, která metoda je pro konkrétní problém ta nejvhodnější.

Podmnožina trénovacích dat: Výběr pouze malého vzorku trénovacích dat, pomocí kterých bude proveden výpočet fitness hodnoty, je častý způsob, jak značně urychlit evaluaci kandidátních řešení. Výběr těch nejvhodnějších vzorků je možné provést pomocí evolučního algoritmu - jedná se o příklad použití koevoluce.

Evolučně specifické metody: Do této skupiny patří metody *dědičnost fitness*, *imitace fitness* a *částečné ohodnocení*.

- Při *fitness dědičnosti* získávají potomci fitness svých rodičů během křížení, tedy proporčně vzhledem k tomu, kolik genů toho kterého rodiče získali.
- Metoda *Imitace fitness* rozdělí jedince do shluků podle podobnosti jejich genotypů. Ohodnoceni jsou pouze jedinci uprostřed shluků a ostatní jejich fitness převezmou.
- *Částečná evaluace* ohodnotí pouze některé jedince, zatímco ostatní svou fitness zdědí po rodičích, anebo je její hodnota odhadnuta.

2.3 Symbolická regrese

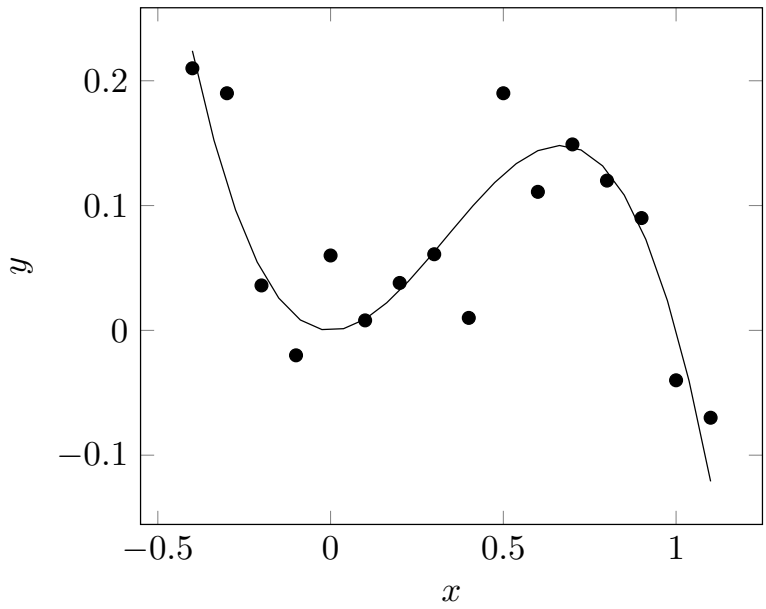
Symbolická regrese je typem úlohy, na jejíž řešení se genetické programování často s úspěchem používá. Jde o proces, kdy máme k dispozici množství naměřených dat a snažíme se nalézt takový matematický vztah, který tato data dostatečně přesně aproximuje. V matematice a fyzice se tato metoda běžně využívá, dokáže z experimentálně zjištěných údajů odvodit skryté vztahy a zákonitosti. U tradičních metod regresní analýzy - například nelineární regrese nebo metody nejmenších čtverců - musíme nejprve odhadnout, jaké jsou vstupní veličiny a jaký je vztah mezi vstupními a výstupními proměnnými. Pak tyto metody dokáži efektivně nalézt konkrétní parametry tohoto modelu. Naproti tomu symbolická regrese dokáže tyto vztahy a jejich parametry odhalit sama a zcela automaticky, bez potřeby jakýchkoli implicitních znalostí z daného vědního oboru.

Symbolická regrese nám umožňuje predikovat hodnoty pro dosud neměřené parametry, ale především jsme díky ní schopni zkoumaný jev dále matematicky analyzovat. Pro nalezenou funkci můžeme přesně určit minima a maxima, derivace v daných bodech, intervaly, ve kterých stoupá či klesá a mnoho dalšího.

Dlouho se nedařilo vyvinout metodu, která by dokázala symbolickou regresi provádět automaticky a dostatečně spolehlivě [10]. Prvním opravdu použitelným algoritmem bylo Genetické programování Johna Kozy, o němž již byla řeč již dříve. Kandidátních řešení zde představují potenciální hledaný vzorec, který popisuje zkoumaný jev. Trénovací sadu tvoří naměřená data - vstupní hodnoty (podmínky měření) a výstupní hodnoty (zjištěné údaje). Předpokládá se, že výstupní hodnoty závisí na jedné nebo více vstupních hodnotách. V tabulce 2.1 je příklad trénovací sady. Na obrázku 2.9 jsou jednotlivé trénovací vektory zobrazeny v grafu a proloženy nalezenou funkcí. Hodnota y je zde závislá na vstupu x podle vztahu $y = x^2 - x^3$.

x	-0,4	-0,3	-0,2	-0,1	0	0,1	0,2	0,3	...	0,9	1	1,1
y	0,21	0,19	0,036	-0,02	0,06	0,008	0,038	0,061	...	0,09	-0,04	-0,07

Tabulka 2.1: Příklad naměřených dat.



Obrázek 2.9: Graf bodů z tabulky 2.1 a jejich proložení funkcí $y = x^2 - x^3$.

2.3.1 Výpočet fitness hodnoty

Fitness jedinců je možné určit například jako *průměrnou odchylku* - průměr vzdáleností předpovězených hodnot $g(x_i)$ od těch skutečně naměřených y_i:

$$f(g) = \frac{\sum_{i=0}^{n} |y_i - g(x_i)|}{n} \tag{2.3}$$

Čím je tato hodnota nižší, tím je jedinec g kvalitnější. Tato metoda podává dobré výsledky, pokud se data v trénovací sadě vyznačují vyšší mírou chybovosti. Jsou upřednostňováni

jedinci, kteří dokáží vstupní data aproximovat. Metoda je ovšem značně citlivá na výskyt odlehlých hodnot. Je méně vhodná, pokud je hledán matematický vztah, který co nejpřesněji odpovídá všem vektorům z trénovací sady.

Dalším způsobem výpočtu fitness jedince je použití *skóre*, jak je uvedeno v [9]. Výpočet skóre je definován jako:

$$f(g) = \sum_{i=0}^{n} h(g(x_i)), \text{ kde} \tag{2.4}$$

$$h(g(x_i)) = \begin{cases} 0 & \text{pro } |y_i - g(x_i)| \geq \epsilon \\ 1 & \text{pro } |y_i - g(x_i)| < \epsilon \end{cases} \tag{2.5}$$

Uživatelem definovaná hodnota ϵ je maximální povolená odchylka. Fitness hodnota tedy v tomto případě udává počet trénovacích vektorů, které se od správných hodnot liší nejvýše o hodnotu ϵ. Tento způsob určování fitness je daleko méně choulostivý na výskyt odlehlých hodnot. Jedinci jsou hodnoceni jen podle počtu správně aproximovaných bodů. Průběh jejich funkcí v ostatních oblastech nemá na hodnotu fitness žádný vliv. Díky tomu není evoluční proces tak omezován a může prohledávat větší prostor možných variant řešení. Hlavní nevýhodou je fakt, že uživatel musí definovat maximální povolenou odchylku. Optimální hodnota je u každé úlohy individuální. Na druhou stranu lze pomocí velikosti odchylky specifikovat, jaká míra přesnosti je u hledané matematické formule požadována.

Metoda počítaní fitness podle skóre se ukázalo být mnohem vhodnější při řešení úloh symbolické regrese. Při jejím použití se zvýšila procentuální úspěšnost nalezení hledaného vztahu až o desítky procent.

Stejně jako u mnoha metod strojového učení, i zde si musíme dávat pozor na *přeučení* - tedy na situaci, kdy se jedinci adaptují i na drobné chyby a odchylky v naměřených datech, místo aby generalizovaly vztah pro zkoumaný jev. Klasickým způsobem řešení je metoda *cross-validace*, kdy se trénovací sada rozdělí na 2 části - na tréning modelu se použije jen jedna sada a na jeho ohodnocení druhá. U genetického programování je *přeučení* částečně spjato s bloatem.

2.3.2 Symbolická regrese a koevoluce

Koevoluce již byla v minulosti v mnoha případech úspěšně aplikována při řešení úloh symbolické regrese. Jako příklad si můžeme uvést práci [9]. Kandidátní řešení zde byla vyvíjena pomocí CGP a vývoj fitness prediktorů probíhal pomocí *genetického algoritmu*[1]. Prediktory fitness měly podobu vektoru celočíselných hodnot, kde každá hodnota byla ukazatelem na jednu položku z trénovací sady. Po řadě experimentů na pěti testovacích úlohách se ukázalo, že se s použitím koevoluce urychlil proces nalezení řešení až pětinásobně oproti klasickému CGP algoritmu.

Do oblasti symbolické regrese však spadá i mnohem více praktických úloh, než pouze odvozování matematických rovnic. Jako příklad lze uvést evoluční návrh obvodů - řadících sítí, obrazových filtrů a podobně. Vstupem evolučního algoritmu je v takovém případě trénovací sada, kde je zachyceno požadované chování výsledného obvodu v určitých situacích situacích - tedy pro různé vstupní hodnoty. Evoluční algoritmus se poté snaží nalézt takové zapojení obvodu, které co nejlépe odpovídá zadané specifikaci. Místo matematických funkcí

[1] Genetický algoritmus je optimalizační metoda patřící do skupiny evolučních algoritmů. Jako genotyp v něm představují vektory hodnot pevné délky, užívá křížení i mutaci. Pro více informací viz například [7].

jsou zde použita logická hradla, komparátory, násobičky a jiné prvky. Aby byl výsledný obvod opravdu robustní, musí být zpravidla trénovací sada velice rozsáhlá. Například při vývoji obrazového filtru musí být jedinci otestováni pomocí desítek tisíc trénovacích vektorů v každé generaci a celý proces trvá velice dlouho. Použití koevoluce, která by celý proces urychlila, se zde přímo nabízí. Vývojem nelineárních obrazových filtrů se zabývá kupříkladu práce [8]. Ukázalo se, že s použitím koevoluce bylo možné zredukovat trénovací sadu na pouhých 15 - 20% a urychlit celý proces téměř trojnásobně ve srovnání s použitím standardního CGP.

2.3.3 Algoritmus koevoluce pro symbolickou regresi

U úloh založených na testu, mezi něž symbolická regrese patří, spočívá koevoluce v souběžné evoluci dvou populací - populace kandidátních řešení a populace fitness prediktorů. Tyto populace se vzájemně ovlivňují, adaptují se jedna na druhou. Obrázek 2.10 převzatý z literatury [9] je dobrým příkladem toho, jak koevoluce u úloh založených na testu standardně pracuje.

Populace kandidátních řešení

Evoluční proces u kandidátních řešení hledá řešení zadané úlohy. Na schématu 2.10 probíhá evoluce pomocí kartézského genetického programování, ovšem mohou být použity i jiné typy evolučních algoritmů vhodné pro danou úlohu založenou na testu. K ohodnocení jedinců se nepoužívá celá trénovací sada. Pomocí prediktoru je z ní vybrán pouze malý reprezentativní vzorek. Díky tomu je ohodnocení mnohem rychlejší. K samotnému ohodnocení je možné použít jednu z metod výpočtu fitness popsaných v části 2.3.1.

Archiv trenérů

Z populace kandidátních řešení je v pravidelných intervalech vybírán nejlépe ohodnocený jedinec a je vložen do archivu. Těmto jedincům se říká trenéři a slouží k ohodnocení prediktorů během jejich evoluce. Archiv trenérů je množina, která může obsahovat nejvýše předem stanovený počet trenérů. Zároveň s vložením nového jedince je proto nejstarší jedinec vyřazen. Při vkládání se současně vypočte jeho skutečná fitness - hodnota, která bude později použita při evoluci prediktorů.

Pro optimální vývoj prediktorů je nutné, aby byli jedinci v archivu dostatečně různorodí. Z tohoto důvodu jsou do archivu přidáváni kromě kandidátních řešení i náhodní jedinci. Tím se zajistí diverzita archivu. Zároveň se jedná o opatření proti uváznutí koevolučního procesu, neboť je tak zajištěno, že se prediktory budou vyvíjet v neustále se měnících podmínkách.

Populace fitness prediktorů

Fitness prediktory jsou jedinci, kteří slouží k výběru malé reprezentativní podmnožiny trénovací sady. Mělo by se jednat o vzorek, pomocí něhož se již dá dostatečně spolehlivě odhadnout skutečná fitness kandidátních řešení. Genotyp prediktoru sestává z vektorů ukazatelů na zvolené trénovací vektory. Tato sada vybraných vektorů bude dále označována jako *predikovaná sada*. Evoluce je prováděna pomocí genetického algoritmu. Ohodnocení kvality prediktoru se provádí pomocí trenérů v archivu. U každého trenéra je stanoveno, o kolik se liší jeho skutečná fitness od fitness odhadnuté pomocí prediktoru. Fitness hodnota

prediktoru je určena jako průměr z těchto odchylek. Prediktor s nejlepší fitness je následně odeslán do archivu prediktorů.

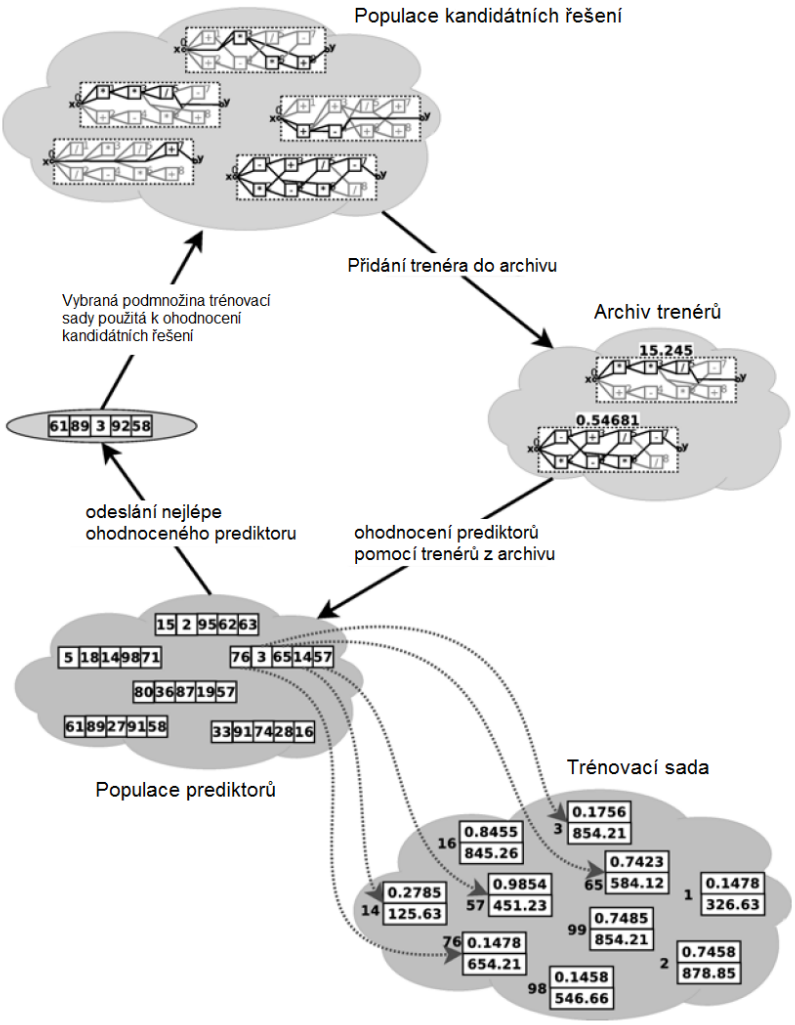
Archiv prediktorů

Tento archiv obsahuje nejlépe ohodnocený prediktor. Jím predikovaná sada je použita při ohodnocování jedinců z populace kandidátních řešení. Tento archiv obsahuje právě jednoho jedince, který je nahrazen po vložení nového prediktoru.

Průběh koevoluce

Výpočet běží ve dvou vláknech - v prvním probíhá evoluce kandidátních řešení, druhé má na starosti evoluci prediktorů. Na počátku se vygenerují náhodné populace prediktorů, kandidátních řešení a obsah archivu trenérů. Proběhne vyhodnocení jejich fitness. Zpočátku se použije nejlépe ohodnocený jedinec z populace prediktorů jako počáteční prediktor. Následně se v obou vláknech spustí evoluční proces. Vzájemná rychlost obou procesů je nastavena tak, aby se během jedné iterace evoluce prediktorů stihlo provést několik set iterací evoluce kandidátních řešení. Toto zpoždění je potřebné, neboť příliš často se měnící prediktor by měl negativní vliv na vývoj kandidátních řešení - evoluce by přestala konvergovat k určitému řešení a dostala by spíše charakter náhodného prohledávání. Mezi vlákny dochází pravidelně k výměně jedinců. Jakmile je objeveno nové kandidátní řešení s odlišnou fitness, je odesláno do archivu trenérů. Zároveň se v každém kroku evoluce prediktorů odešle nejlépe ohodnocený prediktor do archivu prediktorů.

Pravidelně se kontroluje, zda predikovaná fitness kandidátního řešení dosáhla požadované finální úrovně. Predikovaná fitness je stanovena pouze pomocí části trénovacích vektorů, proto je nutné určit i skutečnou fitness jedince pomocí celé trénovací sady. Je-li skutečná fitness na požadované hodnotě, bylo nalezeno řešení a výpočet může skončit. V opačném případě šlo o planý poplach a současný prediktor nedostačuje. Nalezené kandidátní řešení je odesláno do archivu trenérů a počká se na zaslání nového prediktoru.



Obrázek 2.10: Schéma koevolučního algoritmu (převzato z [9]).

Kapitola 3

Návrh řešení koevolučního algoritmu

Použití koevoluce prediktorů fitness tak, jak byla popsána v sekci 2.3.3, může v porovnání s CGP mnohonásobně urychlit nalezení řešení. Uvedený algoritmus má však i své slabé stránky. Ta nejzávažnější se týká velikosti predikované sady - tedy počtu datových trénovacích vektorů, které prediktor vybere z trénovací sady.

Výše popsaný model koevoluce prediktorů fitness využívá pro evoluci genetický algoritmus (GA). Platí, že genotyp prediktoru představuje lineární pole ukazatelů na vybrané trénovací vektory. Velikost tohoto pole musí být nastavena ještě před spuštěním koevolučního algoritmu a zůstává v průběhu koevoluce neměnná.

V optimálním případě má predikovaná sada dostatečnou velikost, aby bylo s její pomocí možné odhadnout fitness vyvíjených kandidátních řešení s dostatečnou přesností. Zároveň by ale měla být co nejmenší, aby se snížil počet potřebných vyhodnocení. Měl by se tak maximálně urychlit proces určení fitness kandidátních řešení. Nalezením rovnováhy mezi těmito dvěma protichůdnými požadavky se v ideálním případě zajistí nejefektivnější průběh prohledávání - a tedy i statisticky nejrychlejší nalezení řešení.

Optimální velikost není možné stanovit obecně - tato hodnota je značně individuální u každé úlohy. Například pokud je hledán vztah pro rovnici s relativně jednoduchým průběhem, může v některých případech postačit i prediktor obsahující pouhé 3 nebo 4 trénovací vektory. Opačným extrémem je například evoluce obrazových filtrů. Tam může být zapotřebí vybrat tisíce trénovacích vektorů. Odhadnout dopředu optimální velikosti predikované sady pro konkrétní danou úlohu bývá poměrně složité. U komplikovanějších úloh je zpravidla nutné ji určit experimentálně - vyzkoušet evoluci pro více možných délek a rozhodnout podle naměřených dat, která velikost je pro danou úlohu nejvýhodnější.

Další slabinou může být rychlost evoluce prediktorů v případech, kdy je trénovací sada velice rozsáhlá. Příkladem budiž již výše zmíněná evoluce obrazových filtrů. Každou generaci tvoří zpravidla desítky jedinců, z nichž každý je reprezentován jako pole tisíců ukazatelů. Jak uvádí [7] v části o problému škálovatelnosti, využívání takto rozsáhlých genotypů může být značně neefektivní.

Tato diplomová práce se zabývá novou metodou evoluce prediktorů, která má za cíl výše zmíněné nedostatky odstranit. Užití této metody umožňuje, aby se velikost predikované sady v průběhu koevoluce dynamicky měnila. Může se tak přizpůsobit konkrétní úloze i aktuálnímu stavu koevolučního procesu.

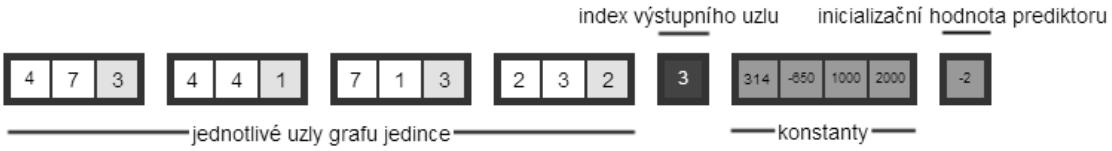
Nejrozsáhlejší částí této kaptoly je sekce 3.6, kde je podrobně představen nový algorit-

mus pro predikci fitness. Dále je popsán návrh jednotlivých částí koevolučního algoritmu - genotyp jedince, způsob práce s konstantami, genetický operátor mutace, implementace archivů a paralelizace algoritmu.

3.1 Genotyp jedince

Evoluce kandidátních řešení, stejně jako evoluce prediktorů fitness, probíhá pomocí CGP. Díky tomu je možné v obou případech použít stejnou reprezentaci genotypu. Genotyp jedince kartézského genetického programování je vektor celočíselných hodnot kódující acyklický orientovaný graf. Jako základ byla použita podoba genotypu popsaná v části 2.1.1. Navíc byla přidána série genů kódujících konstanty. U prediktorů je pak na samém konci umístěn gen pro inicializační hodnotu predikce. Prakticky se jedná o další konstantu - její čtení, ukládání a mutace probíhá obdobným způsobem . Rozdíl je v jejím využití - nemůže se stát součástí kódovaného grafu a je použita pouze při výpočtu predikované množiny.

Struktura genotypu je znázorněna na obrázku 3.1



Obrázek 3.1: Struktura genotypu.

3.2 Typy konstant

Součástí fenotypu jedince se mohou stát prvky s konstantní hodnotou, nezávislou na primárním vstupu. Podle jejich původu a vlastností je lze rozdělit do tří skupin.

- Konstantní výstupy funkcí - Některé funkce mohou při určité kombinaci vstupních hodnot generovat konstantní výstup. U funkcí implementovaných v této aplikaci lze například uvést $\frac{x}{x} = 1$, $x - x = 0$, nebo také $x \bmod x = 0$. Tyto konstanty vznikají samovolně v průběhu evoluce. Pokud je lze nahradit například implicitně definovanou konstantou, pak je možné jejich výskyt zčásti omezit technikami pro redukci *bloatu* zmíněnými v části 2.1.

- Trvalé konstanty - Před zahájením koevoluce může uživatel definovat tabulku hodnot, které mohou být použity ve fenotypu jedince. Při mutaci vstupu uzlu se náhodně určí, zda bude uzel napojen na výstup některého z předchozích uzlů, na jeden z primárních vstupů, anebo právě na jednu z konstant.

- Mutovatelné konstanty - Zde je označení „konstanta“ snad poněkud zavádějící. Podobně jako v předchozím případě se jedná opět o sadu uživatelem definovaných hodnot. Tyto hodnoty jsou však začleněny do genotypu jedince a mohou v průběhu evoluce měnit svou hodnotu. Určité omezení mutovatelných konstant vyplývá z faktu, že genotyp jedince je vektor celých čísel. Konstanta tak nemůže nabývat hodnot v celém

rozsahu čísel v plovoucí čárce. Je zaokrouhlena a vyjadřuje desetinné číslo se stanoveným počtem desetinných míst. Převod mezi hodnotou konstanty c a hodnotou jejího genu $g(c)$ lze vyjádřit vztahem

$$c = \frac{g(c)}{p} \text{ , kde} \tag{3.1}$$

$$p = 10^x \tag{3.2}$$

x je nastavený limit počtu desetinných míst. Tento typ konstant má značné uplatnění při evoluci prediktorů. Naopak při použití konstant, jako je například hodnota π, je vhodnější označit je jako neměnné.

3.3 Mutace genů

U kartézského genetického programování vzniká nová generace jedinců pomocí mutace genotypu rodiče. Při mutaci je z uživatelem definovaného rozsahu náhodně zvoleno číslo, udávající počet mutovaných genů. Poté jsou opět náhodně zvoleny indexy genů, jejichž hodnota bude pozměněna. Geny v genotypu se dělí do několika typů podle své funkce a u každého typu se liší způsob mutace.

Geny pro vstupy uzlů: Nejprve je potřeba stanovit počet uzlů, na které se může daný uzel připojit. Toto číslo závisí na hodnotě L-back a také na umístění daného uzlu v rámci celé mřížky. Dále se přičte 1, popřípadě 2, pokud byly definovány konstanty. Tím se získá rozsah, z něhož je náhodně zvolena hodnota. Pokud hodnota odpovídá některému z dostupných uzlů, pak je jako vstup vybrán tento uzel. V opačném případě je jako vstup náhodně vybrána jedna z konstant nebo primárních vstupů.

Geny funkce uzlu: U genů, které kódují, jakou funkci uzel vykonává, je genu přiřazen nový náhodně vybraný index funkce.

Primární výstupy: Jedná se o geny nesoucí informaci, který uzel je primárním výstupem jedince. Při mutaci je jim přiřazen index náhodného uzlu.

Konstanty: V těchto genech jsou uloženy mutovatelné konstanty. Evoluce těchto hodnot je inspirována technikou zvanou *evoluční strategie*. Při mutaci je vygenerováno náhodné číslo v normálním rozdělení se středem v hodnotě jedna a směrodatnou odchylkou 0,05. Stávající hodnota konstanty je pak tímto číslem vynásobena.

3.4 Označení aktivních uzlů

Při počítání odezvy jedince v CGP se postupně vyhodnocují výstupy jednotlivých uzlů v mřížce, dokud není získána hodnota primárních výstupů. Mnoho uzlů je však neaktivních - nejsou součástí fenotypu a bylo by zbytečné ztrácet čas jejich vyhodnocováním. Z toho důvodu se před prvním vyčíslením trénovacího vektoru provádí označení aktivních uzlů jedince. Příslušný algoritmus pracuje se zásobníkem a polem pro počítání odkazů na jednotlivé uzly. Na počátku je počítadlo odkazů nulové a do zásobníku jsou vloženy indexy výstupních uzlů. Následně se v iteracích prochází grafem uzlů od výstupů ke vstupům. V každém kroku je ze zásobníku vyňat index jednoho uzlu. Počet odkazů na tento uzel se inkrementuje o 1 a do zásobníku jsou vloženy všechny uzly, na které je napojen. Napojení

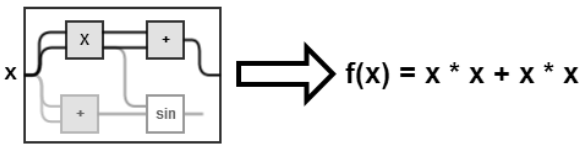
na konstanty či primární vstupy se počítá v oddělené proměnné. Celý proces se opakuje tak dlouho, dokud není zásobník prázdný. Na závěr se prochází pole s počtem odkazů. Každý uzel, který má nenulovou hodnotu je zařazen do vektoru aktivních uzlů. Součtem odkazů navíc získáme délku matematického vztahu, se kterou se dále pracuje při určování fitness hodnoty jedince.

3.5 Určení fitness kandidátních řešení

Při posuzování kvality kandidátního řešení je nejdůležitějším faktorem to, jak přesně jeho výstup, jenž je odezvou na zadané vstupy, odpovídá referenčním výstupům. Pro vyhodnocení přesnosti byla použita metoda počítání skóre, popsaná v sekci 2.3.1. Bylo experimentováno i s možností počítat fitness jako průměrnou odchylku hodnot. Pro triviální zadání dokázala tato varianta nalézt řešení znatelně rychleji, nicméně při hledání složitějších funkcí byla úspěšnost minimální. Dále je vhodné při výběru nejkvalitnějšího jedince brát v potaz i jeho velikost. Zde se nabízejí dva možné přístupy - můžeme jedince posuzovat podle počtu jeho aktivních uzlů, nebo podle délky vzorce, který jeho fenotyp představuje. Příklad použití těchto metrik lze vidět na obrázku 3.2. Obě varianty slouží pro omezení *bloatu*. Pokud bude evoluce upřednostňovat jedince s menším počtem uzlů, evoluční proces bude probíhat o něco rychleji. Ve druhém případě budou zase výsledkem kratší a lépe čitelné matematické vztahy. V popisovaném algoritmu byla vybrána varianta hodnotící délku vzorce. Spojením těchto dvou kritérií získáváme následující vztah pro kvalitu q:

$$\forall j, k \in K : q(j) > q(k) \Leftrightarrow \Big(f(j) > f(k) \vee \big(f(j) = f(k) \wedge d(j) < d(k)\big)\Big) \tag{3.3}$$

kde K je množina kandidátních řešení, $f(x)$ je fitness skóre jedince x a $d(x)$ je délka vzorce jedince x.



Obrázek 3.2: Jedinec se 2 aktivními uzly a délkou vzorce 7.

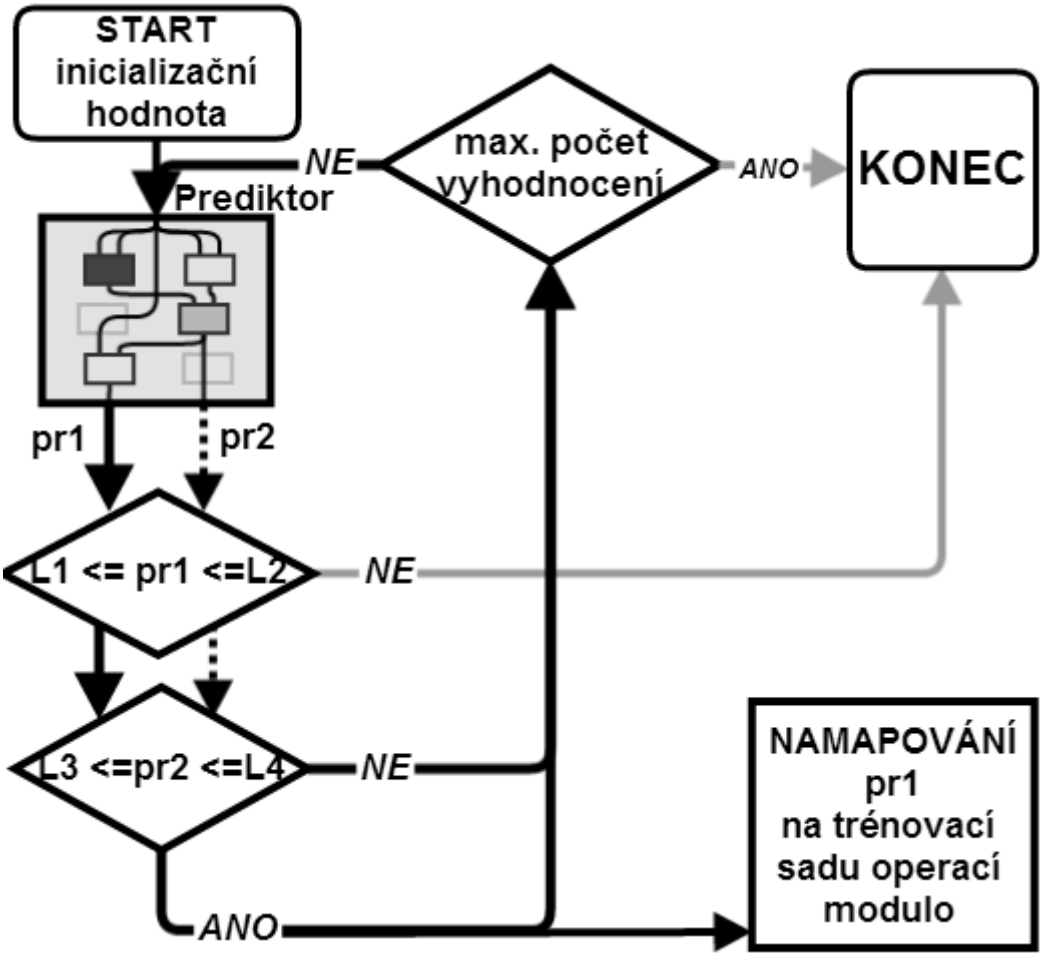
3.6 Predikce Fitness

V této části je představen návrh nové metody evoluce prediktorů, která má za cíl odstranit dosavadní nedostatky predikce fitness. Hlavním rozdílem oproti stávající metodě je fakt, že prediktory jsou vyvíjeny s pomocí kartézského genetického programování (CGP). Prediktor není vektorem ukazatelů do trénovací sady, ale matematický vztah, pomocí něhož se určí, které trénovací vektory vybrat.

3.6.1 Princip predikce

Prediktor fitness je jedinec kartézského genetického programování, jehož fenotyp představuje matematický vztah. Má jeden vstup x_i a dva výstupy - hlavní výstup $pr_1(x_i)$ a kontrolní

výstup $pr_2(x_i)$. Na počátku predikce je stanovena maximální velikost predikované sady a inicializační hodnota x_0. Tato hodnota se přivede na vstup jedince a vyhodnotí se odpovídající výstupy. První výstupní hodnota se pomocí funkce *modulo* namapuje na trénovací sadu - udává index trénovacího vektoru, který byl vybrán do predikované sady. Současně je tato hodnota v další iteraci přivedena opět na vstup jedince a je vybrán další vektor. Proces se opakuje tolikrát, na kolik je nastaven maximální počet predikovaných vektorů. Průběh predikce je znázorněn na obrázku 3.3.



Obrázek 3.3: Průběh predikce.

V tomto obrázku odpovídá *pr1* a *pr2* výstupním hodnotám prediktoru $pr_1(x_i)$ a $pr_2(x_i)$. Konstanty *L1* a *L2* jsou maximální a minimální povolenou hodnotou výstupu $pr_1(x_i)$. Obdobně *L3* a *L3* jsou maximální a minimální povolenou hodnotou $pr_2(x_i)$. Účel těchto omezení bude vysvětlen v sekci 3.6.3.

3.6.2 Porovnání s variantou využívající genetický algoritmus

Oproti genetickému algoritmu má tato nová metoda určité výhody. Dále v textu bude nová metoda predikce nazývána*CGP predikce*. Původní metoda, popsaná v části 2.3.3, bude označována jako *GA predikce*.

konstantní velikost genotypu: Genotyp jedince kóduje mřížku konstantní velikosti, jež

se zpravidla skládá pouze z několika málo desítek uzlů. Přesto může predikovat sadu tisíců trénovacích vektorů. Rychlost manipulace s genotypem je tak poměrně vysoká a konstantní pro různé vstupní úlohy.

Proměnlivá velikost predikované sady: Počet trénovacích vektorů, které prediktor vybere, je nezávislý na velikosti jeho genotypu a může se dynamicky měnit.

Je ale potřeba zmínit i určitá úskalí tohoto přístupu. Při použití *CGP predikce* je predikovaná sada výsledkem vyhodnocení matematického vztahu. Zřejmě proto může platit, že ve výběru prvků bude možné vystopovat určitou pravidelnost. Mohou proto existovat kombinace trénovacích vektorů, pro jejichž výběr může být velice obtížné (ne-li nemožné) nalézt odpovídající vztah. Tato vlastnost však nemusí být až tak zásadní problém. Dá se předpokládat, že hodnoty v trénovací sadě budou obsahovat jisté pravidelnosti - ostatně cílem symbolické regrese je nalézt výraz, jenž bude tyto pravidelnosti vystihovat. Je proto možné, že ideální predikovaná sada bude výrazné rysy trénovací sady do určité míry zrcadlit a bude se snažit přizpůsobit se jim.

Dlužno podotknout, že *GA predikce* tímto problémem netrpí. Jakákoliv kombinace trénovacích vektorů může být v genotypu zakódována stejně snadno - samozřejmě pokud má stanovenou velikost.

Návrh *CGP predikce* se bude soustředit na to, aby se vytěžilo co nejvíce z předností této metody a současně se omezily její nedostatky. To znamená, že by se velikost predikované sady měla být schopná adekvátně přizpůsobit zkoumané úloze. Zároveň by mělo být možné predikovat co nejrozmanitější kombinace trénovacích vektorů.

Seřazení trénovacích vektorů

Prediktor vybírá pouze indexy trénovacích vektorů. Pracuje tedy jen s jejich momentální pozicí v trénovací sadě, nikoliv s hodnotami, které obsahují. Body by v trénovací sadě proto měly být seřazeny tak, aby případné pravidelnosti zůstaly pokud možno zachovány. U sady s jedinou nezávislou proměnnou to znamená seřadit vektory podle hodnoty této proměnné. Složitější je situace u sad s více nezávislými proměnnými - není zcela jednoznačné, podle jakých kritérií řazení provádět. Problematické mohou být také také případy, kdy jsou mezi hodnotami nezávislých proměnných nerovnoměrné rozestupy. Neznamená to, že by se v těchto případech *CGP predikce* nedala použít, pouze se tím dosti ztíží případná možnost přizpůsobit se průběhu hledané funkce.

3.6.3 Velikost predikované sady

Mapování výstupu prediktoru na trénovací množinu proběhne tolikrát, na kolik je nastavena maximální velikost predikované sady. Bylo použito několik mechanismů, pomocí kterých může prediktor vybrat menší predikovanou sadu, než je toto maximum.

Vícenásobný výběr stejného trénovacího vektoru: Opakovaně vybraný vektor může být v sadě přítomen pouze jednou. Každý další pokus o jeho přidání tak v důsledku pouze sníží velikost výsledné sady o 1. Aby byl prediktor schopen s touto metodou lépe pracovat, může ve svém fenotypu využít funkce $a \bmod b$, $\max(a, b)$ a $\min(a, b)$. Zároveň by tyto funkce měly prediktoru umožnit i výběr nerovnoměrně rozmístěných trénovacích vektorů.

Povolený rozsah výstupů prediktoru: U obou výstupů prediktoru se kontroluje, zda jejich hodnoty nepřesáhly povolené limity. Pokud se tak stane u hodnoty *pr1*, je

predikce předčasně ukončena. Tento mechanismus má však jen omezené využití, neboť se tak stane pouze u rychle rostoucích (či rychle klesajících) funkcí. Navíc tento způsob změny velikosti predikované sady je svázán se změnou rovnice odpovědné za výběr prvků. Z toho důvodu byl přidán i druhý výstup prediktoru. Tento výstup se namapuje na trénovací sadu. Pouze se provádí kontrola, zda tento výstup leží v povoleném rozsahu. Díky tomu není tolik provázán výběr konkrétních prvků a velikost výsledné sady a je možné prozkoumat více kombinací těchto dvou faktorů. Jedná se však pouze o kompromisní řešení. V ideálním případě by měl být zcela oddělen vztah pro výběr trénovacích vektorů od vztahu a pro určení velikosti výsledné sady. V takovém případě by každého jedince tvořila dvojice genotypů. Výpočetní náročnost predikce by však u takového řešení byla několikanásobná.

Evoluce konstant: Jak bude podrobněji uvedeno dále, prediktor má k dispozici několik konstant, které se mohou vyvíjet v průběhu evoluce. To umožňuje upravovat funkci předchozích dvou mechanismů s mnohem větší přesností, než kdyby byly hodnoty konstant po celou dobu evoluce neměnné.

Omezení velikosti predikované sady

Ojediněle se stává, že je jako nejlepší prediktor vybrán jedinec, jenž predikuje pouze jediný vektor, nebo rovnou celou trénovací sadu. Pravděpodobnou příčinou může být specifická kombinace jedinců v archivu a složení aktuální generace prediktorů. Oba extrémy mají negativní vliv. Při predikci celé sady se evoluční proces výrazně zpomalí a prohledávací proces ztratí výhody koevoluce. Jednoprvková predikce má zase velmi negativní důsledky pro evoluci kandidátních řešení. V tom okamžiku totiž bude mít maximální fitness jakákoliv funkce, která náhodou prochází jediným konkrétním bodem. Tak přijdeme o veškeré dosavadní výsledky koevolučního procesu. Z tohoto důvodu se nelze spoléhat na to, že takto nekvalitní prediktory nebudou vybrány. Je nutné ohraničit maximální a minimální velikost predikce. Maximum bylo určeno na 25 % velikosti trénovací sady. Číslo přibližně odpovídá zjištění z dřívějších prací o optimálním nastavení pro koevoluční vývoj obrazových filtrů, které jsou v tomto ohledu velice náročné. Minimální velikost pak byla stanovena na hodnotu 5. Je možné, že u některých jednodušších zadání stačí i méně – 3 nebo 4 trénovací vektory. Nicméně je lepší mít určitou rezervu pro hledání komplikovanějších vztahů, byť za cenu mírného zpomalení u triviálních zadání.

3.6.4 Inicializační hodnota predikce

Na začátku predikce je potřeba vybrat číslo, které poslouží jako počáteční vstup prediktoru. Na této hodnotě velmi závisí výsledek predikce, neboť se od ní odvíjí výstupní hodnoty v následných iteracích. Otázkou je, jak takové číslo nejlépe zvolit.

Prvním způsobem je zvolit u každého jedince inicializační hodnotu náhodně ze zadaného intervalu čísel. Nebo je možné nastavit ji napevno předem, jako konstantu. Je možné, že se takto omezí různorodost predikcí - evoluce prediktorů přijde o možnost měnit predikovanou sadu pomocí změny inicializační hodnoty. Při sérii testů ukázalo, že koevoluce má většinou lepší úspěšnost s rozumně zvolenou predikční hodnotou než při jejím náhodném výběru. Důvodem může být to, že jsou v mnoha případech generovány hodnoty, které nemusí být až tak vhodné pro inicializaci predikce. Nebo je také možné, že je zvolená hodnota jednoduše lépe vyhovuje zvoleným testovacím úlohám. Tomu jsem se snažil vyhnout zahrnutím většího množství různorodých trénovacích sad, ale zcela vyloučit to nelze.

Výběr nejvýhodnější inicializační hodnoty

Ideální inicializační hodnotu lze jen těžko určit bez rozsáhlých experimentů. Možná ani taková hodnota neexistuje a pro každou úlohu je odlišná. Základním pravidlem ovšem je, že by měla podporovat rozmanitost predikcí. To mimo jiné znamená, že by se pro danou inicializační hodnotu mělo co nejméně často stávat, že budou genotypy dvou rozdílných prediktorů zobrazeny na stejnou predikovanou množinu. Tato vlastnost se odvíjí od množiny funkcí, která je pro uzly prediktoru použita.

Příkladem nevhodné inicializační hodnoty je číslo 0. Pro mnoho prediktorů, jako je například $f(x) = 5 \cdot x \cdot x$ nebo $f(x) = x + x \cdot x$ bude výsledkem vyčíslení $f(0) = 0$ Tato nula se opět přivede na vstup a výsledek je opět nulový. Výsledkem predikce tak bude pouze jednoprvková množina. Uvedené prediktory jsou přitom zcela validní při použití odlišné inicializační hodnoty. Odlišný výsledek budou dávat pouze ty fenotypy, kde je použito přičítání či odčítání nenulové konstanty, například $f(x) = x - 1$ či $f(x) = x \cdot x + \cos(x)$.

Podobná situace je u inicializační hodnoty 1, kde jsou vyřazeny fenotypy bez sčítání, odčítání, či násobení / dělení konstantou. Nemusí být špatným nápadem použít pro inicializační hodnotu záporné číslo. Funkce pro absolutní hodnotu se tak stane lépe využitelná.

Z těchto dvou přístupů bylo zvoleno kompromisní řešení. Ze začátku je použita konstanta $-0,002$, což je jedna z hodnot, která se během experimentů osvědčila. Inicializační hodnota je však součástí genotypu jedince, a může se tak během evoluce vyvíjet. Takovéto řešení by mělo ideálně skloubit přednosti obou přístupů.

3.6.5 Konstanty prediktoru

Každému prediktoru je na počátku evoluce přidělena sada konstantních hodnot. Jedinec má možnost využít konstanty jako součást matematického vztahu, který kóduje. Pro tento účel byla zvolena série čísel 1, 2, 4, 8 a dále pak hodnoty s, $\frac{s}{2}$ a $\frac{s}{4}$, kde s je velikost trénovací sady. Záměrem pro použití tří hodnot odvozených od velikosti trénovací sady bylo usnadnit prediktorům možnost zaměřit se na určitou oblast trénovací sady. Konstanty jsou součástí genotypu jedince a mohou se během evoluce postupně vyvíjet, jak je popsáno v sekci 3.2. Tak je postupně možné získat vhodnější hodnoty, což je velmi důležité. Úpravou hodnoty konstanty je možné dosáhnout mnohem menších změn, než se zpravidla děje při změně zapojení uzlů.

3.6.6 Určení fitness hodnoty prediktorů

Způsob ohodnocování kandidátních prediktorů je u *CGP predikce* zcela zásadní. Použitý mechanismus rozhoduje o tom, zda se velikost predikované sady dokáže přizpůsobit přizpůsobit zadané úloze.

U prediktorů je při určování fitness potřeba zohlednit dva faktory - velikost predikované sady a přesnost predikce. S rostoucí délkou predikované sady klesá rychlost evoluce kandidátních řešení. Zároveň musí být predikovaná hodnota dostatečně přesná. Fitness funkce musí být navržena tak, aby upřednostňovala optimální rovnováhu mezi těmito dvěma protichůdnými požadavky. Navíc je potřeba provést kontrolu, zda prediktor splňuje podmínku minimální velikosti predikované sady. Tento požadavek při ohodnocování předem vyřadí významný počet jedinců.

Hodnota udávající přesnost predikce se určí pomocí ohodnocení trenérů umístěných v archivu, jak bylo popsáno v části 2.3.3. Posuzuje se rozdíl mezi skutečnou fitness trenéra a

jejím odhadem pomocí daného prediktoru. Analogicky ke způsobům výpočtu fitness u kandidátních řešení popsaných v sekci 2.3.1 i zde je možné zvolit ze dvou odlišných přístupů - výpočet průměrné odchylky a počítání skóre. Obě metody mají obdobné klady a zápory. Při použití skóre je potřeba určit maximální tolerovanou odchylku fitness. Prediktor dostane bod za každého trenéra, jehož fitness se mu podaří odhadnout přesněji, než je tato hranice. Velkým pozitivem této metody je fakt, že podobně kvalitním jedincům je přiděleno stejné skóre. Tím se budou jedinci v archivu prediktorů častěji střídat, což by mělo mít pozitivní vliv na evoluci kandidátních řešení. Nevýhodou je opět nutnost předem definovat optimální hodnotu tolerance. Počítání průměrné odchylky fitness má tu výhodu, že dovede ohodnotit i prediktory, jejichž průměrná odchylka odhadu by u metody počítání skóre ležela hluboko pod (či vysoko nad) nastaveným prahem tolerance. Mohou ovšem nastat problémy, pokud je například do archivu vložen jedinec s extrémně špatnou fitness.

Při počítání fitness u prediktorů je užitečnější používat takzvanou *relativní odchylku fitness*. To znamená, že nepracujeme s absolutní hodnotou rozdílu fitness, ale spíše hodnotou vyjadřující poměr skutečné a predikované fitness dané pro prediktor p a trenéra t vztahem:

$$prec(p,t) = \frac{|f_{pred}(t) - f_{true}(t)|}{f_{true}(t) + n} \tag{3.4}$$

kde $f_{pred}(p,t)$ je fitness trenéra t predikovaná prediktorem p a $f_{true}(t)$ je skutečná fitness trenéra t . Konstanta n má za úkol zmírnit prudký nárůst hodnoty relativní odchylky u trenérů s fitness blížící se nule. Také zamezí dělení nulou v situaci, kdy by bylo do archivu vloženo jako trenér finální řešení úlohy[1].

Celková přesnost predikce prediktoru p se určí jako

$$prec(p) = frac\sum_{n=0}^{c-1} f_{pred}(p,t_n)c \tag{3.5}$$

kde c je počet trenérů v archivu. Rozdíl mezi způsoby určení velikosti odchylky ilustruje následující příklad:

- Trenér A má skutečnou fitness 2, ale je mu predikována fitness 3. Absolutní odchylka je 1, ale relativní odchylka činí 50%.
- Trenér B má skutečnou fitness 1001 a je mu predikována fitness 1004. Absolutní odchylka je v tomto případě 3, ale relativní rozdíl je necelých $0,3\%$.

Bylo testováno více metod výběru nejlepšího prediktoru. Na základě velikosti predikované sady a přesnosti predikce lze snadno vybrat podmnožinu paretovsky optimálních jedinců. Nicméně u kartézského genetického programování se může stát rodičem pouze jediný z nich. Proto by měl být vybrán jedinec, který představuje optimální rovnováhu mezi rychlostí a přesností predikce.

Určování přesnosti predikce pomocí skóre se při experimentech příliš neosvědčilo. Ideální hodnota tolerance byla u každé úlohy jiná a navíc se ukázalo, že tato metrika je poměrně nepřesná. Během evolučního procesu měla naprostá většina prediktorů skóre téměř nulové, nebo naopak maximální. U prediktorů je požadováno, aby byl během krátkého počtu iterací nalezen přibližně co nejvýchodnější jedinec. K tomuto účelu se více hodí určení přesnosti predikce pomocí relativní odchylky fitness.

Nakonec je třeba určit matematický vztah, který určí fitness hodnotu prediktoru v závislosti na obou zmíněných kritériích. Základní podmínky jsou následující:

[1] Hodnota n je 0.002.

- Se vzrůstající velikostí predikované sady by se celková fitness měla zhoršovat logaritmicky. To vyjadřuje skutečnost, že například u sady velikosti 10 probíhá ohodnocení kandidátních řešení dvakrát delší dobu oproti sadě o velikosti 5. Při porovnání sad s velikostmi 100 a 105 je už zpomalení pouze 5%.
- Se vzrůstající odchylkou predikované a skutečné fitness by se celková fitness jedince měla zhoršovat exponenciálně. Tedy například predikce s vysokou přesností do 5% mohou být považovány za přibližně stejně kvalitní a evoluční algoritmus se může více soustředit na optimalizaci velikosti. S klesající přesností se však čím dál tím strměji zhoršuje použitelnost dané predikce a evoluční proces by se proto měl zaměřit na její zlepšení i za cenu nárůstu velikosti.

Pro určení fitness prediktoru byl vybrán následující vztah:

$$f(p) = \big(a \cdot prec(p)\big)^4 + b \cdot size(p)\big(1 + a \cdot prec(p)^2\big) \text{ , kde} \qquad (3.6)$$
$$a = 17$$
$$b = 0.04$$

$prec(p)$ je průměrná relativní odchylka fitness u prediktoru p a $size(p)$ je velikost predikované sady prediktoru p. K získání tohoto vztahu byl použit koevoluční algoritmus v aplikaci `Eureqa`[2]. Trénovací sada sestávala ze dvou nezávislých proměnných - relativní odchylky fitness a velikosti predikované sady. Závislou proměnnou byla požadovaná hodnota fitness nastavená tak, aby graf výsledné funkce svým tvarem přibližně odpovídal výše zmíněným požadavkům. Z řady výsledných vztahů byl vybrán tento vzorec pro svou relativní jednoduchost a výhodný tvar funkce. Obě použité konstanty a a b ovlivňují míru preference mezi přesností a rychlostí predikce. Byly prováděny experimenty pro různé hodnoty konstanty a v intervalu $\langle 3, 20\rangle$. Čím je tato hodnota vyšší, tím více je upřednostňována přesnost predikce na úkor rychlosti. U nižších hodnot a se tak častěji objevovaly velice krátké, ale dosti nepřesné predikované sady. To může být výhodné zvláště u jednodušších úloh. Nepřesný prediktor umožní evoluci kandidátních řešení uniknout z případného lokálního optima a prozkoumat větší část prostoru potenciálních řešení. U složitějších úloh má však příliš častý výskyt krátkých predikovaných sad značně negativní vliv na úspěšnost prohledávání. Podle průběžných výpisů fitness a velikosti predikované sady se zdá, že nepřesné predikce mají tendenci objevovat se o něco častěji vzápětí po prudkém zlepšení fitness, než ve chvílích, kdy hodnota fitness stagnuje. Jakýkoliv dosažený pokrok bývá v takovém případě ztracen.

Tuto negativní vlastnost se podařilo částečně omezit nastavením vysoké hodnoty a - nejlépe se osvědčila hodnota 17. Kladný vliv také mělo prodloužení intervalu odesílání nejlépe hodnoceného prediktoru. Ve výsledku se významně zvýšila úspěšnost při řešení komplikovanějších úloh. Zároveň se ale prodloužila doba nalezení výsledku u triviálních zadání.

3.7 Archiv trenérů a prediktorů

Algoritmus využívá dvojici archivů pro ukládání nejlépe ohodnocených kandidátních řešení a prediktorů. Jejich funkce do značné míry odpovídá archivům popsaným v části 2.3.3. Při vkládání nového kandidátního řešení do archivu trenérů se počítá jeho skutečná fitness. Zde nemá příliš smysl pracovat s fitness v podobě skóre. Při vkládání náhodných jedinců má

[2] Jedná se o komerční aplikaci, která využívá koevoluční algoritmus uvedený v [6].

naprostá většina z nich skóre nula a jsou tedy pro evoluci prediktorů nepoužitelní. Ať už by prediktor vybral jakoukoliv podmnožinu trénovací sady, vždy by odhadl jejich fitness zcela přesně - nula. Proto je použita fitness v podobě střední odchylky.

Dále bylo upraveno vkládání náhodných jedinců do archivu. Náhodní jedinci jsou do archivu pravidelně vkládáni, aby se zajistila jeho rozmanitost. Nicméně bylo zjištěno, že většina náhodně vygenerovaných jedinců kódovala dosti triviální lineární funkce a jejich fitness byla velmi podobná. Proto byla upravena jejich L-back hodnota nastavena na 2. Takovéto omezení vedlo častějšímu generování složitějších jedinců. Zvýšila se rozmanitost fenotypů a fitness hodnot. Také to ale muselo mít negativní dopad na rychlost evoluce prediktorů, neboť ohodnocení jedince s větším počtem aktivních uzlů trvá déle.

U archivu prediktorů je hlavní odlišnost v tom, že do něj není nový prediktor zasílán s každou iterací evoluce prediktorů. Důvodem je rozdílné chování genetického algoritmu a kartézského genetického programování. U GA je vysoká pravděpodobnost, že v každé iterací bude jako rodič vybrán nový odlišný jedinec. Pro CGP je naproti tomu typické, že nejlepší jedinec může mít stejný fenotyp po dlouhou řadu iterací, dokud náhodnou mutací nevznikne lepší. Jednotlivé iterace u CGP navíc bývají zpravidla rychlejší než u GA. Proto je efektivnější odesílat prediktor v delších intervalech, případně alespoň kontrolovat, jestli se nový nejlepší prediktor liší od předešlého.

3.8 Paralelizace

Koevoluce může být časově velice náročná. Je proto nutné snažit se optimalizovat časově nejnáročnější části algoritmu. Zároveň by aplikace měla být schopna využít co nejvíce výpočetního výkonu, který má právě k dispozici. Toho lze dosáhnout rozložením výpočtu na více procesorových jader. Diagram 3.4 znázorňuje paralelní běh algoritmu. Koevoluční algoritmus pro úlohy založené na testu pracuje se dvěma evolučními procesy, které běží paralelně, pouze si v pravidelných intervalech navzájem zasílají nejlépe vyvinuté jedince. Je proto snadné rozdělit tyto dva do značné míry nezávislé celky do oddělených vláken. Veškerou komunikaci iniciuje vlákno s evolucí kandidátních řešení. V pravidelných intervalech odesílá nové trenéry do archivu evoluce prediktorů. Dále po čase zašle požadavek na zaslání nového prediktoru, který mu je obratem doručen. Vlákno kandidátních řešení kontroluje, zda nebyly splněny podmínky pro ukončení běhu algoritmu (nalezeno řešení, vypršel maximální počet iterací a podobně). Pokud se tak stane, nastaví sdílenou proměnnou na příslušnou hodnotu a skončí. Vlákno evoluce prediktoru tuto proměnnou pravidelně kontroluje a také ukončí svůj běh. Bude-li k dispozici 3 a více vláken, naskýtá se možnost paralelizovat další část algoritmu - výpočet fitness hodnoty jedinců. Výpočet fitness je paralelizován pouze u evoluce kandidátních řešení. Je tomu tak ze dvou důvodů:

- Nalezení řešení úlohy je hlavní prioritou algoritmu. Možnost prozkoumat mnohem více variant kandidátních řešení.
- Lze předpokládat menší časové ztráty při čekání vláken na dokončení běhu ostatních. Doba vyhodnocení se může lišit v závislosti na počtu aktivních uzlů ve fenotypu jedince, ovšem rozdíly nebudou tak zásadní, jako u evoluce prediktorů. Pro maximalizaci efektivity je velikost populace kandidátních řešení navýšena na nejbližší násobek počtu dostupných vláken. Je tak možné výpočet rovnoměrně rozdělit mezi jednotlivá vlákna. Tím se sníží průměrná doba, kdy některá vlákna nečinné čekají na dokončení běhu ostatních vláken. Dojde tak k navýšení výkonu algoritmu.

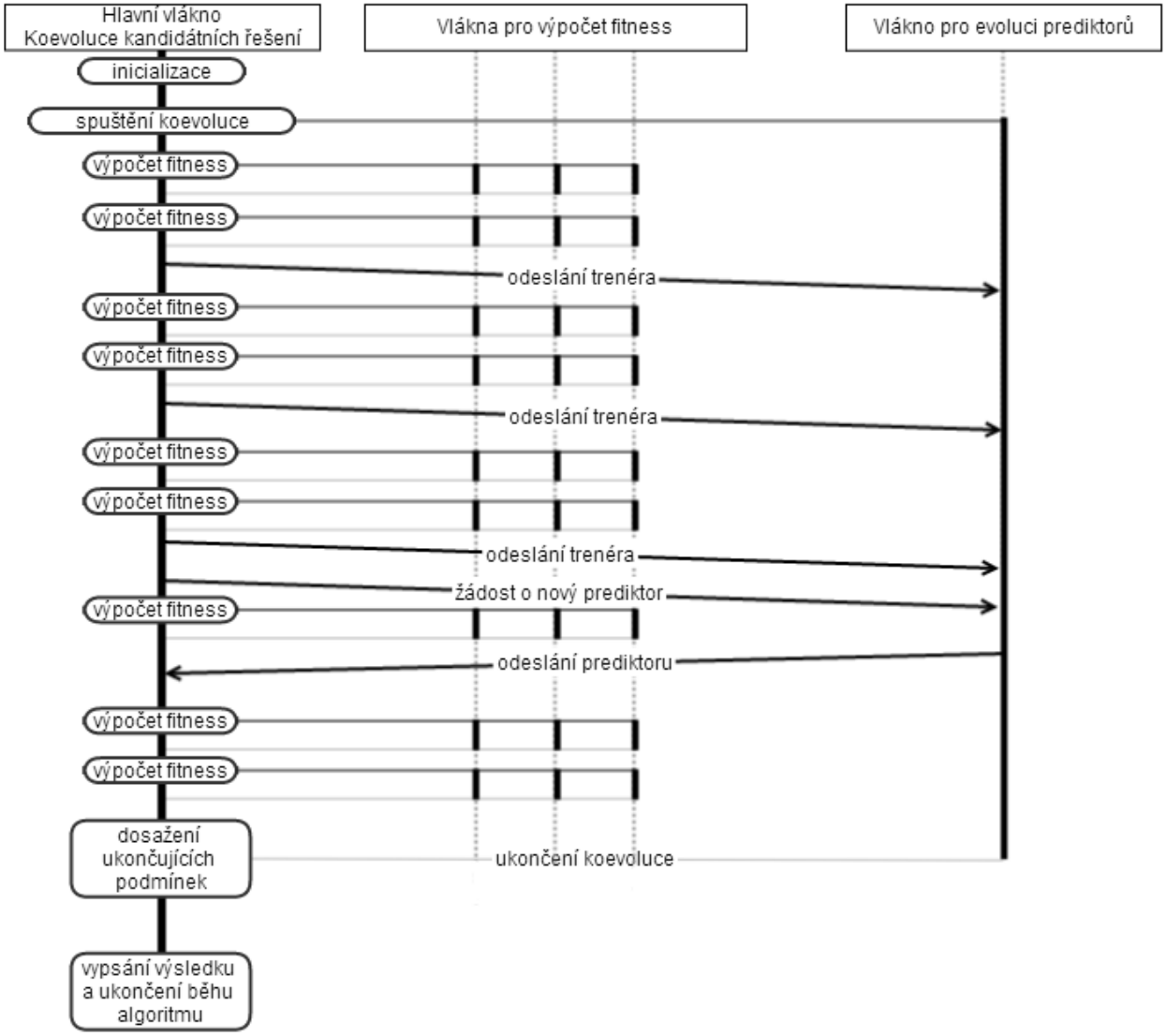
Je ovšem potřeba ověřit, zda se paralelizace výpočtu fitness opravdu vyplatí. Již samo využití predikce totiž určení fitness výrazně urychlí. Zároveň režie spojená s krátkodobým využitím více vláken není zcela zanedbatelná.

3.8.1 Synchronizace vláken

Běh obou hlavních vláken je vhodné alespoň částečně synchronizovat. Může se stát, že některé z vláken dostane k dispozici více výpočetního výkonu a jeho evoluce tak poběží rychleji, než bylo plánováno. Časově nejproměnlivější částí algoritmu zůstává výpočet fitness jedince. Potřebná doba se bude značně lišit podle velikosti právě používané predikované sady - tedy podle počtu vyčíslení, které je nutné při výpočtu fitness provést. Budou-li vlákna po delší časový úsek používat predikované sady s výrazně rozdílnou délkou, pravděpodobně se to promítne do rozdílu v rychlosti jejich běhu.

Kdyby byl použit genetický algoritmus, mohlo by mít výrazné opožďování vláken negativní vliv na diverzitu populace. Kvůli neměnným podmínkám by hrozilo *přeučení*. Kartézské genetické programování k tomu není tolik náchylné, do další generace postupuje vždy jen jeden rodič, takže o žádné diverzitě nemůže být řeč. Nicméně platí, že pokud by kandidátní řešení nedostávala pravidelný přísun adaptovaných prediktorů, jejich vlastní vývoj by to výrazně zpomalilo. A pokud by byly objeveny optimální intervaly pro komunikaci mezi vlákny, pak by jakékoli odchylky mohly mít negativní vliv. Všechny tyto dopady je však velice těžké dopředu předjímat.

Synchronizaci by bylo možné řešit zpožďováním jednoho z vláken, respektive bariérou, na které by na sebe vlákna čekala. V případě vytíženého systému by tak zpožděné vlákno mohlo získat více procesorového času ke svým výpočtům. V ostatních případech by se tak ale cenným výpočetním časem naopak plýtvalo. Jako za jakousi formu synchronizace lze považovat fakt, že je veškerý přenos jedinců řízen vláknem s evolucí kandidátních řešení. V evoluci prediktorů je alespoň jakási bezpečnostní kontrola, kdy je odeslání prediktorů odloženo o několik desítek iterací, pokud je jeho fitness příliš špatná. Bude však třeba otestovat, jaké jsou nejvýhodnější parametry tohoto mechanismu a zda není ve skutečnosti spíše kontraproduktivní.



Obrázek 3.4: Příklad paralelizace pomocí vláken a jejich vzájemná komunikace.

Kapitola 4

Implementace

Program byl napsán v jazyce C++ ve dvou variantách. Jedna z verzí pracuje v terminálu, druhá varianta obsahuje nadstavbu v podobě grafického uživatelského rozhraní. V kódu byly použity některé součásti standardu `C++11`, a proto je při překladu nutné použít překladač, který tento standard podporuje - například `GCC` verze 4.6 a vyšší. Překlad verze s grafickým uživatelským prostředím lze provést pomocí `Qt Creatoru` verze 5 a vyšší.

4.1 Paralelizace

V podkapitole 3.8 bylo uvedeno, že souběžná evoluce každé ze dvou populací bude probíhat paralelně. Pro implementaci paralelizace byla zvolena knihovna *OpenMP*. Ta umožňuje poměrně rychlou a snadnou paralelizaci zdrojového kódu pomocí direktiv pro kompilátor. Jednotlivé části kódu se provádějí v samostatných vláknech se sdílenou pamětí. Při mezivláknové komunikaci stačí zajistit exkluzivní přístup při zápisu do sdílených proměnných. Navíc se neomezí přenositelnost aplikace - knihovnu *OpenMP* lze dnes považovat za standardní knihovnu pro paralelní programování se sdílenou pamětí, a je tak podporována většinou operačních systémů a architektur procesorů. Za určitou nevýhodu lze považovat skutečnost, že některé aspekty paralelního běhu programu jsou řešeny zcela automaticky a programátor nad nimi má jen omezenou kontrolu.

Může se stát, že aplikace bude spuštěna v prostředí, kde *OpenMP* není podporováno, nebo bude k dispozici pouze jediné vlákno. Pro tyto případy byla implementována i sériová verze algoritmu - metoda `Coevolution::sequenProcess()`. Není však příliš vhodné ji používat - běh metody je velice pomalý a také nebylo dostatečně ověřeno, zda dosažené výsledky opravdu odpovídají vícevláknové verzi.

Paralelní evoluce obou populací

Základní algoritmus pro evoluci populace v kartézském genetickém programování je implementován v rodičovské třídě `cgp`. Z ní je odvozena třída pro evoluci kandidátních řešení `EvoSolution` a třída pro evoluci prediktorů `EvoPrediktor`. Paralelní běh evoluce obou populací je implementován ve třídě `Coevolution`, v metodě `run()`. Jsou zde vytvořena dvě vlákna. První vlákno provádí evoluci kandidátních řešení - metodu `SolutionProcess()`. Pokud je zapnuto použití koevoluce, pak je ve druhém vlákně spuštěna evoluce prediktorů - metoda `PredictorProcess()`. V opačném případě se druhé vlákno okamžitě ukončí. Obě metody pracují v nekonečné smyčce tak dlouho, dokud má sdílená proměnná `running` hodnotu true. V každé iteraci se voláním metody `evolve()` provede jeden krok evoluce -

ohodnocení jedinců a vytvoření následující generace. Dále se v pravidelných intervalech po stanoveném počtu iterací provádějí určité akce, které shrnuje tabulka 4.1. U intervalů jsou uvedeny hodnoty ve výchozím nastavení.

vlákno	**interval**	**akce**
evoluce řešení	400	Odeslání nového trenéra.
evoluce řešení	2000	Odeslání požadavku o zaslání nového prediktoru.
evoluce řešení	100	Kontrola, zda nebyl doručen nový prediktor.
evoluce řešení	1	Kontrola, zda nebylo nalezeno řešení.
evoluce řešení	1000	Sběr statistických dat o průběhu koevoluce.
evoluce prediktoru	10	Výměna jednoho z náhodných trenérů v archivu
evoluce prediktoru	5	Kontrola, zda nebyl doručen nový trenér.
evoluce prediktoru	5	Kontrola požadavku na zaslání prediktoru.

Tabulka 4.1: Periodicky se opakující akce v průběhu koevoluce.

Mezi oběma vlákny probíhá výměna jedinců. Odesíláni jsou noví trenéři a nejlépe ohodnocené prediktory. Příslušné metody pro odesílání jedinců jsou umístěny ve třídě `GeneMail`. Tam je definováno pole, kam je zkopírován genotyp odeslaného jedince. Zároveň je nastaven příznak doručení nového genotypu na hodnotu *true*. Veškeré zápisy a čtení z tohoto pole a příznaku jsou součástí kritické sekce. Příjemce periodicky kontroluje hodnotu příznaku doručení. Nově doručený genotyp je použit na inicializaci jedince a příznak doručení je nastaven na *false*. Při doručení nového trenéra je okamžitě vypočítána jeho skutečná fitness. Dvojice trenér - fitness je poté vložena do archivu. Pokud je doručeným jedincem nový prediktor, pak je určena jím predikovaná sada. Odkaz na tuto sadu je následně přidělen všem kandidátním řešením.

Paralelizace výpočtu fitness kandidátních řešení

Pokud má aplikace k dispozici více než dvě vlákna, využije je k souběžnému výpočtu fitness hodnoty více kandidátních řešení najednou. Je toho dosaženo použitím direktivy `#omp pragma parallel for`. Jak již bylo zmíněno, velikost populace kandidátních řešení je přizpůsobena počtu vláken tak, aby bylo možné jedince mezi vlákna rozdělit rovnoměrně a maximalizovat tak výkon aplikace. Tento přístup má však i své úskalí. Způsob, jakým program s vlákny nakládá, může být implementován odlišně u každé architektury procesoru. Ve většině případů jsou vlákna po skončení své činnosti uložena do zásobníku. U následující paralelní sekce jsou pak znovu využita již existující vlákna. Pokud by však zásobník vláken použit nebyl, probíhala by tvorba a rušení vláken při každém výpočtu fitness. Vzhledem k tomu, že samotný výpočet fitness jedince probíhá relativně krátkou dobu, mohl by v tomto případě paralelní výpočet dokonce způsobit zpomalení algoritmu. Nelze jednoduše zjistit, jakým způsobem daný procesor s vlákny pracuje. Možným řešením by bylo při startu

aplikace spustit srovnatelný kus kódu paralelně a sériově a ověřit, zda je opravdu paralelní verze rychlejší. Tato kontrola ale implementována nebyla. Proto pokud se zdá, že výpočet probíhá nezvykle pomalu, je vhodné pokusit se nastavit počet dostupných jader na hodnotu 2.

Oddělení výpočtu od uživatelského rozhraní

Další stupeň paralelizace byl použit u verze programu s grafickým uživatelským rozhraním. Všechny procedury, jejichž provedení trvá delší dobu, je obecně vhodné oddělit od uživatelského rozhraní a úmistit je do samostatného vlákna. Jinak hrozí, že rozhraní přestane během výpočtu reagovat na vstup uživatele a „zamrzne“. Grafické uživatelské rozhraní bylo implementováno s použitím *Qt frameworku*. Pro paralelizaci byla v tomto případě kvůli lepší provázanosti použita třída `QThread`, jež je součástí *Qt*. Při spuštění koevoluce je vytvořeno a spuštěno vlákno `WorkerThread`, které hlavnímu vláknu pravidelně odesílá informace o aktuálním stavu koevolučního procesu. Grafické rozhraní pak tyto údaje průběžně zobrazuje a zaznamenává. Vlákno `WorkerThread` ukončí svou činnost ve chvíli, kdy jsou splněny podmínky pro ukončení koevoluce, anebo pokud je proces přerušen či pozastaven uživatelem.

4.2 Sběr statistických dat

Při běhu (ko)evolučního algoritmu se v pravidelných intervalech zaznamenávají údaje aktuálním stavu evoluce obou populací. Interval sběru informací byl stanoven na 1000 generací evoluce kandidátních řešení. Jsou sledovány následující údaje:

- počet dosud provedených iterací u evolucí obou populací
- počet dosud provedených vyčíslení trénovacích vektorů u všech jedinců
- skutečná i predikovaná fitness nejlépe ohodnoceného jedince z populace kandidátních řešení
- fitness právě používaného prediktoru
- velikost právě používané predikované sady
- indexy právě predikovaných trénovacích vektorů

Informace o aktuální fitness a velikosti predikované sady jsou vypisovány v průběhu evoluce. Zároveň je vypsán každý jedinec, jenž má lepší fitness než všichni jedinci nalezení před ním. Při ukončení koevoluce se vypíše nalezené řešení společně s počtem generací a vyčíslení trénovacích vektorů. Pokud řešení nebylo nalezeno, vypíše se alespoň nejlepší nalezený vzorec a údaj o hodnotě jeho fitness. Statistická data o predikci, počtu kol a vyčíslení trénovacích vektorů jsou shromažďována napříč mnoha opakováními koevoluce. V závěru běhu programu se vypíší následující údaje:

- Procentuální úspěšnost nalezení řešení.
- Údaje o počtu generací, počtu vyčíslení a času provádění. Vše je uvedeno v kvartilech, aby bylo možno snadno sestavit kvartilový graf.
- Histogram velikosti predikovaných sad.

- Histogram četnosti výběru konkrétních trénovacích vektorů.

U verze s grafickým uživatelským rozhraním se histogramy pro predikci nezobrazují a ostatní statistické informace se zobrazují průběžně - údaje jsou aktualizovány po skončení každé (ko)evoluce.

4.3 Struktura souborů s trénovací sadou

Trénovaci sady jsou načítány ze souborů s pevně danou strukturou. Začínají řádkem s dvojicí čísel - první udává počet nezávislých proměnných a druhé počet závislých proměnných. Dále následují jednotlivé trénovací vektory. Na každém řádku je uložen jeden vektor. Hodnoty proměnných jsou odděleny libovolným počtem bílých znaků. Nejprve jsou zapsány nezávislé proměnné a za nimi následují závislé proměnné v pevně daném pořadí. Nepovinnou součástí souboru je řádek uvozený značkou `[description]`. Za ní následuje stručný text popisující danou trénovací sadu. Ten se vypíše po jejím načtení. V souboru mohou být použity komentáře - jakýkoli text na řádku následující za značkou // je ignorován. Také jsou vynechány všechny řádky obsahující pouze bílé znaky. U čísel je pro oddělení desetinných míst možno použít tečku i čárku.

Počet hodnot na řádku by měl odpovídat deklarovanému počtu závislých a nezávislých proměnných. Pokud je na řádku hodnot více, jsou přebývající hodnoty ignorovány. Je-li jich naopak méně nebo text na řádku nereprezentuje číslo, vypíše se chyba a daný trénovací vektory je ignorován. Nyní následuje příklad trénovací sady s deseti vektory.

```
//komentar - tento radek bude ignorovan
1 1
[description] a= x^2-x^3 ; 10 vzorku v~intervalu <-10.. -9.1>
-10     1100 //-10 je nezavisla 1100 je zavisla promenna
-9.9 1068,309
-9.8 1037.232
-9.7 1006.7630
-9.6 976.8960
-9.5 947.6250
-9.4 918.9440
-9.3 890.8470
-9.2 863.3280
-9.1 836.3810
```

Jak již bylo zmíněno v části 3.6.2, trénovací vektory by měly být řazeny podle hodnoty nezávislé proměnné a rozestupy v hodnotách nezávislých proměnných by měly být konstantní. Toto je potřeba zajistit již v souboru s trénovací sadou. Pokud je použito více nezávislých proměnných, je jim v rámci zjednodušení náhodně přiřazena priorita. Data jsou poté řazena postupně podle priority jednotlivých proměnných.

4.4 Nastavení parametrů koevoluce

U koevolučního algoritmu je značné množství parametrů, jejichž hodnota může ovlivnit průběh koevoluce. U jedinců obou populací lze změnit množinu použitých funkcí, konstanty, rozměry mřížky uzlů a L-back hodnotu. Dále je zde míra mutace, velikost obou

populací, velikost archivu, nastavení frekvence komunikace mezi vlákny, ukončující podmínky a další. Nastavovat všechny tyto parametry jen pomocí argumentů příkazové řádky by bylo značně zdlouhavé. Proto je použit konfigurační soubor ve formátu xml. Do něj se zapíší všechna požadovaná nastavení a následně se aplikují spuštěním programu s parametrem `-s [nazev_souboru.xml]`. Aby nebylo nutné ručně psát celý konfigurační soubor, je možné nechat vygenerovat soubor s výchozím nastavením a v něm pak pouze provést požadované úpravy. Nejčastěji upravované parametry (maximální počet iterací, fitness toleranci, vypnutí koevoluce a další) je také pro urychlení možné nastavit přímo pomocí parametrů příkazové řádky. Nastavení jsou aplikována v následujícím pořadí:

- Parametry zadané jako argumenty programu mají nejvyšší prioritu.
- Pokud byl zadán soubor s nastavením, pak jsou z něj načteny všechny parametry kromě těch, které byly zadány jako argumenty programu.
- Pokud některý z parametrů nebyl nastaven žádným z výše uvedených způsobů, bude pro něj použita výchozí hodnota.

PDF dokument s popisem struktury souboru nastavení je umístěn na přiloženém CD.

I přes veškerou snahu se výše uvedený přístup nemusí zdát dostatečně uživatelsky komfortní. Snaha ještě více zjednodušit ovládání programu byla hlavní motivací pro vytvoření varianty aplikace s grafickým uživatelským rozhraním. U této verze je možné veškeré parametry zadat v záložce `Nastavení` úpravou hodnot v příslušných polích.

4.5 Výpis jedince v textové podobě

Při zobrazení nejlepších nalezených kandidátních řešení i průběžných výsledků se vypisuje matematický vztah, který daný jedinec představuje. Průběh sestavení tohoto textového řetězce je obdobný jako při určování aktivních uzlů. Opět se prochází aktivní uzly jedince tvořící syntaktický strom. Začíná se u kořene - primárního výstupu - a pokračuje se až po dosažení listů stromu, což mohou být primární vstupy nebo konstanty. Na rozdíl od zjišťování aktivních uzlů není v tomto případě tolik kritická rychlost provedení. Namísto práce se zásobníkem byl proto použit elegantnější způsob implementace pomocí rekurzivní funkce.

Názvy vstupních a výstupních proměnných jsou pevně nastaveny. Nezávislým proměnným jsou přiřazena písmena ze začátku abecedy: a, b, c, d ... Pro závislé proměnné byla použita písmena x, y, z, v, u, t... . Navíc se provádí kontrola priority funkcí uzlů (či lépe řečeno, priority aritmetických operací, které dané uzly vykonávají). To umožňuje odstranění všech redundantních závorek. Díky tomuto jsou výsledné rovnice mnohem čitelnější.

Kapitola 5

Experimenty

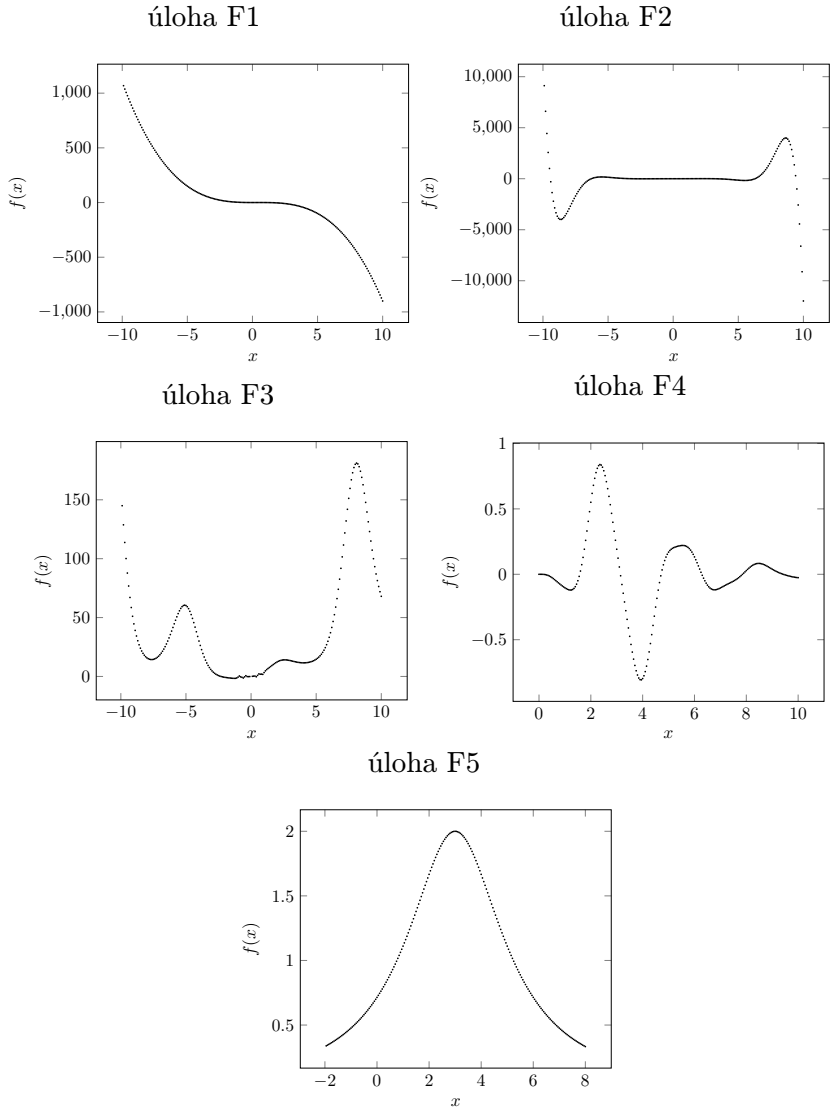
Otestování vlastností navrženého koevolučního algoritmu bylo provedeno na úlohách nalezení matematického vztahu pro zadanou trénovací sadu. Pro teto účel byla vybrána pětice úloh přejatých z článku [9]. Jsou shrnuty v následující tabulce:

Označení	Hledaný vztah	Definiční interval
$F1$	$f(x) = x^2 - x^3$	$x \in \langle -10, 10 \rangle$
$F2$	$f(x) = e^{\lvert x \rvert} \sin(x)$	$x \in \langle -10, 10 \rangle$
$F3$	$f(x) = x^2 e^{\sin(x)} + \sin\left(\frac{\pi}{x^3}\right)$	$x \in \langle -10, 10 \rangle$
$F4$	$f(x) = e^{-x} x^3 \sin(x) \cos(x) \left(\sin(x)^2 \cos(x) - 1\right)$	$x \in \langle 0, 10 \rangle$
$F5$	$f(x) = \frac{10}{(x-3)^2+5}$	$x \in \langle -2, 8 \rangle$

Tabulka 5.1: Seznam funkcí použitých při experimentech.

Úlohy jsou seřazeny vzestupně podle mediánu počtu generací, po kterých bylo v původním článku nalezeno řešení bez použití koevoluce. V dalším textu budou označením *jednodušší úlohy* myšleny úlohy F1, F2 a F3. Pojem *složitější úlohy* bude zahrnovat úlohy F4 a F5. Pro každou úlohu byla vytvořena trénovací sada o velikosti 201 trénovacích vektorů (výjimkou je F3 s 200 vektory, neboť v bodě $x = 0$ nemá definovanou hodnotu). Trénovací vektory jsou rozmístěny rovnoměrně v celém zkoumaném intervalu. Na obrázku 5.1 jsou zachyceny grafy s hodnotami trénovacích vektorů pro jednotlivé trénovací sady. Experimenty byly pro každou funkci a nastavení prováděny opakovaně. U úloh F1, F2 a F3 byla statistická data získána ze 100 nezávislých běhů. Pro úlohy F4 a F5 bylo provedeno pouze 30 běhů, a to z důvodu vysoké časové náročnosti. Během experimentů byly sledovány především statistické údaje o tom, po jakém počtu generací se podařilo nalézt řešení a kolik bylo celkově provedeno vyčíslení trénovacích vektorů (tedy kolikrát byla zjišťována výstupní hodnota jedince pro zadaný vstup). U koevoluce byly také sestaveny histogramy velikosti a složení nejlépe ohodnocených predikovaných sad.

Ve statistikách jsou zahrnuty pouze údaje z úspěšných běhů. Pokud se nepodařilo nalézt řešení během stanoveného maximálního počtu generací, pouze se upravil údaj o úspěšnosti koevoluce a byl vypsán nejlepší nalezený jedinec společně s hodnotou fitness, které se mu podařilo dosáhnout.



Obrázek 5.1: Trénovací sady u jednotlivých úloh

5.1 Nastavení evoluce kandidátních řešení

Nastavení popsaná v této sekci jsou společná pro běh algoritmu s použitím koevoluce i bez ní. Parametry evoluce kandidátních řešení byla přejata z článku [9]. Zahrnují nastavení velikosti populace, parametry genotypu jedinců, míru mutace, toleranci při výpočtu fitness a také ukončující podmínky evoluce - maximální počet generací a minimální akceptovatelné skóre řešení. Záměrem je, aby výsledky dosažené pouze pomocí základního CGP algoritmu bez použití koevoluce byly obdobné jako ve zmíněném článku. V takovém případě pak bude možné vzájemně porovnat výsledky, kterých bylo dosaženo s použitím rozdílných variant koevoluce. Nastavení pro evoluci kandidátních řešení je uvedeno v tabulce 5.2.

Parametr	Hodnota
velikost populace CGP	12
počet řádků mřížky CGP	1
počet sloupců mřížky CGP	32
L-back parametr	32
počet mutovaných genů	1 - 8
množina funkcí uzlů	$i_1 + i_2$, $i_1 - i_2$, $i_1 \cdot i_2$, i_1/i_2, $\sin(i_1)$, $\cos(i_1)$, e^{i_1}; F2: navíc $\lvert i_1 \rvert$
konstanty	F3: π; F5: 3, 5
tolerance u výpočtu fitness skóre	F1: 0.5, F2: 0.5, F3: 1.5, F4: 0.025, F5: 0.025
požadované fitness skóre u řešení	97%
maximální počet generací	8 000 000

Tabulka 5.2: Shrnutí nastavení evoluce kandidátních řešení.

5.2 Nastavení koevoluce

Parametry uvedené v této sekci se týkají nastavení evoluce prediktorů a koevoluce obecně. Tabulka 5.3 tyto parametry shrnuje. Většina hodnot byla stanovena na základě experimentů prováděných s trénovací sadou F3 a také $f(x) = 6 \cdot \sin(x) \cdot cos(x)$. Koevoluční algoritmus nalezne řešení pro tyto dvě úlohy relativně rychle, ale zároveň proces trvá dostatečně dlouho na to, aby se efekt odlišného nastavení stačil projevit.

5.2.1 Velikost populace prediktorů

Velikost populace prediktorů byla nastavena na hodnotu 5, což je u CGP často používaná velikost populace. Bylo provedeno několik experimentů s vyššími hodnotami, ovšem nebylo pozorováno žádné výraznější zrychlení nebo zvýšení úspěšnosti prohledávání.

5.2.2 Míra mutace prediktorů

Tento parametr udává minimální a maximální počet mutovaných genů u evoluce prediktorů. Pokud byla míra mutace příliš nízká, kódovala značná část jedinců nové generace stejný matematický vztah jako rodič. Důsledkem pak byla delší doba běhu koevoluce. Při vysoké míře mutace se zase začali v populaci mnohem častěji objevovat neplatní jedinci, kteří nesplňovali podmínku minimální velikosti predikované sady a byli okamžitě vyřazeni. Jako horní hranice počtu mutovaných genů byly testovány hodnoty 15%, 20%, 30% a 40% genů. Pro minimální počet byly vyzkoušeny hodnoty v rozmezí od 1 genu po 10% genů v genotypu. Nejlépe se osvědčilo nastavení mutace v rozmezí od 1 genu až po 30% genů v genotypu.

5.2.3 Intervaly výměny trenérů a prediktorů

Tyto parametry představují počet iterací evoluce kandidátních řešení, po kterém bude odeslán nový trenér do archivu, respektive po kterém bude odeslán pokyn k zaslání nového prediktoru. Pro oba parametry bylo testováno několik značně rozdílných hodnot, aby bylo

možné stanovit, jaký má tento parametr vliv na průběh koevoluce. Noví trenéři byli zasíláni v intervalech 100, 400 nebo 800 generací, pro odesílání prediktorů byly vyzkoušeny hodnoty 500, 1000, 2000. Obecně se dá říci, že s kratšími intervaly se mnohem rychleji a dramatičtěji měnila velikost prediktoru a složení predikované sady. U úloh F3, F4 a F5 bylo dosaženo výrazně lepších výsledků s delšími intervaly. U úlohy F1 se však touto změnou zvýšil průměrný počet potřebných generací a vyčíslení přibližně na dvojnásobek. Stejně tak došlo ke zhoršení u úlohy F2, ačkoli nebylo tak výrazné.

5.2.4 Velikost archivu trenérů

Počet trenérů v archivu ovlivňuje evoluci prediktorů z několika hledisek. Čím je archiv větší, tím déle trvá ohodnocení populace prediktorů a tím pomalejší je tedy jejich vývoj. Zároveň ale platí, že čím rozmanitější sada trenérů je použita při ohodnocení prediktoru, tím lépe lze stanovit jeho skutečnou použitelnost. U většího archivu se také po vložení nového trenéra méně změní podmínky pro výpočet fitness. Populace prediktorů se tak pomaleji adaptuje na aktuální stav populace kandidátních řešení. Velikost archivu byla původně nastavena na 8 jedinců, po řadě experimentů pak byla navýšena na hodnotu 10. Toto vedlo opět ke zvýšení úspěšnosti koevoluce u úloh F4 a F5 a zároveň se zvýšil počet vyčíslení u úlohy F3. U úloh F1 a F2 nebyla pozorována žádná výraznější změna.

5.2.5 Množina funkcí uzlů prediktoru

Sadu operací, které mohou vykonávat uzly prediktoru, tvoří $i_1 + i_2$, $i_1 - i_2$, $i_1 \cdot i_2$, i_1/i_2, $|i_1|$ a $-i_1$. Dále zde byly zahrnuty i funkce $\sin(i_1)$, $a \bmod b$, $\min(a, b)$ a $\max(a, b)$. Jak bylo uvedeno v části 3.6.3, použitím těchto funkcí je prediktoru dána možnost výrazněji měnit velikost predikované sady a také způsob, jakým lze vybrat i nepravidelně rozmístěné trénovací vektory.

5.2.6 Množina konstant prediktoru

Prediktor může do svého matematického vztahu zahrnout následující konstanty: 2, 4, 8, $\frac{s}{4}$, $\frac{s}{2}$ a s. Symbol s představuje velikost trénovací sady. Všechny tyto hodnoty se mohou v průběhu koevoluce vyvíjet. Více je o výběru a evoluci konstant uvedeno v části 3.6.5.

Parametr	Hodnota
velikost populace prediktorů	5
počet řádků mřížky CGP	2
počet sloupců mřížky CGP	15
L-back parametr	4
počet mutovaných genů	1 - 30%
množina funkcí uzlů	$i_1 + i_2$, $i_1 - i_2$, $i_1 \cdot i_2$, i_1/i_2, $\lvert i_1 \rvert$, $-i_1$, $\sin(i_1)$, $\min(i_1, i_2)$, $\max(i_1, i_2)$, $i_1 \bmod i_2$
množina konstant prediktoru	2, 4, 8, $\frac{s}{4}$, $\frac{s}{2}$, s
interval odesílání trenéra	každých 400 generací
interval odesílání prediktoru	každých 2000 generací

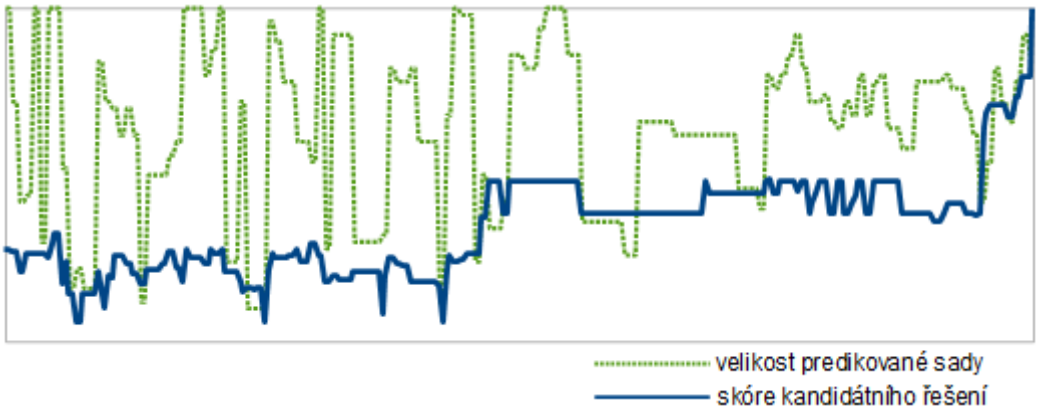
Tabulka 5.3: Souhrn nastavení koevoluce.

5.3 Průběh koevoluce

Graf na obrázku 5.2 ukazuje, jak se mění velikost predikované sady a skutečná fitness kandidátních řešení v průběhu koevoluce. Jako příklad zde byl vybrán jeden z běhů koevoluce u úlohy F3, u něhož jsou jednotlivé fáze dobře rozlišitelné. V počáteční fázi koevoluce, dokud je fitness kandidátních řešení nízká, se velikost sady často a výrazně mění v závislosti na aktuálním obsahu archivu. V této fázi archiv obsahuje dosti rozmanitou množinu matematických vztahů. Proto jsou více preferovány univerzálnější predikované sady o větší délce. Na zvolenou metodu výpočtu fitness má negativní dopad, pokud je do archivu vloženo více trenérů s abnormálně vysokou odchylkou fitness. To se stává zejména u kandidátních řešení s nízkou fitness. Důsledkem mohou být ony krátké, ale výrazné propady velikosti.

Pomineme-li výběr správných trénovacích vektorů, dá se obecně říci, že čím je predikovaná sada větší, tím je predikce přesnější. Ve většině případů platí, že pokles velikosti predikované sady má za následek i pokles fitness kandidátních řešení. Při následném růstu velikosti pak zpravidla roste i fitness řešení, mnohdy i nad svou původní hodnotu. Jedná se o příklady toho, jak může změna predikované sady pomoci při uváznutí kandidátních řešení v lokálním optimu.

Jakmile fitness kandidátních řešení dostatečně vzroste, změní se zpravidla i charakter evoluce prediktorů. V archivu se začnou více objevovat trenéři kódující obdobné matematické vztahy, které více odpovídají trénovací sadě. Predikce se začne více zaměřovat na správné ohodnocení těchto konkrétních vztahů. Může být nalezena predikovaná sada o menší velikosti, která postačuje pro dostatečně přesné ohodnocení momentálně nejlepšího kandidátního řešení. V grafu je vidět, že velikost predikované sady se v této fázi již nemění tak výrazně. Kvůli snížení rozmanitosti trenérů v archivu zde hrozí přeučení prediktoru, což může mít za následek prudký pokles fitness kandidátních řešení. Proto je také polovina archivu generována náhodně a periodicky měněna.



Obrázek 5.2: Příklad průběhu koevoluce u úlohy F3.

5.4 Výsledky experimentů

V této části budou shrnuty údaje z experimentů, provedených pro každou z pěti výše popsaných úloh. Cílem je porovnat, jakých výsledků navržený program dosahuje při použití koevoluce v porovnání s výsledky klasického CGP algoritmu. Byla použita nastavení popsaná v předchozí části. V kvartilových grafech nejsou zobrazeny odlehlé hodnoty. Při jejich zobrazení se hodnoty kvartilů - to jest nejdůležitější informace obsažená v grafu - staly zcela nečitelné.

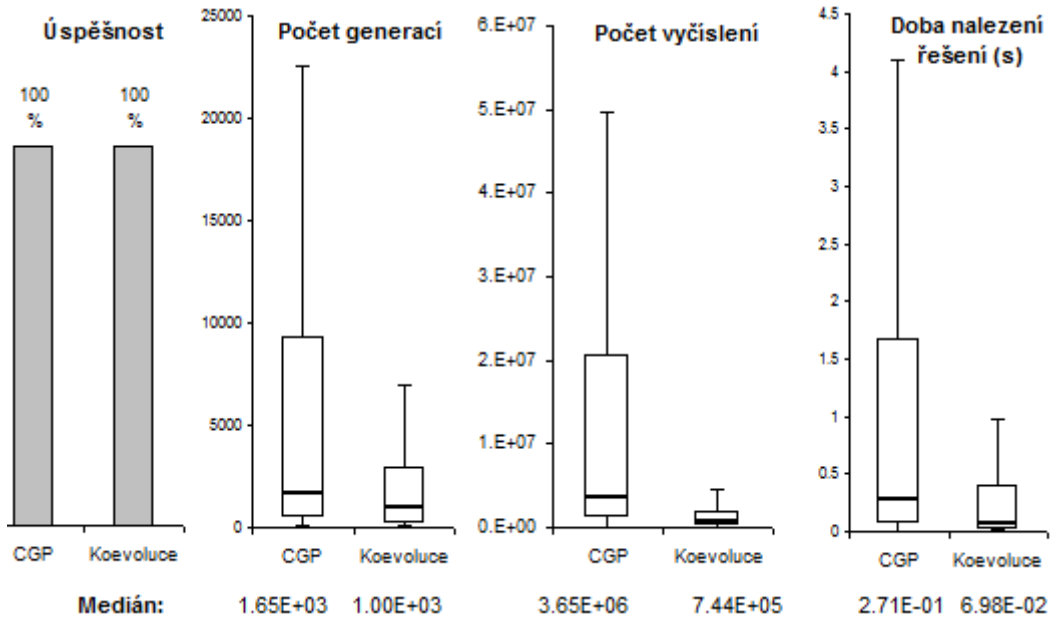
5.4.1 Úloha F1

Porovnání výsledků dosažených pomocí koevoluce oproti standardnímu CGP shrnuje obrázek 5.3. Je vidět, že počet generací potřebných k nalezení řešení klesl přibližně o 40%, ovšem potřebný počet vyčíslení byl téměř pětinásobně nižší, což se projevilo i v průměrné době nalezení řešení.

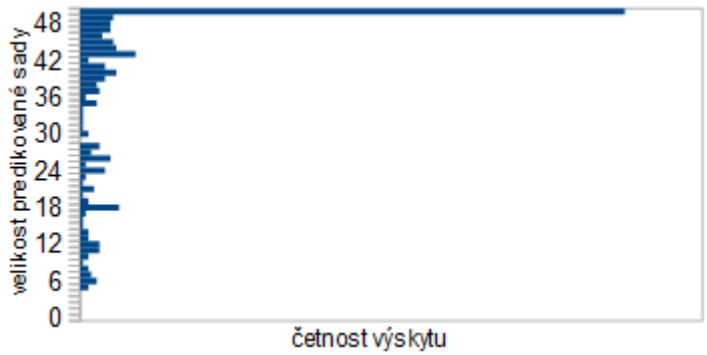
Na obrázku 5.4 je histogram délek nejlépe ohodnocených predikovaných sad a obrázek 5.5 zobrazuje četnost, s jakou nejlepší prediktory vybíraly konkrétní trénovací vektory [1]. V tabulce je pro názornost zachycen i průběh hledané funkce.

U této úlohy však zmíněné dva histogramy nemají příliš velkou vypovídací hodnotu. Řešení bylo zpravidla nalezeno během přibližně 1000 generací. Prediktor se přitom mění až každých 2000 generací. V histogramech jsou tak zachyceny převážně počáteční prediktory, vybrané z generace pěti náhodných jedinců, jejichž fitness byla určena pomocí archivu plného náhodných trenérů. Výběr trénovacích vektorů se v histogramu jeví jako zcela náhodný. Velikost predikované sady byla v naprosté většině případů maximální. To odpovídá pozorování, že při zvoleném nastavení výpočtu fitness prediktorů jsou preferovány přesné, byť rozsáhlé predikované sady. Jejich velikost může v průběhu koevoluce klesnout, pokud se podaří nalézt sadu, která je menší, ale rovněž relativně přesná.

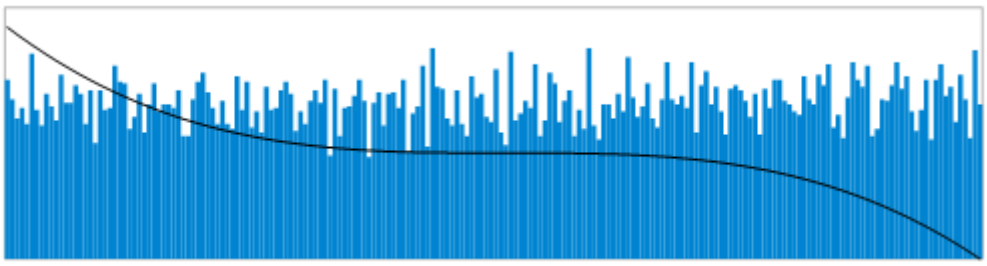
[1] Histogramy v této kapitole zachycují data shromažďovaná v celém průběhu koevoluce, nikoliv pouze parametry závěrečných prediktorů v okamžiku nalezení řešení.



Obrázek 5.3: F1 - porovnání výsledků dosažených bez koevoluce a s koevolucí.



Obrázek 5.4: F1 - velikost nejlépe ohodnocených predikovaných sad v průběhu koevoluce.

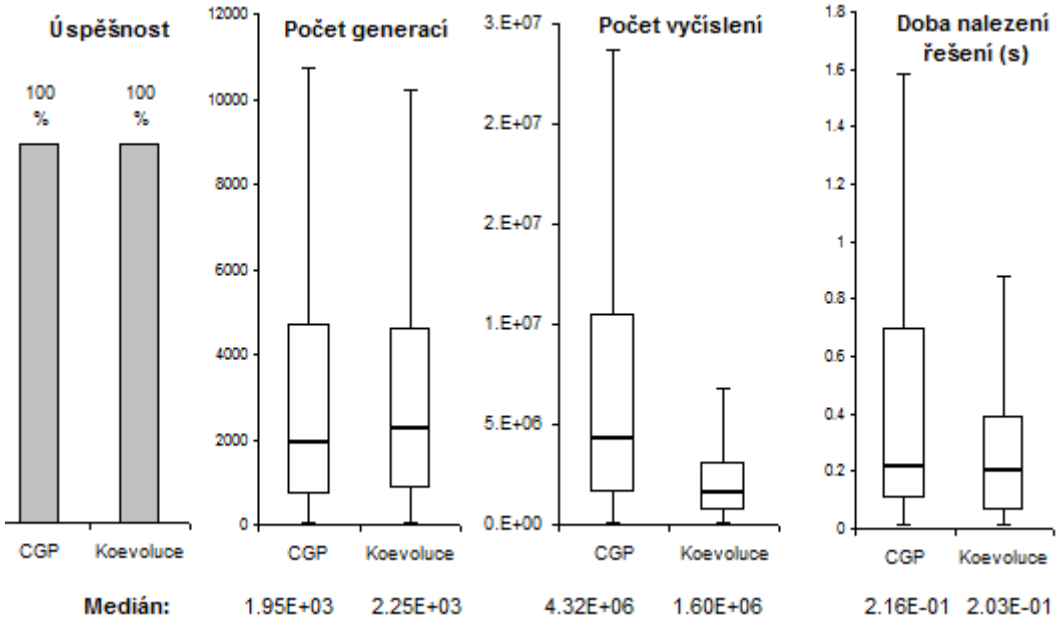


Obrázek 5.5: F1 - četnost výběru konkrétních trénovacích vektorů v průběhu koevoluce.

5.4.2 Úloha F2

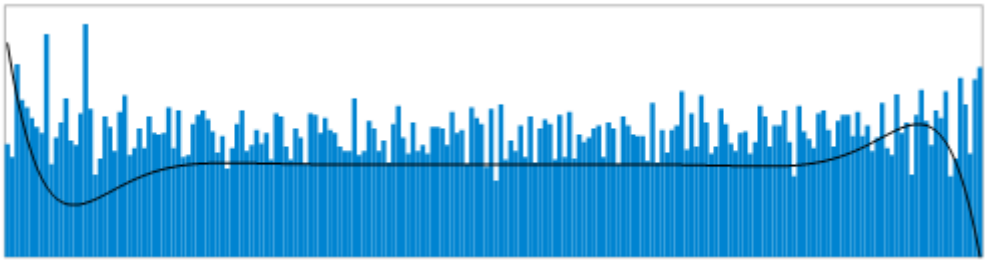
Vše, co bylo napsáno o úloze F1, do značné míry platí i pro úlohu F2. Jak je patrné z obrázku 5.6, medián počtu generací potřebných pro nalezení řešení je 2250. To znamená, že byly zaslány přibližně dva prediktory. Při použití koevoluce došlo dokonce k mírnému nárůstu počtu generací. Medián počtu vyčíslení je u koevoluce nižší o více než polovinu, ovšem suma uběhlého času je u obou verzí přibližně stejná - 0,2 vteřiny. Důvodem může být fakt, že na evoluci řešení je při koevoluci vyčleněno méně vláken a jedna iterace tak trvá déle. Rozdíl by ovšem neměl být takto markantní.

Z histogram 5.7 obsahuje délky všech prediktorů, které v průběhu koevoluce sloužily k ohodnocení kandidátních řešení. Opět je patrné, že prediktory nenalezly lépe ohodnocenou sadu než tu, která má maximální povolenou velikost. Na histogramu 5.8 lze pozorovat mírný nárůst výběru trénovacích vektorů ležících na okrajích trénovací sady, kde se nachází lokální extrémy v trénovací sadě.



Obrázek 5.6: F2 - porovnání výsledků dosažených bez koevoluce a s koevolucí.

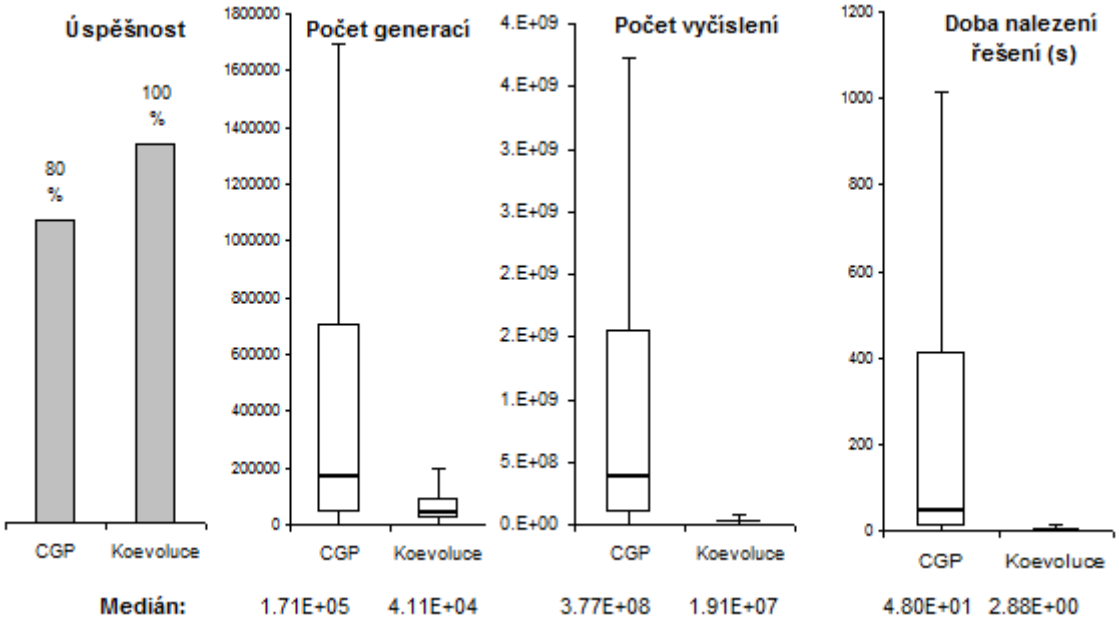
Obrázek 5.7: F2 - velikost nejlépe ohodnocených predikovaných sad v průběhu koevoluce.



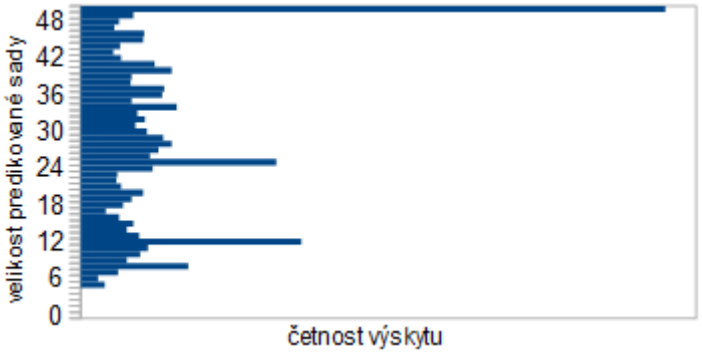
Obrázek 5.8: F2 - četnost výběru konkrétních trénovacích vektorů v průběhu koevoluce.

5.4.3 Úloha F3

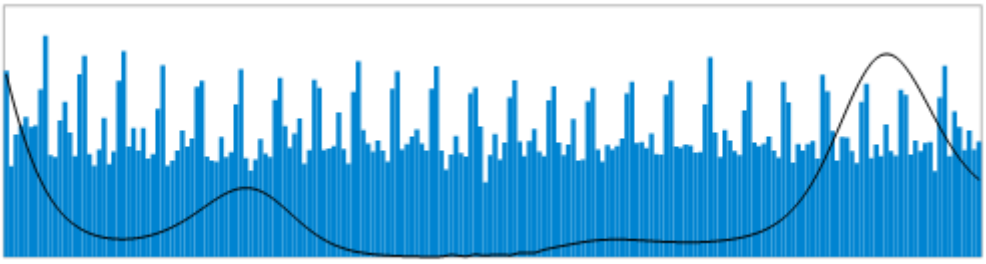
U této úlohy přineslo použití koevoluce výrazné zlepšení. Podle grafu 5.9 byl počet generací méně než čtvrtinový a počet vyčíslení se snížil dvacetinásobně. Podle histogramu 5.10 se u této úlohy ve významném počtu případů podařilo nalézt relativně krátké predikované sady o délkách 6, 12 a 24. Na histogramu 5.11 je jasně patrné pravidelné rozložení prediktory zvolených trénovacích vektorů.



Obrázek 5.9: F3 - porovnání výsledků dosažených bez koevoluce a s koevolucí.



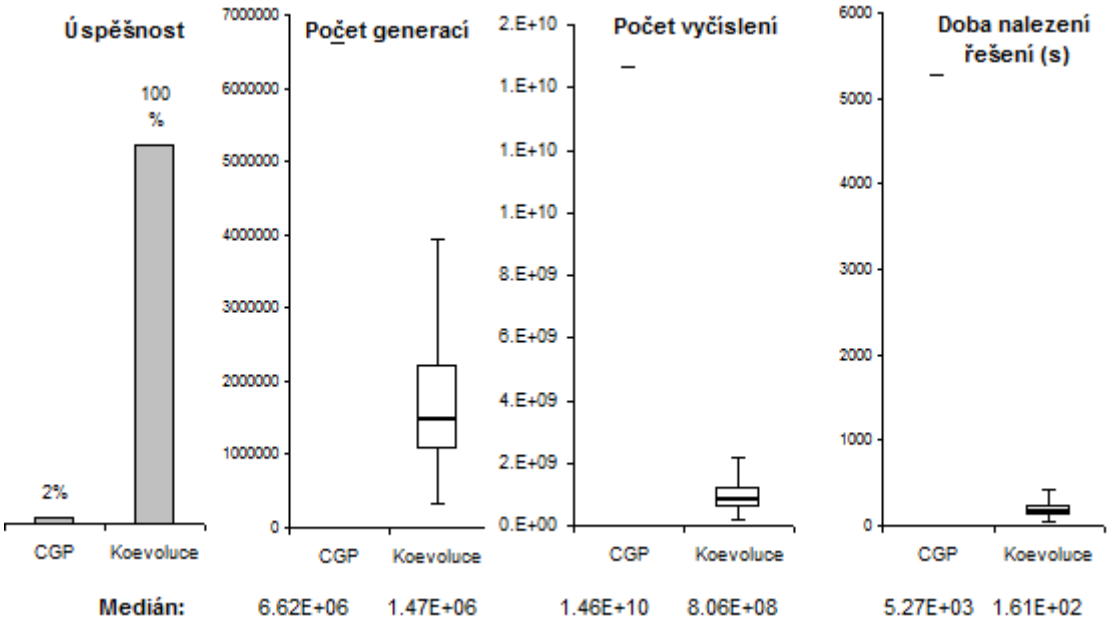
Obrázek 5.10: F3 - velikost nejlépe ohodnocených predikovaných sad v průběhu koevoluce.



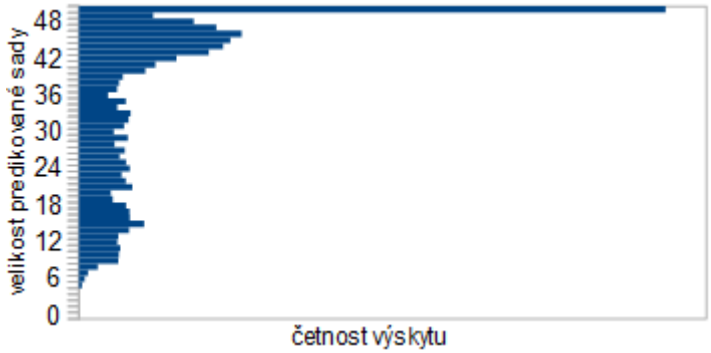
Obrázek 5.11: F3 - četnost výběru konkrétních trénovacích vektorů v průběhu koevoluce.

5.4.4 Úloha F4

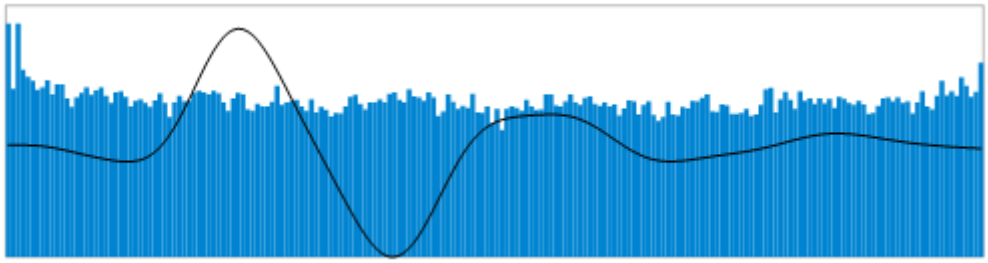
U úlohy F4 nebylo možné provést porovnání, neboť se nepodařilo shromáždit dostatek dat u varianty bez použití koevoluce. Celkem bylo provedeno 60 pokusů a pouze v jediném případě se CGP algoritmu podařilo nalézt řešení. V grafu 5.12 byla proto jako výsledek experimentu bez použití koevoluce použita jen tato jediná hodnota. U varianty s koevolucí bylo řešení nalezeno pokaždé. Jak ukazuje histogram 5.13, kromě maximálního povoleného limitu se velikost predikované sady také často pohybovala okolo hodnoty 40. Jednotlivé vektory však byly vybírány velice rovnoměrně. Podle histogramu 5.14 se nezdá, že by se u této úlohy výběr trénovacích vektorů jakkoliv adaptoval na průběh hledané funkce.



Obrázek 5.12: F4 - porovnání výsledků dosažených bez koevoluce a s koevolucí.



Obrázek 5.13: F4 - velikost nejlépe ohodnocených predikovaných sad v průběhu koevoluce.

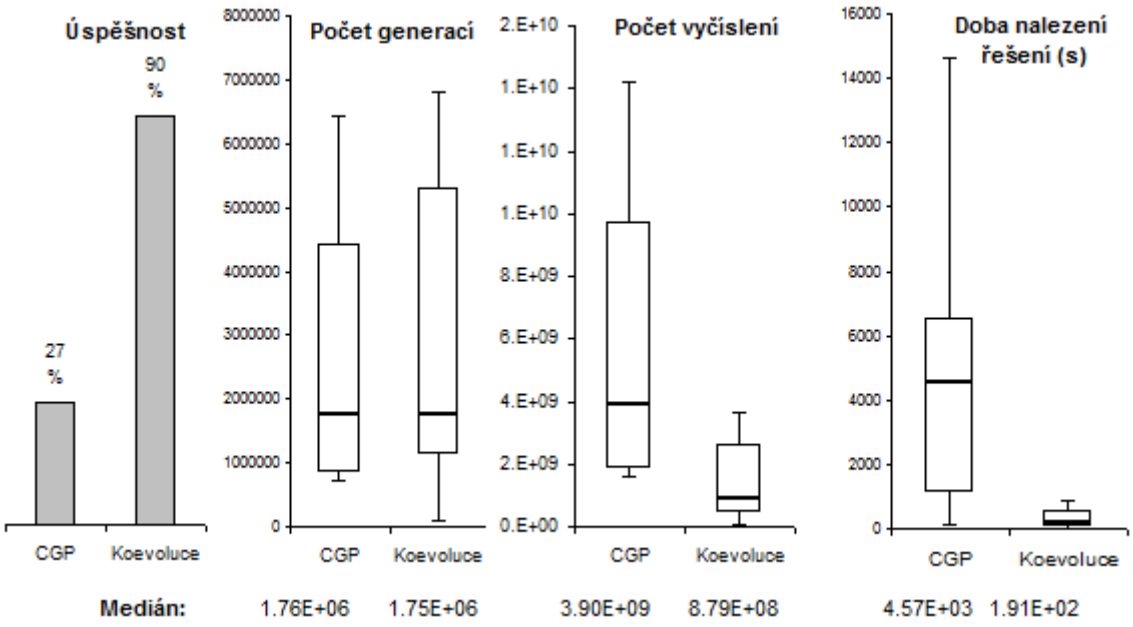


Obrázek 5.14: F4 - četnost výběru konkrétních trénovacích vektorů v průběhu koevoluce.

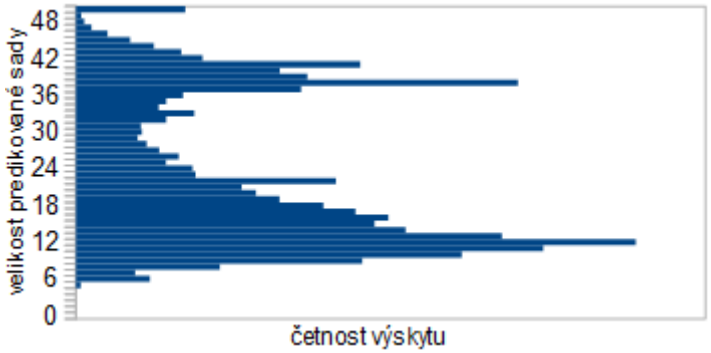
5.4.5 Úloha F5

Naměřené údaje z úlohy F5 jsou shrnuty v grafu na obrázku 5.15. Potřebný počet generací byl u obou variant přibližně stejný, ovšem počet provedených vyčíslení trénovacích vektorů byl s použitím koevoluce výrazně nižší (potřebný počet vyčíslení klesl téměř o 80%). Řešení bylo také nalezeno během 27násobně kratší doby. Bez použití koevoluce se podařilo nalézt řešení pouze v 27% případů, a dokonce ani u koevoluční verze nebylo řešení v 10% případů během zadaného počtu generací nalezeno.

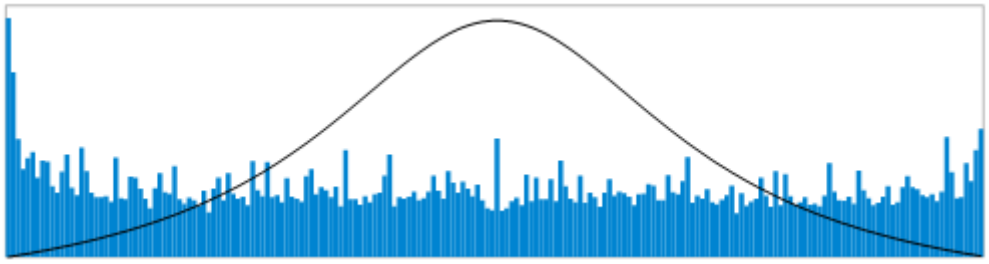
F5 je jediná ze zkoumaných úloh, u které byl preferován prediktor s kratší délkou predikované sady, než je maximální povolená hodnota. V histogramu 5.16 je zaznamenán největší výskyt velikostí okolo hodnoty 12. Dále se často vyskytovaly velikosti predikované sady v blízkosti hodnoty 38. Ačkoli je F5 pravděpodobně nejobtížnější úlohou ze zvolené sady úloh, samotný graf hledané funkce $f(x) = \frac{10}{(x-3)^2+5}$ má poměrně jednoduchý tvar. Je proto možné, že pro predikci fitness je 12 trénovacích vektorů dostačující počet. Podle histogramu na obrázku 5.17 byl výběr indexů dosti proměnlivý: Je vidět mírná preference vektorů na okrajích trénovací sady. Také byl často vybírán trénovací vektor, ve kterém má hledaná funkce maximum. Je ovšem třeba poznamenat, že výběr maxima funkce byl značně usnadněn faktem, že tento bod je umístěn uprostřed trénovací sady. Velikost poloviny trénovací sady je přitom jednou z konstant, která je součástí genotypu prediktorů. Je proto pravděpodobné, že kdyby se maximum nacházelo na jiné pozici - jako tomu bylo u úloh F3 a F4 - nebylo by vybráno tak jednoznačně.



Obrázek 5.15: F5 - porovnání výsledků dosažených bez koevoluce a s koevolucí.



Obrázek 5.16: F5 - velikost nejlépe ohodnocených predikovaných sad v průběhu koevoluce.



Obrázek 5.17: F5 - četnost výběru konkrétních trénovacích vektorů v průběhu koevoluce.

5.5 Porovnání s koevolucí využívající GA prediktor

V článku [9] jsou uvedeny výsledky experimentů provedených s koevolučním algoritmem, který pro evoluci prediktorů využívá genetický algoritmus tak, jak bylo popsáno v části 2.3.3. Velikost predikované sady je u této verze konstantní a je předem nastavena na co nejvýhodnější nalezenou hodnotu. Stejně jako tomu bylo v kapitole 3, v textu bude pro zjednodušení tato implementace označována termínem *GA predikce*, implementace popsaná v této diplomové práci bude označená jako *CGP predikce*. Experimenty byly provedeny na stejné množině funkcí za stejného nastavení evoluce kandidátních řešení. Díky tomu je možné výsledky obou přístupů mezi sebou porovnat. K dispozici jsou údaje o procentu úspěšnosti (ko)koevoluce a také medián počtu generací, vyčíslení a doby nalezení řešení. Porovnání bude provedeno pouze v prvních třech uvedených kategoriích - experimenty byly prováděny na sestavách se značně rozdílným výpočetním výkonem. Dobu provádění algoritmu proto nemá smysl srovnávat. U údaje o počtu vyčíslení se pravděpodobně projeví jeden z rozdílů ve fungování obou algoritmů. U *GA predikce* po většinu doby vlákno pro evoluci prediktorů nečinně čeká. Pouze po obdržení signálu zasílaného v pravidelných intervalech vygeneruje novou generaci prediktorů. Poté opět přeruší svou činnost. Evoluci prediktorů je potřeba zpomalit, aby nedošlo k přeučení populace. Tento problém u *CGP predikce* nehrozí a evoluce prediktoru zde proto může probíhat nepřetržitě. Při ohodnocování jedinců z populace prediktorů je navíc potřeba provést počet vyčíslení odpovídající maximální velikosti predikované sady. Do naměřených výsledků se tyto skutečnosti mohou promítnout vyšším počtem vyčíslení u *CGP predikce*.

Výsledky bez koevoluce

Tabulka 5.4 zachycuje porovnání výsledků dosažených bez použití koevoluce. Ačkoli je cílem především srovnání rozdílných přístupů k evoluci prediktorů, tato tabulka alespoň pomůže nastínit, do jaké míry mají na výsledky vliv rozdíly v implementaci evoluce kandidátních řešení. Podmínky experimentů jsou u obou verzí totožné, v ideálním případě by se tak výsledky měly lišit jen minimálně.

Soudě podle dostupných informací, základní CGP algoritmus pro evoluci řešení je stejný. Oba přístupy ohodnocují jedince na základě skóre a délky kódovaného matematického vztahu. Nicméně rozdílný je například způsob práce s konstantami.CGP algoritmus v této diplomové práci ke konstantám přistupuje jako k dalším primárním vstupům. Naproti tomu u CGP algoritmu ze článku [9] jsou konstanty namapovány na uzly, jež mají konstantní nebo neplatný výstup. Dá se předpokládat, že určité rozdíly budou v mnoha dalších implementačních detailech a některé z nich mohou mít vliv na úspěšnost evoluce.

Po provedení srovnání je nejvýraznější rozdíl u úlohy F4, pro kterou CGP z této diplomové práce s výjimkou jediného případu nedokázala nalézt řešení. Přitom výsledky v článku [9] uvádějí 80% úspěšnost. Tento rozdíl je ještě více zarážející, vezme-li se v potaz, že úspěšnost pro zbylé úlohy je u obou variant takřka totožná. Bylo ověřeno nastavení koevoluce i použitá trénovací sada, ale příčinu se nalézt nepodařilo.

Výsledky bez použití koevoluce jsou srovnatelné u úloh F3 a F5. U dalších úloh jsou řádově srovnatelné. Rozdíly, které vznikly při porovnání CGP algoritmů mohou být způsobeny rozdíly v některých částech implementace.

	Verze	F1	F2	F3	F4	F5
Úspěšnost	GA prediktor	100%	100%	78%	80%	24%
	CGP prediktor	100%	100%	80%	1.6%	27%
Medián	GA prediktor	$1.11 \cdot 10^3$	$4.46 \cdot 10^3$	$1.76 \cdot 10^5$	$7.15 \cdot 10^5$	$1.36 \cdot 10^6$
počtu generací	CGP prediktor	$1.65 \cdot 10^3$	$1.95 \cdot 10^3$	$1.71 \cdot 10^5$	–	$1.76 \cdot 10^6$
Medián	GA prediktor	$2.68 \cdot 10^6$	$1.08 \cdot 10^7$	$4.24 \cdot 10^8$	$1.72 \cdot 10^9$	$3.28 \cdot 10^9$
počtu vyčíslení	CGP prediktor	$3.65 \cdot 10^6$	$4.32 \cdot 10^6$	$3.77 \cdot 10^8$	–	$3.90 \cdot 10^9$

Tabulka 5.4: Porovnání výsledků evoluce verze s GA prediktory oproti výsledkum verze s CGP prediktory bez použití koevoluce.

Výsledky s koevolucí

V tabulce 5.5 je uvedeno srovnání výsledků obou verzí za použití koevoluce. Z výsledků vyplývá, že *CGP predikce* při současném nastavení průměrně preferuje predikovanou sadu s větším počtem trénovacích vektorů. Nejvíce je to patrné u úloh F1 a F2. Ačkoliv *CGP predikce* dokázala nalézt řešení během menšího počtu generací, počet vyčíslení trénovacích vektorů byl vyšší. Obdobná byla situace u úlohy F3. K nalezení řešení stačila téměř třetina generací oproti *GA predikci*. Počet vyčíslení je u *CGP predikce* verze také nižší, ovšem jen minimálně. U funkcí F4 a F5 dokázala *GA predikce* nalézt lepší výsledky při menším počtu generací a výrazně méně vyčísleních.

CGP predikce navíc v několika případech u úlohy F5 nestihla ve stanoveném intervalu nalézt řešení, což se v případě *GA predikce* nestalo.

	Verze	F1	F2	F3	F4	F5
Úspěšnost	GA prediktor	100%	100%	100%	100%	100%
	CGP prediktor	100%	100%	100%	100%	90%
Medián	GA prediktor	$2.62 \cdot 10^3$	$2.53 \cdot 10^3$	$1.10 \cdot 10^5$	$1.10 \cdot 10^6$	$1.34 \cdot 10^6$
počtu generací	CGP prediktor	$1.00 \cdot 10^3$	$2.25 \cdot 10^3$	$4.11 \cdot 10^4$	$1.47 \cdot 10^6$	$1.74 \cdot 10^6$
Medián	GA prediktor	$5.20 \cdot 10^5$	$5.01 \cdot 10^5$	$2.19 \cdot 10^7$	$2.00 \cdot 10^8$	$2.67 \cdot 10^8$
počtu vyčíslení	CGP prediktor	$7.43 \cdot 10^5$	$1.60 \cdot 10^6$	$1.90 \cdot 10^7$	$8.05 \cdot 10^8$	$8.78 \cdot 10^8$

Tabulka 5.5: Porovnání výsledků koevoluce s GA prediktory oproti koevoluci s CGP prediktory s použitím koevoluce.

5.6 Shrnutí

V porovnání se standardním CGP algoritmem bez použití koevoluce dokázal koevoluční algoritmus založený na *CGP predikci* ve všech případech nalézt řešení v kratším čase, s výrazně menším počtem vyčíslení trénovacích vektorů a s vyšší úspěšností nalezení řešení požadované kvality. U čtyř z pěti úloh bylo pozorováno, že se prediktor dokázal určitým způsobem adaptovat na zkoumanou úlohu. Pro úlohu F2 se prediktory dokázaly částečně adaptovat na lokální extrémy hledané funkce. Naproti tomu u F3 byly predikované trénovací vektory vybírány rovnoměrně v celé trénovací sadě. Pro úlohy F1 a F2 je známo odlišné nastavení evoluce *CGP prediktorů*, při kterém je dosahováno lepších výsledků. U úloh, kde bývá řešení nalezeno během několika desítek tisíc generací, dojde ke zlepšení výsledků při kratších intervalech výměny prediktoru a při zvýšení preference kratších, ale méně přesných prediktorů. Avšak toto nastavení není vhodné pro složitější úlohy.

Výsledky dosažené pomocí koevoluce by tak pravděpodobně bylo možné zlepšit, pokud by tyto parametry nebyly nastaveny jako neměnné, ale mohly se v průběhu koevoluce měnit. Stačilo by, aby v úvodní fázi algoritmu byly intervaly výměny prediktoru výrazně kratší a postupně by se zvyšovaly až na stanovenou konečnou úroveň. Obdobně by bylo možné upravovat preference přesných prediktorů. Takto by se pravděpodobně urychlilo nalezení řešení u úloh F1 a F2. U úloh, kde nalezení řešení trvá podstatně déle, by tato změna neměla mít žádný zásadní vliv.

Kapitola 6

Závěr

Hlavním cílem této práce byl návrh a implementace koevolučního algoritmu u úloh založených na testu, který dokáže řešit úlohy symbolické regrese. Výsledkem je aplikace schopná nalézt matematický vztah popisující zadanou sadu vstupních hodnot. Byla použita varianta koevolučního algoritmu, při které jsou testy vyvíjeny novým a dosud nevyzkoušeným způsobem - pomocí evolučního mechanismu na principu symbolické regrese. Oproti dřívějším přístupům umožňuje tato metoda dynamicky měnit velikost trénovací sady a přizpůsobit ji zadané úloze.

Algoritmus byl testován na sadě vybraných úloh. Použití koevoluce vedlo vždy k rychlejšímu nalezení řešení, ve většině případů bylo zrychlení dosti výrazné. Také se zvýšila úspěšnost symbolické regrese. Velikost sady testů u některých úloh konvergovala k optimální hodnotě, v jiných případech se výrazně měnila po celou dobu běhu symbolické regrese.

Bylo provedeno srovnání s výsledky dosaženými pomocí obdobného algoritmu se statickou velikostí sady testů. Nová metoda dokázala u tří z pěti zkoumaných úloh nalézt řešení během menšího počtu generací. Ve většině případů bylo ale potřeba provést více vyčíslení trénovacích vektorů. Vyšší počet vyčíslení zčásti plyne z rozdílu principů srovnávaných přístupů, zejména v rámci výpočtu fitness prediktorů. U koevolučního algoritmu popsaného v této diplomové práci probíhá evoluce prediktorů bez přerušení po celou dobu koevoluce. Provede se tak i vyšší počet vyčíslení trénovacích vektorů.

Úspěšnost koevoluce a rychlost nalezení řešení závisí na hodnotách několika parametrů, jejichž co nejvýhodnější nastavení se ukázalo být rozdílné pro každou z úloh. Při experimentech bylo použito nastavení, při kterém byly naměřeny lepší výsledky u déle trvajících běhů. Následkem toho se ale zhoršily výsledky u jednodušších úloh. V textu je navrženo případné rozšíření, které by mohlo tento problém výrazně zredukovat. Charakter chování evoluce prediktorů se do značné míry odvíjí od způsobu výběru nejlepšího prediktoru. Významný vliv má také princip obměny obsahu archivu trenérů. V rámci dalšího pokračování vývoje by bylo možné například otestovat variantu, kde do archivu trenérů nejsou vkládáni náhodní jedinci, ale pouze kandidátní řešení. Na jejich genotypu by byl proveden náhodný počet mutací pro zajištění dostatečné rozmanitosti obsahu archivu.

Během práce na projektu jsem si zopakoval principy, na kterých pracuje množství evolučních algoritmů a nastudoval si problematiku koevoluce a symbolické regrese. Tento projekt měl pro mne přínos i v oblastech, které s jeho primárním zaměřením souvisejí jen velmi okrajově - jedná se o problematiku paralelizace algoritmů a tvorbu uživatelských rozhraní.

Tato práce byla zařazena do sborníku soutěže STUDENT EEICT 2014.

Literatura

[1] Clegg, J.; Walker, J. A.; Miller, J. F.: A new crossover technique for cartesian genetic programming. *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, 2007, ISBN 978-1-59593-697-4, s. 1580–1587.

[2] Jaśkowski, W.; Krawiec, K.; Wieloch, B.: Fitnessless coevolution. *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ACM, 2008, ISBN 978-1-60558-130-9, s. 355–362.

[3] Nordin, P.; Francone, F.; Banzhaf, W.: Explicitly Defined Introns and Destructive Crossover in Genetic Programming. 1995.

[4] Poli, R.; McPhee, N. F.: Parsimony pressure made easy. *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ACM, 2008, ISBN 978-1-60558-130-9, s. 1267–1274.

[5] Popovici, E.; Bucci, A.; Wiegand, R. P.; aj.: Coevolutionary principles. *Handbook of Natural Computing*, Springer, 2012, ISBN 978-3-540-92909-3, s. 987–1033.

[6] 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.

[7] Sekanina, L.: Evolvable hardware. *Evolvable Components*, Springer, 2004, ISBN 978-3-540-92909-3, s. 42–66.

[8] Sikulova, M.; Sekanina, L.: Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP. *Lecture Notes in Computer Science*, ročník 2012, č. 7491, 2012: s. 163–172, ISSN 0302-9743.

[9] Sikulova, M.; Sekanina, L.: Coevolution in Cartesian Genetic Programming. *Proc. of the 15th European Conference on Genetic Programming*, LNCS 7244, Springer Verlag, 2012, ISBN 978-3-642-29138-8, s. 182–193.

[10] Zelinka, I.; Oplatkova, Z.; Nolle, L.: Analytic programming-symbolic regression by means of arbitrary evolutionary algorithms. *Journal of Simulation*, ročník 6, č. 9, 2005: s. 44–56, ISSN 1473-8031.

Příloha A

Obsah CD

Soubory na CD jsou uloženy v následující adresářové struktuře:

- `Aplikace` – adresář s již přeloženou aplikací v obou variantách,
  - `Console` – spustitelný soubor s konzolovou verzí aplikace pro Windows XP a vyšší, dávkové soubory pro provedení experimentů,
    - `testy` – soubory s trénovacími sadami a nastavením,
  - `GUI` – spustitelný soubor verze s grafickým uživatelským rozhraním pro Windows XP a vyšší, soubory s nastavením a trénovacími sadami,
- `Dokumentace` – adresář obsahující PDF soubor s technickou zprávou,
  - `Ovladani.pdf` – popis překladu a ovládání programu (tedy obsah příloh B a C),
  - `Tridy.pdf` – popis jednotlivých tříd ve zdrojovém kódu,
  - `XMLnastaveni.pdf` – popis struktury souboru s nastavením koevoluce,
- `Technická zpráva` – adresář obsahující PDF soubor s technickou zprávou,
  - `LaTeX` – zde jsou uloženy veškeré zdrojové materiály pro sestavení technické zprávy,
- `Zdrojové kódy` – adresář se zdrojovými soubory pro obě verze,
  - `Console` – zdrojové soubory pro konzolovou verzi aplikace, Makefile a také shellové skripty pro provedení experimentů,
    - `testy` – soubory s nastavením a trénovacími sadami,
  - `GUI` – zdrojové soubory pro verzi aplikace grafickým uživatelským rozhraním,
    - `testy` – soubory s trénovacími sadami a nastavením,

Některé soubory jsou na CD umístěny vícekrát v rozdílných adresářích. Například obě verze aplikace sdílí zdrojové kódy popisující jádro algoritmu. Soubory s trénovací sadou a s nastavením jsou umístěny v adresáři se zdrojovými kódy i ve složce s již přeloženými aplikacemi. Je tomu tak proto, aby jednotlivé adresáře na sobě nebyly navzájem závislé. Uživatel tak nemusí složitě pátrat po souborech, které potřebuje – vše je k dispozici ve vybrané podsložce.

Příloha B

Konzolová verze programu

Tato verze aplikace pracuje čistě v terminálu.

B.1 Překlad programu

Překlad je možné provést s použitím přiloženého `Makefile`. V kódu jsou využity některé prvky standardu `C++11`. Na školních počítačích byl překlad úspěšně testován na studentském serveru `Merlin`. Na serveru `Eva` byla v době testování nainstalována starší verze překladače `gcc`, která zmíněný standard dosud plně nepodporovala, a překlad se nezdařil.

B.2 Ovládání

Chování programu je možné modifikovat pomocí velkého počtu argumentů. Všechny je shrnuje tabulka B.1. Zde budou zdůrazněny ty nejdůležitější z nich.

- Pro spuštění programu je povinný pouze parametr `-t filename`, kde `filename` je název souboru s trénovací sadou, tedy například:

  ```
  ./coea -t f1.train
  ```

  V takovém případě program provede jeden koevoluční běh a skončí. V průběhu koevoluce průběžně vypisuje dosud nejlépe ohodnocená kandidátní řešení. V pravidelných intervalech také zobrazuje aktuální velikost a rozložení predikované sady. Nakonec vypíše nalezené řešení společně s potřebným počtem vyčíslení. Pokud není řešení nalezeno, vypíše alespoň nejlepšího dosud nalezeného jedince.

- Parametrem `-repeat #` lze nastavit vícenásobně opakování experimentu, kde # je číslo udávající počet opakování. V takovém případě aplikace na závěr vypíše statistické údaje udané v kvartilech o počtu generací, počtu vyčíslení datových bodů a době provádění jednotlivých experimentů.

- Načtení souboru s nastavením koevoluce se provede pomocí parametru `-s filename`, kde `filename` je název xml souboru s nastavením.

Pro popis dalších parametrů již čtenáře odkáži na zmíněnou tabulku B.1.

Stojí za to připomenout, že na CD jsou k dispozici soubory se shellovými skripty (respektive dávkové soubory pro Windows verzi aplikace), které aplikaci spustí se stejnými

parametry, jaké byly použity u testovacích úloh F1 až F5 popsaných v kapitole 5. Jedná se o soubory `F1.sh` až `F5.sh` (a jejich protějšky pro Windows `F1.bat` až `F5.bat`). Tyto soubory jsou přítomny i ve variantě s přídomkem `nocoea`, čímž se vypne použití predikce fitness – je tedy použit pouze základní CGP algoritmus bez koevoluce. Nalezené výsledky a statistické údaje včetně histogramů prediktorů budou zapsány do souboru `out.txt`.

Příkaz	Výchozí	Popis
`-t filename`		Načte soubor `filename` s trénovaci sadou. Povinné.
`-o filename`		Do souboru `filename`, bude zapsán výstup programu.
`-s filename`		Načte soubor `filename`, s nastavením koevoluce.
`-genxml filename`		Vygeneruje soubor s výchozím nastavením koevoluce.
`-repeat n`	1	Zopakuje experiment n-krát.
`-tolerance n`	0,2	Nastaví toleranci přesnosti řešení na hodnotu n.
`-nocoea`		Nebude použita predikce. Běh programu bez koevoluce.
`-maxrounds n`	8000000	Nastaví maximální počet generací.
		Při záporné hodnotě nebude podmínka použita.
`-maxsecs n`	-1	Maximální doba (ko)evoluce v sekundách.
		Při záporné hodnotě nebude podmínka použita.
`-maxmins n`	-1	Maximální doba (ko)evoluce v minutách.
		Při záporné hodnotě nebude podmínka použita.
`-threads n`	0	Nastaví počet vláken. Kladná hodnota n značí počet vláken.
		Při záporné nebo nulové hodnotě platí, že:
		počet vláken = max.počet vláken - n
		pro koevoluci musí být $n \geq 2$, pro CGP $n >= 1$
`-predhist`		Na závěr se vypíší histogramy velikosti a rozložení prediktoru.
`-h --help`		Program vypíše nápovědu a skončí.

Tabulka B.1: Tabulka parametrů konzolové verze aplikace.

Příloha C

Verze programu s grafickým uživatelským rozhraním

Pro usnadnění konfigurace parametrů koevoluce a také sledování jejího průběhu a výsledků byla pro aplikaci vytvořena nadstavba v podobě grafického uživatelského rozhraní. Již dopředu je třeba předeslat, že byla vyvíjena především pro účely demonstrace funkčnosti samotného algoritmu. Pro širší použití by bylo vhodné doplnit další funkcionalitu pro vyšší uživatelský komfort. Například se jedná o lepší ošetření chybných vstupů nebo intuitivnější ovládání některých prvků.

C.1 Překlad programu

Pro přeložení této verze je potřeba mít nainstalováno `Qt SDK` verze 5.2 a vyšší. Překlad se provede pomocí `qmake` a přiloženého projektového souboru. Překlad této verze nebyl na školních počítačích testován. Nicméně na operačních systémech Windows lze použít již přeložený program umístěný v adresáři `Aplikace/GUI`.

C.2 Ovládání

Rozhraní aplikace je rozděleno do čtyř záložek.

Trénovací sada: Tabulka sloužící k načtení a úpravám použité trénovací sady.

Nastavení (ko)evoluce: Zde je možné měnit nastavení průběhu koevoluce.

Spustit: Spuštění koevoluce a sledování jejího průběhu.

Statistiky: Zobrazení souhrnných výsledků z opakovaných běhů koevoluce.

Při spuštění aplikace by se měla automaticky načíst ukázková trénovací sada a výchozí nastavení. Je proto možné se rovnou přepnout do záložky *Spustit* a koevoluci spustit.

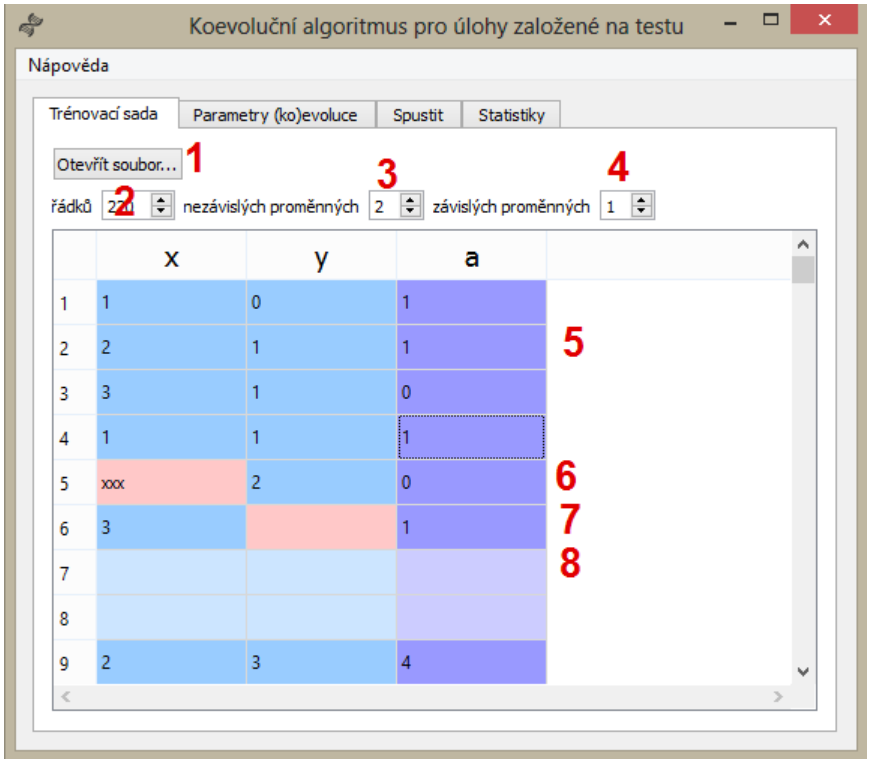
V případě, že chce uživatel zopakovat některý z experimentů uvedených v technické zprávě, musí postupovat následovně: Nejprve načte soubor s trénovací sadou ve stejnojmenné záložce. Poté načte soubor s nastavením v v záložce *Nastavení (ko)evoluce*. Nakonec tamtéž zvolí pomocí políčka *Predikovat fitness*, zda testovat variantu s koevolucí, či jen standardní CGP a experiment spusti v záložce *Spustit*.

Následuje popis prvků rozhraní v jednotlivých záložkách.

C.2.1 Záložka *Trénovací sada*

Zde je možné načíst soubor s trénovací sadou, případně trénovací sadu přímo vytvořit či upravit. Každý řádek tabulky představuje jeden trénovací vektor. Světlejší sloupce představují nezávislé (vstupní) proměnné a ve tmavších sloupcích jsou zobrazeny hodnoty závislých (výstupních) proměnných. Červená barva buňky značí chybu - chybí zde hodnota, nebo obsah buňky není možné převést na číselnou hodnotu. Prázdné řádky, nebo řádky s chybnými buňkami jsou při sestavování trénovací sady ignorovány.

Jednotlivé prvky v záložce *Trénovací sada* jsou označeny na obrázku C.1.



Obrázek C.1: Záložka *Trénovací sada*.

1 Otevře dialogové okno pro načtení trénovací sady ze souboru.

2 Políčko pro úpravu počtu řádků trénovací sady.

3 Políčko pro úpravu počtu nezávislých proměnných (světlé sloupce).

4 Políčko pro úpravu počtu závislých proměnných (tmavé sloupce).

5 Řádky 1 až 4 a 9 obsahují platné trénovací vektory.

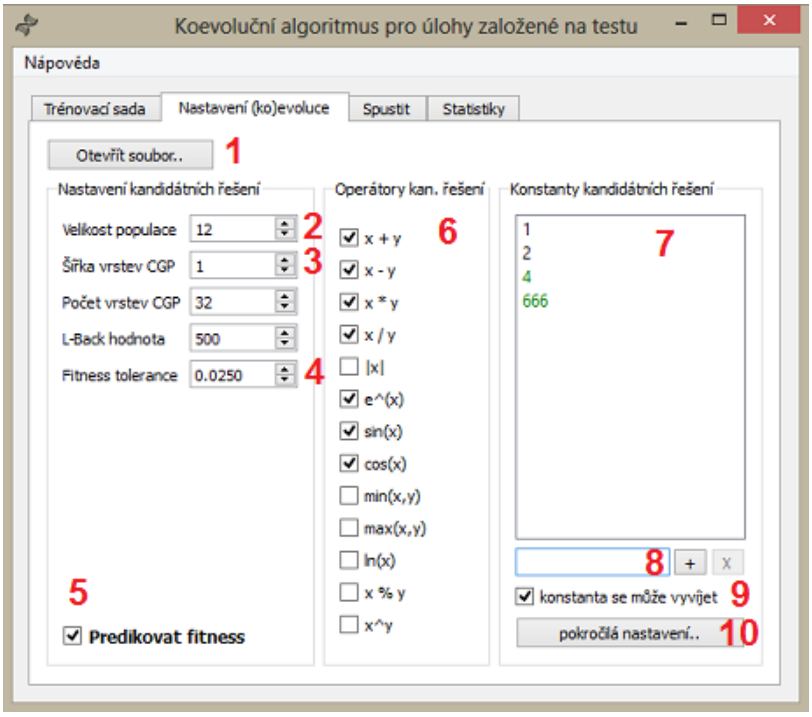
6 Řádek 5 obsahuje v 1. sloupci neplatnou hodnotu. Bude ignorován.

7 Na řádku 6 ve druhém sloupci chybí hodnota. Bude ignorován.

8 Řádky 7 a 8 jsou prázdné. Budou ignorovány.

C.2.2 Záložka *Nastavení (ko)evoluce*

Zde je možné měnit parametry koevoluce. V této záložce jsou umístěna nastavení týkající se evoluce kandidátních řešení. Jedná se o parametry, které je zpravidla vhodné upravovat v závislosti na zadané trénovací sadě. Nastavení evoluce prediktorů a komunikace mezi populacemi lze nastavit až v okně *Pokročilá nastavení*, popsaném v části C.2.3. Nastavení jsou při spuštění načtena ze souboru `default.xml`. Pokud tento soubor chybí, je při ukončení aplikace vytvořen. Jednotlivé prvky v záložce *Nastavení (ko)evoluce* jsou označeny na obrázku C.2.



Obrázek C.2: Záložka *Nastavení (ko)evoluce*.

1 Dialogové okno pro načtení souboru s nastavením.

2 Velikost populace kandidátních řešení.

3 Nastavení CGP mřížky uzlů jedinců z populace kandidátních řešení.

4 Hodnota tolerance při výpočtu fitness jedince jedince pomocí skóre.

5 Je-li políčko zatrženo, bude použita koevoluce, v opačném případě pouze CGP algoritmus.

6 Výběr operátorů, které se mohou stát součástí fenotypu kandidátních řešení.

7 Seznam použitých konstant. Zeleně označené konstanty se mohou vyvíjet. Černě označené si zachovávají svou hodnotu po celou dobu (ko)evoluce.

8 Pole pro zadávání nových konstant.

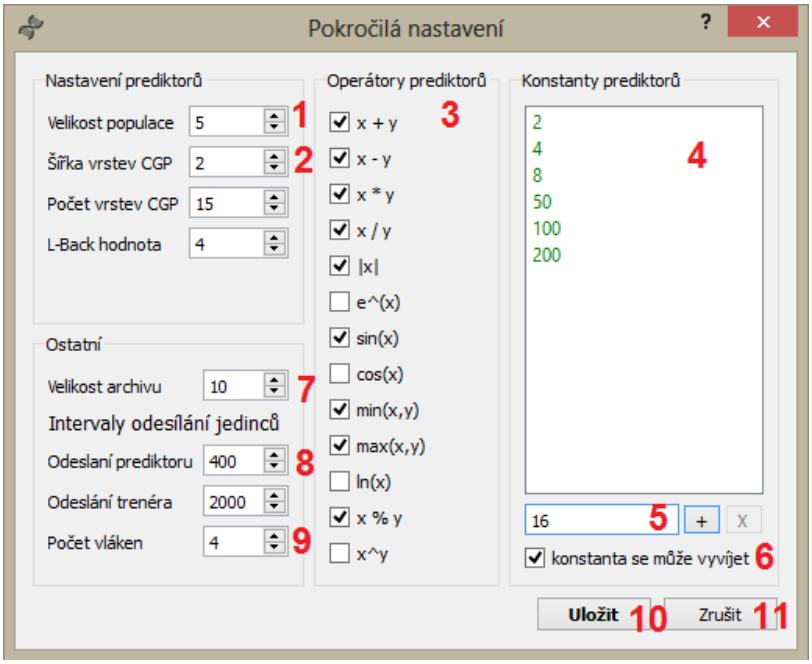
9 Zatrhávácí políčko které udává, zda se nová konstanta může vyvíjet.

10 Otevře okno *Pokročilá nastavení*.

C.2.3 Okno *Pokročilá nastavení*

Toto okno je možné otevřít stisknutím tlačítka `pokročilá nastavení..` v záložce *Nastavení (ko)evoluce*. Parametrům evoluce prediktorů byla během experimentů přiřazena konstantní, co nejvýhodnější hodnota. Zde je ovšem možné s většinou těchto parametrů experimentovat a měnit jejich hodnoty.

Jednotlivé prvky v okně *Pokročilá nastavení* jsou označeny na obrázku C.3.



Obrázek C.3: Okno *Pokročilá nastavení*.

1 Velikost populace prediktorů.

2 Nastavení CGP mřížky uzlů jedinců z populace prediktorů.

3 Výběr operátorů, které se mohou stát součástí fenotypu prediktorů.

4 Seznam použitých konstant. Zeleně označené konstanty se mohou vyvíjet. Černě označené si zachovávají svou hodnotu po celou dobu (ko)evoluce.

5 Pole pro zadávání nových konstant.

6 Zatrhávácí políčko které udává, zda se nová konstanta může vyvíjet.

7 Kapacita archivu trenérů.

8 Intervaly zasílání jedinců mezi oběma vlákny.

9 Celkový počet vláken, který bude využit při (ko)evolučním algoritmem.

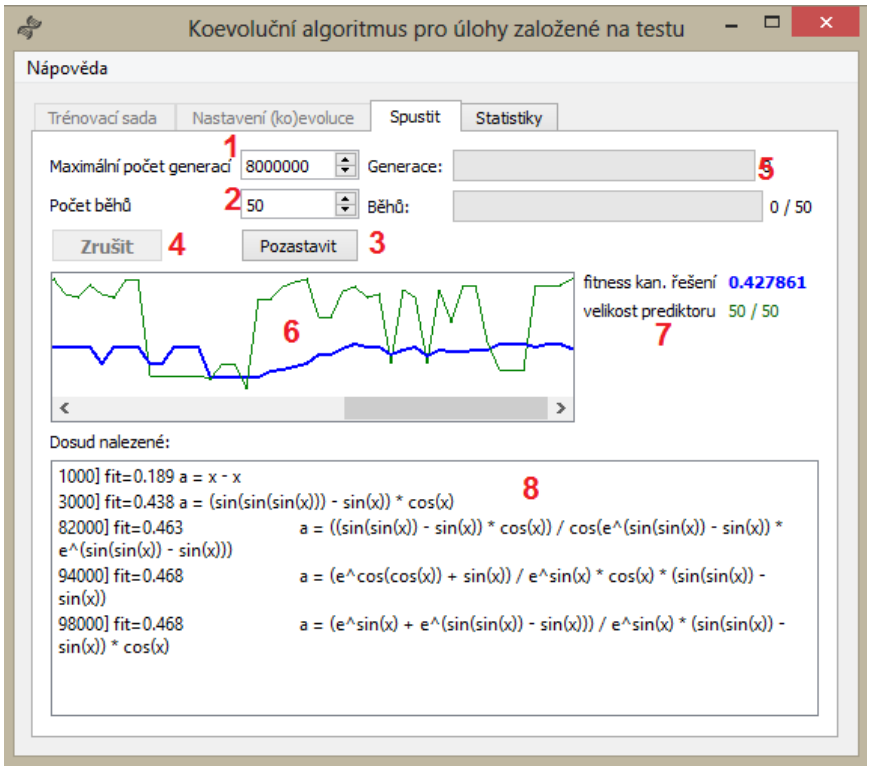
10 Uloží provedené změny

11 Zavře okno bez uložení změn

C.2.4 Záložka *Spustit*

V průběhu koevoluce není možné měnit nastavení – záložky *Trénovací sada* a *Nastavení koevoluce* jsou znepřístupněny.

Jednotlivé prvky v záložce *Spustit* jsou označeny na obrázku C.4.



Obrázek C.4: Záložka *Spustit*.

1 Maximální počet generací evoluce kandidátních generací. Po jeho dosažení bude evoluce ukončena s tím, že řešení nebylo nalezeno.

2 Počet opakování běhu (ko)evoluce. Statistické údaje jsou shromažďovány napříč všemi provedenými opakováními.

3 Spuštění / pozastavení (ko)evoluce.

4 Zrušení (ko)evoluce. (Ko)evoluci lze zrušit pouze v okamžiku, kdy je pozastavena.

5 Ukazatele průběhu.

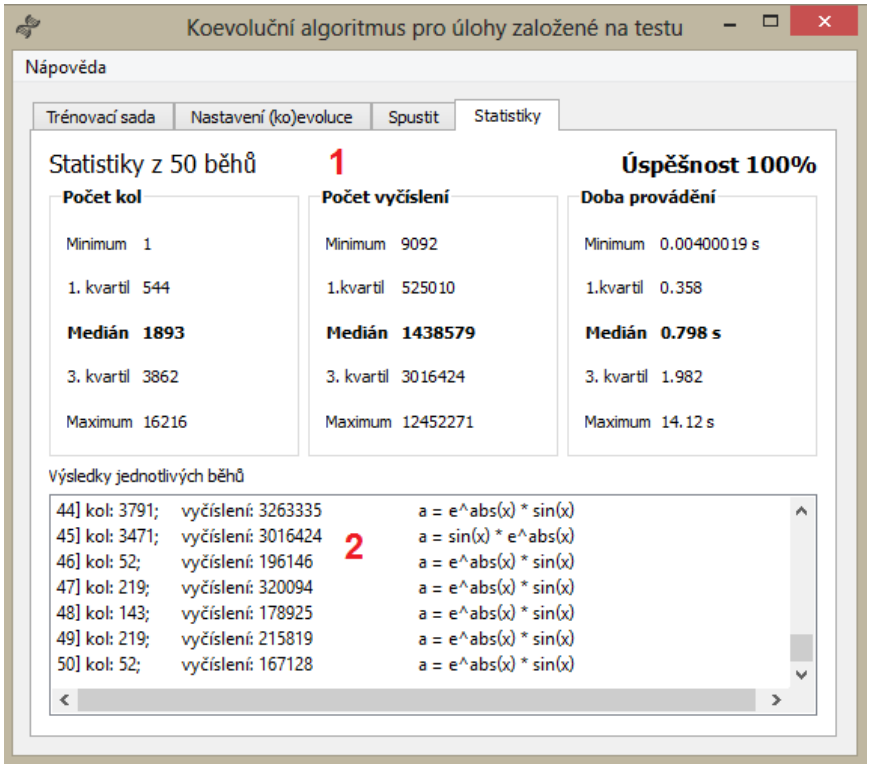
6 Panel zachycující změny fitness kandidátních řešení (modře) a velikosti predikované sady (zeleně).

7 Hodnota aktuální fitness nejlépe ohodnoceného kandidátního řešení a současná velikost predikované sady.

8 Dosud nalezená nejlépe ohodnocená kandidátní řešení v aktuálním běhu.

C.2.5 Záložka *Statistiky*

Tato záložka obsahuje souhrn statistických dat naměřených napříč všemi provedenými běhy (ko)evoluce. Vyjma informace o úspěšnosti jsou všechny zobrazené údaje stanoveny pouze na základě běhů, kde bylo úspěšně nalezeno řešení. Jednotlivé prvky v záložce *Statistiky* jsou označeny na obrázku C.5.



Obrázek C.5: Záložka *Statistiky*.

1 Statistické údaje o úspěšnosti, počtu generací, počtu vyčíslení trénovacích vektorů a době provádění koevoluce.

2 Zde jsou vypsány výsledky jednotlivých běhů - dosažené hodnoty a nalezený vztah.