VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MUTACE V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

MUTATION IN CARTESIAN GENETIC PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ KONČAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL WIGLASZ
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Končal Ondřej**
Obor: Informační technologie
Téma: **Mutace v kartézském genetickém programování**
Mutation in Cartesian Genetic Programming
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s principy evolučních algoritmů, kartézského genetického programování a symbolické regrese.
2. Navrhněte program umožňující řešit úlohy typu symbolická regrese pomocí kartézského genetického programování, které využívá různé strategie mutace jedinců.
3. Navržené řešení implementujte.
4. Ověřte funkčnost programu na zadaných úlohách a porovnejte jednotlivé strategie mutace jedinců.
5. Shrňte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Wiglasz Michal, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

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

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato práce se zabývá zkoumáním různých druhů mutací v kartézském genetickém programování (CGP) na úlohách symbolické regrese. CGP je druh evolučního algoritmu, který pracuje se spustitelnými strukturami. Mutace je u CGP hlavním genetickým operátorem a v kombinaci s ohodnocením zabírá nejdelší dobu běhu algoritmu. Nalezení lepšího druhu mutace proto může výrazně zrychlit tvorbu nových jedinců, a tak i zkrátit dobu běhu algoritmu. Tato práce představuje čtyři druhy mutací používané v CGP. Experimenty porovnávají tyto mutační operátory při řešení pěti úloh symbolické regrese. Ukazuje se, že vhodnou volbou mutace lze dosáhnout až skoro dvojnásobného zrychlení oproti standardnímu mutačnímu operátoru.

Abstract

This thesis examines various kinds of mutations in the Cartesian Genetic Programming (CGP) on tasks of symbolic regression. The CGP is kind of evolutionary algorithm which operates with executable structures. Programs in CGP are evolved using mutation, which leads to offspring evaluation, which is the most time–consuming part of the algorithm. Finding more suitable kind of mutation can significantly accelerate the creation of new individuals and thus, reduce the time necessary to find a satisfactory solution. This thesis presents four different mutations for CGP. Experiments compare these mutation operators to solve five tasks of symbolic regression. Experiments have shown that a choice of suitable mutation can almost double the computing speed in comparison to the standard mutation.

Klíčová slova

Symbolická regrese, evoluční algoritmus, kartézské genetické programování, mutace.

Keywords

Symbolic regression, evolutionary algorithm, cartesian genetic programming, mutation.

Citace

KONČAL, Ondřej. *Mutace v kartézském genetickém programování.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Michal Wiglasz.

Mutace v kartézském genetickém programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Wiglasze.

. . . . . .
Ondřej Končal
16. května 2016

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. Michalu Wiglaszovi za odborné vedení, vstřícnost při konzultacích a cenné rady, které mi pomohly tuto práci vyhotovit.

Obsah

Kapitola 1

Úvod

I se zvětšujícím se výkonem počítačů trvají některé výpočty stále příliš dlouho. Jednou ze skupin úloh náročných na výpočetní čas jsou úlohy založené na prohledávání stavového prostoru. Pro tyto úlohy vykazují dobré výsledky algoritmy inspirované přírodou, jakými jsou kolektivní inteligence nebo evoluční algoritmy, které vychází z myšlenek Charlese Darwina a z genetiky.

Evoluční algoritmy se obvykle používaly pro optimalizaci již existujících systémů, ale ukázalo se, že genetické programování (GP) dokáže systémy přímo navrhovat. Výpočetně nejnáročnější částí GP je tvorba potomků. Tato práce se zabývá srovnáním různých druhů mutačních operátorů používaných v kartézském genetickém programování (CGP). Volbou vhodného operátoru mutace lze snížit počet ohodnocení potřebných k nalezení řešení. Navržené mutace jsou implementovány a jejich vliv na celkový počet evaluací je experimentálně vyhodnocen na pěti úlohách symbolické regrese.

Tato bakalářská práce je strukturována následovně: V kapitole 2 jsou představeny evoluční algoritmy, včetně GP a CGP, kterými se tato práce zabývá. Kapitola 3 popisuje návrh programu řešící symbolickou regresi za pomoci kartézského genetického programování se čtyřmi různými mutačními operátory. Kapitola 4 přibližuje implementaci programu a obsahuje podrobný popis ovládání programu. Návrh a výsledky experimentů jsou popsány v kapitole 5. Závěr (kapitola 6) shrnuje výsledky a poznatky získané v této práci.

Kapitola 2

Evoluční algoritmy

Pojem evoluční algoritmy zahrnuje stochastické prohledávací algoritmy, které nachází inspiraci v Darwinově evoluční teorii, teoriích neodarwinismu a genetice. Základem je myšlenka, že rozmanité druhy v přírodě vznikají postupným vývojem tak, jak se adaptují na prostředí, ve kterém žijí. Čím úspěšnější jedinec, tím větší pravděpodobnost, že zplodí potomky a jeho gen přežije. Takto inspirované algoritmy vznikaly nezávisle na sobě už od poloviny 20. století. V 90. letech byly zahrnuty pod termín evoluční algoritmy (EA).

Hlavním rysem EA je existence multimnožiny kandidátních řešení, nad kterou jsou prováděny operace a vyhodnocování. Používají se především pro úlohy s velkým prohledávaným prostorem nebo pro úlohy, jejichž parametry se mění za běhu programu.

Původně se používaly především pro nalezení hodnot optimalizovatelného systému. V poslední době se začaly využívat i pro kompletní návrh systémů a mohou dosáhnout lepších výsledků než návrhy vytvořené člověkem [8].

2.1 Základní pojmy

V práci se nachází řada pojmů z oboru EA, které je potřeba si vymezit. Většina pojmů má podklad v biologii a snaží se korespondovat s biologickými termíny. Pojmy jsou definovány podle [3, 8]:

Gen – v molekulární biologii se tak označuje základní jednotka genetické informace, která je tvořena sekvencí nukleotidů. V oblasti evolučních algoritmů jsou geny nejčastěji reprezentovány binárními, celými nebo reálnými čísly, které mohou kódovat různé struktury (logický člen, matematická funkce).

Chromozom – pole lineárně uspořádaných genů (obr. 2.1).

1	0	0	1	0	0	1	0

Obrázek 2.1: Chromozom

Jedinec – kandidátní řešení představované jedním nebo více chromozomy.

Genotyp – zakódovaný jedinec tvořený posloupností genů. V případě existence pouze jednoho chromozomu jsou pojmy genotyp a chromozom zaměnitelné.

Fenotyp – dekódovaný genotyp, který označuje projevy jedince. Fenotyp může vzniknout přímou interpretací jednotlivých genů, anebo složitějším algoritmem.

Populace – multimnožina kandidátních řešení vstupujících do evolučního algoritmu. Velikost populace může být proměnná. Populace mohou být:

Generační – populace je tvořena vždy jen jednou, novou generací. Je to ekvivalent jednoletých rostlin.

Překrývající se – část populace je nahrazena novou generací. Dochází tak k překrývání populací jako u zvířat.

Fitness – číselné ohodnocení kvality jedince. Obecně platí, že větší hodnota fitness značí lepší vlastnosti a jedinec tak má větší šanci, že jeho gen bude předán potomkům. Existují různá vyjádření hodnoty fitness:

Hrubá fitness – hodnota z množiny hodnot přirozených problémové doméně.

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

Přizpůsobená hodnota fitness – výpočet je proveden pomocí vzorce

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

Výsledné hodnoty se budou nacházet v intervalu $(0,\ 1\rangle$, kde hodnota 1 je tou nejlepší.

Normalizovaná hodnota fitness – vypočítá se jako

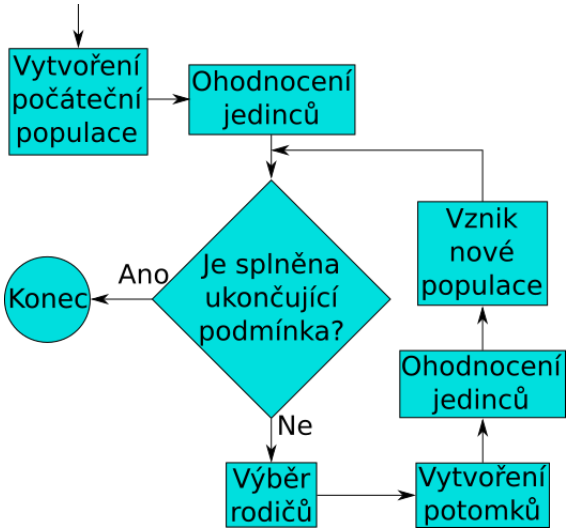
$$\frac{\text{hrubá hodnota fitness}}{\sum_{i=0}^{N-1} \text{hrubá hodnota fitness i-tého jedince}} \tag{2.2}$$

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

Fitness funkce – funkce, která jedinci přiřazuje hodnotu fitness.

2.2 Princip evolučního algoritmu

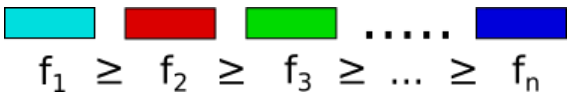
Algoritmus je znázorněn na obrázku 2.2. Prvním krokem je vytvoření počáteční populace. Má předem určenou velikost a může být vytvořena náhodně, nebo za pomoci již známých vědomostí o problému, které přiblíží algoritmus k řešení. Po vytvoření je populace ohodnocena za pomoci fitness funkce. Dokud nejsou splněny ukončující podmínky, kterými jsou nalezení dostatečně kvalitního jedince nebo dosažení určeného počtu generací, vyberou se z aktuální populace rodiče pomocí některého z řady selekčních algoritmů:



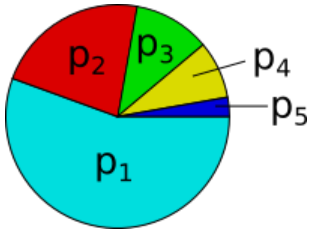
Obrázek 2.2: Schéma evolučního algoritmu

Deterministická selekce – vybrán příslušný počet jedinců s nejvyšší hodnotou fitness (obr. 2.3).

Ruletová selekce – mezi jedince jsou rozděleny úseky pomyslné rulety, kde velikost úseku zabraná každým jedincem je přímo úměrná jeho hodnotě fitness (obr. 2.4). Suma velikostí všech úseků je rovna jedné. Vygenerováním čísla z intervalu $\langle 0, 1\rangle$ je vybrán úsek na ruletě, a tím i jedinec, kterému úsek patří. Při výběru více rodičů je generování opakováno (jedinec může být jako rodič vybrán vícekrát). U kvalitnějšího jedince je tak větší pravděpodobnost, že se stane rodičem, ale zároveň méně kvalitní jedinec o tuto šanci úplně nepřichází.



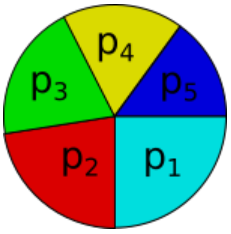
Obrázek 2.3: Deterministická selekce



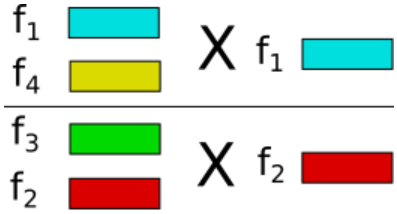
Obrázek 2.4: Ruletová selekce

Podle pořadí – jedinci jsou seřazeni podle hodnoty fitness a pravděpodobnost výběru jedince je přímo úměrná jeho pořadí (obr. 2.5). Tato metoda řeší problém degenerace u ruletové selekce, kdy jeden jedinec měl výrazně větší hodnotu fitness než ostatní.

Turnajová selekce – v rámci populace je náhodně vybráno s jedinců (obvykle se s nachází mezi 2 až 8), ze kterých se ten s největší hodnotou fitness stane rodičem (obr. 2.6). Proces je opakován tolikrát, kolik rodičů je potřeba vybrat.



Obrázek 2.5: Podle pořadí



Obrázek 2.6: Turnajová selekce

Obecně platí pravidlo, že čím lepší je hodnota fitness, tím pravděpodobněji bude jedinec vybrán za rodiče. Za pomoci genetických operátorů vznikají z rodičů potomci. Selekce a genetické operace probíhají s určitou pravděpodobností, kterou volí uživatel. Když jsou nově vzniklí jedinci ohodnoceni fitness funkcí, vytvoří se nová populace s určitou částí rodičů a potomků. Poměr rodičů a potomků odpovídá zvolenému typu populace. Nová populace je opět otestována na ukončující podmínky. Algoritmus končí, pokud je nalezen dostatečně kvalitní výsledek, nebo byl dosažen maximální počet generací, který je určen uživatelem na počátku algoritmu [8].

2.3 Varianty evolučních algoritmů

V průběhu vývoje a rozšiřování evolučních algoritmů začaly vznikat různé varianty algoritmu kvůli přizpůsobení se požadavkům rozličných úloh, na které byly aplikovány. V knize [1] jsou popsány následovně:

Genetický algoritmus – řešení je v chromozomu zakódováno většinou binárně. Pokud by to úloha vyžadovala, chromozom nemusí mít konstantní délku. Genetický algoritmus využívá jako operátory mutaci a křížení. Mutace se provádí v binárním zakódování inverzí bitů. Ze způsobů křížení dvou chromozomů se nejčastěji používá jednobodové a vícebodové křížení, ale existují i jiné, například prohození genů na náhodně vybraných pozicích. Speciální typy operátorů se vyskytují u úloh s pemutacemi, například problém obchodního cestujícího. Mutace znamená prohození dvou měst. Křížení je provedeno odebráním a přeskupením měst tak, aby i potomek obsahoval každé město právě jednou. Selekce je obvykle prováděna ruletovou selekcí.

Evoluční strategie – využívá se například pro optimalizaci parametrů, kdy je cílem najít nejlepší vektor. Pracuje přímo s vektorem reálných čísel. Základní operací je zde mutace, která je realizována jako přičtení malého náhodného čísla, generovaného na základě Gaussova rozdělení, ke každé z dimenzí vektoru. Později vzniklo i několik variant operace křížení. Kromě jednobodového a vícebodového křížení lze potomka vytvořit jako aritmetický průměr hodnot rodičů. Nová populace je tvořena jedním ze dvou typů selekce. První vybírá μ nejlepších jedinců z μ rodičů a λ potomků, nazývá se $(\mu + \lambda)$. Druhá vybere μ nejlepších jedinců z λ potomků, kteří vytvoří novou generaci. Je označována (μ, λ).

Evoluční programování – podobné evoluční strategii. Jeho cílem je simulovat inteligenci, a to obvykle za pomoci konečného automatu, který predikuje výstup na základě vstupního řetězce. Hodnota fitness udává celkovou chybu tvořenou součtem odchylek jedince od hle-

daného řešení po přechodu do nového stavu. Při tvorbě potomka je využita některá z pěti mutací: změna symbolu na výstupu, změna hrany, přidání stavu, odebrání stavu, a nebo změna počátečního stavu. Z každého rodiče jsou vytvořeni dva potomci a následně je ze všech potomků vybrána lepší polovina, která se stane rodiči v další generaci.

Genetické programování – pracuje se spustitelnými strukturami jakými jsou stromy. Používá se především při optimalizaci a na vytváření nových spustitelných struktur. Podrobně popsáno v kapitole 2.4.

2.4 Genetické programování

Vznik genetického programování (GP) se datuje do 80. let 20. století a jeho duchovním otcem je John Koza [4]. Liší se od jiných EA tím, že pracuje se spustitelnými strukturami. Nejčastěji to jsou programy reprezentované stromy s proměnnou délkou, které se vykonají průchodem stromem. Listy stromu jsou tvořeny terminály (vstupy, funkce bez argumentů a konstanty) a vnitřní uzly reprezentují funkce s jedním a více argumenty. Konstanty lze generovat náhodně, anebo může existovat množina základních konstant, z které jsou jiné konstanty tvořeny za pomoci aritmetických operací. Množina funkcí může být obecná, nebo specifická pro daný problém. Funkce by měly být zvoleny tak, aby se terminály a výstupy funkcí mohly použít jako vstupy funkcí – tzv. *uzavřenost*. Pro funkce, které jsou pro některé vstupní hodnoty nedefinované, se používají tzv. *chráněné varianty* – funkce místo nedefinované hodnoty vrátí předem zvolenou hodnotu.

Počáteční populace je vygenerována z náhodně vybraných funkcí a terminálů. Pro úpravu již existujících jedinců jsou na chromozomy aplikovány operátory křížení a mutace.

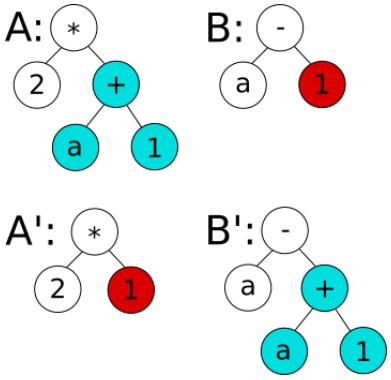
Existují tři strategie, jak vytvářet nové jedince:

> *Grow* – jako uzly jsou náhodně vybrány terminály nebo funkce, až do chvíle, kdy je dosažena maximální hloubka stromu. Stromy tak mají hloubku od 1 po maximální.
>
> *Full* – jako uzly jsou vybírány pouze funkce až do maximální hloubky stromu. Následně je tento rozvoj ukončen výběrem jednoho z množiny terminálů.
>
> *Ramped half-and-half* – generovány stromy s různou délkou až po maximální, kde je na jednu polovinu stromů použita strategie Grow a na druhou strategie Full.

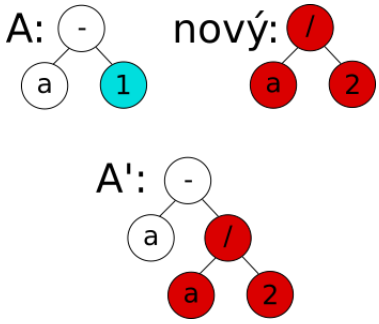
Křížení (obr. 2.7) je základní operace, do které vstupují dva jedinci (rodiče), kteří byli vybráni některým ze selekčních algoritmů. U každého z rodičů je vybrán jeden uzel a jeho podstrom, které si mezi sebou prohodí. Pokud je potřeba zachovat rodiče, musí se před křížením vytvořit jejich kopie.

Operace mutace (obr. 2.8) mění s určitou pravděpodobností geny jedince. Pravděpodobnost mutace je obvykle velmi malá a jedná se o operaci, která vnáší do populace rozmanitost. V GP se náhodně vybere jeden uzel a nahradí se novým náhodně vygenerovaným podstromem.

Fitness kandidátního řešení závisí na aplikaci. Často se využívá trénovací množina, která obsahuje různé vstupy programu a jim odpovídající výstupní hodnoty. Kandidátní program je proveden pro každý záznam z trénovací množiny a jeho výstupní hodnoty jsou



Obrázek 2.7: Křížení dvou stromů



Obrázek 2.8: Mutace uzlu

porovnány s požadovanými. Fitness f může být vypočítána vzorcem

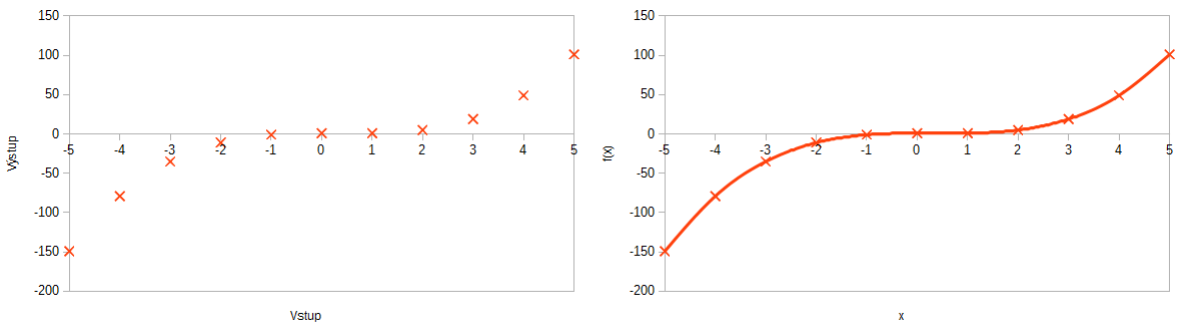
$$f = \sum_{i=1}^{N} (y_i^{GP} - y_i)^2 \tag{2.3}$$

kde N je počet vstupních hodnot, y_i^{GP} výstupy kandidátního programu a y_i požadované hodnoty. Jinou možností je zadání povolené odchylky výstupu kandidátního programu od referenčního, a pokud získaná výstupní hodnota spadá do této oblasti, je brána jakožto správná. Fitness je pak vypočítána jako poměr správně nalezených hodnot k celkovému počtu výstupních hodnot [8].

S neomezenou možnou velikostí syntaktického stromu může nastat tzv. *bloat*. S aplikováním operací může velikost chromozomu narůstat bez toho, aniž by se zvyšovala hodnota fitness. Narůstá tak čas a paměť, kterou program spotřebovává. To lze omezit například nastavením maximální výšky stromu nebo zahrnutím velikosti stromu do fitness funkce [6].

2.4.1 Symbolická regrese

Typickou úlohou, kde se uplatňuje GP, je symbolická regrese. Symbolická regrese se zabývá hledáním matematického modelu, který co nejvíce odpovídá vzorku zadaných dat. Hledá funkci popisující vztah mezi nezávislými (vstupy) a závislými (výstupy) proměnnými. Přesnost metody závisí na počtu vzorků a na použitých dílčích funkcích [4]. Na obrázku 2.9 jsou vlevo zaznamenané vstupní a výstupní hodnoty a vpravo nalezená funkce.



Obrázek 2.9: Symbolická regrese ($y = x^3 - x^2 + 3$)

Výpočet hodnoty fitness u úlohy symbolické regrese lze provádět dvěma způsoby. Prvním je výše uvedená celková odchylka výstupu kandidátního programu od požadovaného výstupu (viz vzorec 2.3). Druhá možnost je zjišťovat, zda programem nalezený bod spadá do jistého okolí referenčního bodu. Hodnota h určující, zda nalezený bod dostatečně odpovídá požadovanému výsledku, je určena vzorcem

$$h = \begin{cases} 1 & |y^{GP} - y| < \epsilon \\ 0 & \text{jinak} \end{cases} \tag{2.4}$$

kde y^{GP} je výstup kandidátního programu, y požadovaná hodnota a ϵ je velikost okolí, do kterého může ještě bod spadat, aby byl určen jako validní. Hodnota fitness f je pak určena jako poměr hodnot blízkých referenční hodnotě ku počtu testovacích vektorů vzorcem

$$f = \frac{\sum_{i=1}^{N} h_i}{N} \tag{2.5}$$

kde N je počet testovacích vektorů.

Výhoda druhého způsobu výpočtu hodnoty fitness se projeví například v případech, kdy funkce má v některém bodu extrémní hodnotu. Může nastat situace, že se kandidátní řešení liší od hledaného pouze v tomto bodu. Pokud je v takovéto situaci použito hodnocení s celkovou odchylkou, rozdíl funkčních hodnot v tomto bodu výrazně ovlivní fitness hodnotu. Při druhém způsobu výpočtu je ale rozdíl funkcí vztažen na každý bod zvlášť a změna fitness hodnoty je minimální.

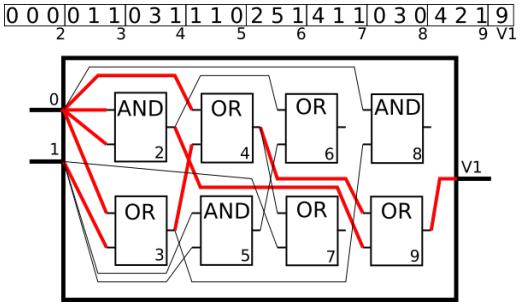
2.5 Kartézské genetické programování

Kartézské genetické programování (CGP) vzniklo z metody pro návrh číslicových obvodů vytvořené Julianem F. Millerem roku 1997. Roku 2000 bylo CGP představeno jakožto nová forma GP [7]. Hlavní využití CGP se nalézá v oblasti vývoje číslicových obvodů. Exisují totiž hradla, která mají dvě funkce, jež se přepínají, a ruční návrh, který používá tato hradla je téměř neproveditelný. Dalšími oblastmi ve kterých lze využít CGP jsou symboliská regrese, filtrace signálů, návrh antén nebo dokonce tvorba umění.

2.5.1 Reprezentace

Programy jsou reprezentovány jako forma orientovaných acyklických grafů na dvojrozměrné mřížce tvořené uzly – odtud pochází název „kartézské“. Množiny terminálů a funkcí jsou nahrazeny tabulkou funkcí a případných konstant, do níž se odkazují uzly. Každý uzel má v chromozomu zakódovánu svoji funkci a vstupy (obr. 2.10). Chromozom má konstantní délku, čímž se zabrání vzniku bloatu. Velikost fenotypu se však mění, neboť ne všechny uzly bývají použity pro výpočet výstupu. Takovým uzlům se říká neaktivní. Určení aktivních uzlů lze provádět rekurzivním sestupem s počátkem ve výstupu, nebo iterativně od posledního uzlu k prvnímu. Vznikne tak množina uzlů použitých pro výpočet výstupu.

Mřížka má předem určený počet sloupců (n_c) a řádků (n_r). Počet dostupných uzlů je pak $L_n = n_c n_r$. Vstupy uzlu mohou být připojeny na primární vstup nebo na výstup některého uzlu v předchozích sloupcích. Parametr l (*l-back*) určuje, o kolik sloupců vlevo může uzel připojit své vstupy. Pokud je $l = n_c$, může se uzel připojit na libovolný sloupec vlevo od něj. Primární výstupy obvodu se mohou připojit na kterýkoli uzel nebo primární vstup, ale nikdy nemohou být připojeny na jiný primární výstup [5].



Obrázek 2.10: Chromozom a jeho fenotyp

2.5.2 Mutace

Hlavním operátorem v CGP je mutace. Náhodně vybraný gen změní svoji hodnotu na náhodnou validní hodnotu. U genu představujícího funkci uzlu se jedná o index do tabulky funkcí, geny primárních výstupů a vstupů uzlů nabývají hodnot v závislosti na parametru l. První možností je náhodné zvolení 1–N genů, které změní hodnotu. Počet genů, které mají být zmutovány si volí uživatel, a to obvykle jako procento z počtu genů v genotypu. Další možností je vybrat všechny geny a měnit u nich hodnotu pouze s určitou pravěpodobností [2]. Čím blíž je mutovaný gen napojen k výstupu, tím větší je změna fenotypu. Stává se tak, že i mutace jediného genu má velký dopad na výsledný fenotyp, a nebo naopak mutace mnoha genů, které jsou momentálně neaktivní, nezmění fenotyp vůbec. Pokud mutace nezmění fenotyp, nazývá se neutrální. Při takové mutaci je hodnota fitness potomka stejná jako rodičova. Pokud je po několika neutrálních mutacích provedena mutace, která zapojí dosud neaktivní uzly, může vést k výrazným změnám fenotypu. Takto výrazné změny do jisté míry nahrazují křížení, které se v CGP standardně nepoužívá. Studie ukázaly, že křížení zde má spíše rušivý účinek.

2.5.3 Průběh evoluce

Evoluce je prováděna pomocí variace algoritmu $(1+\lambda)$. Populace je tvořena $(1+\lambda)$ jedinci, z nichž ten nejlepší je vybrán jako rodič a z něj je mutací vytvořeno λ potomků. Pokud má potomek stejně dobrou nebo lepší fitness než rodič, stává se rodičem v příští generaci. Kdyby se stal novým rodičem potomek pouze tehdy, když má vyšší fitness, evoluce by nacházela méně kvalitní řešení [5, 8].

Kapitola 3

Návrh

V této kapitole je popsán návrh programu, který zkoumá efektivitu různých způsobů mutací v CGP na úloze symbolické regrese. Kapitola se zaměřuje na konkrétní použití teoretických základů položených v kapitole 2.5 při tvorbě programu.

3.1 Genotyp a tvorba počáteční populace

Pro úlohu symbolické regrese má šířka mřížky uzlů velikost 1, neboť zde nezáleží na rozložení uzlů, ale pouze na pořadí. Připojení vstupu je možné na jakýkoliv uzel vlevo ($l = n_c$). Každý uzel je reprezentován strukturou o čtyřech položkách – dva vstupy, funkce a hodnota uzlu. Hodnota uzlu je použita pro uchování hodnoty v případě konstanty, nebo pro uchování hodnoty při výpočtu hodnoty fitness.

Konstanty jsou buďto předem definovány, pokud se jedná o časté nebo speciální hodnoty $(0, 1, \pi, \mathrm{e})$, nebo generovány náhodně z intervalu ⟨-10,00; 10,00⟩. Aritmetické funkce jsou vybírány z tabulky funkcí 3.1. Sinus, cosinus a absolutní hodnota mají jeden vstup, konstanta žádný.

Číslo funkce	Funkce	Číslo funkce	Funkce
0	Konstanta	5	$\sin(x)$
1	+	6	$\cos(x)$
2	-	7	e^x
3	*	8	$\ln(x)$
4	/		

Tabulka 3.1: Aritmetické funkce použité v programu

Počet uzlů (délka choromozomu), počet vstupů, počet výstupů a velikost populace je volena parametry. Velikost populace je obvykle volena kolem 8 jedinců.

Zobrazení výrazu, které propojené uzly představují, je prováděno postupným přepisem uzlů ve výrazu na funkce vstupů. Začíná u uzlu napojeného na výstup a přepisuje uzly ve výrazu na funkce uzlů a jejich vstupy až do doby, než výraz obsahuje pouze funkce, které operují se vstupy nebo s konstantami.

3.2 Ohodnocení jedince

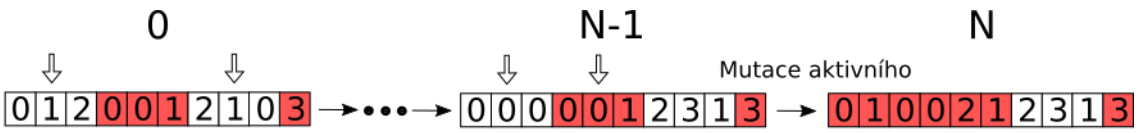
Aktivní uzly jsou nejprve vybrány pomocí rekurzivního sestupu a poté jsou pro každý vstupní vektor v souboru získány výstupní hodnoty. Každá získaná výstupní hodnota je ohodnocena dle vzorce 2.4. Velikost okolí ϵ je volena před spuštěním programu. Pokud se při výpočtu y^{GP} objevila nedefinovaná hodnota (např. výsledek dělení nulou), je automaticky ohodnocení výstupní hodnoty $h = 0$. Celková hodnota fitness je vypočítána podle vzorce 2.5.

3.3 Rodiče a potomci

Nejlepší jedinec v populaci se stane rodičem (deterministická selekce). Pokud má více jedinců nejlepší ohodnocení fitness, dalším kritériem je délka výrazu a vybrán je ten s kratším výrazem, aby se zamezilo obdobě bloatu, který se v CGP projevuje až na úrovni fenotypu. Četnost jednotlivých uzlů použitých při tvorbě výrazu je zjišťována za pomoci rekurze. Při shodě hodnot fitness i délek výrazů se rodičem stává jedinec, který v minulé generaci rodičem nebyl, aby se zvýšila diverzita populace.

Potomci jsou standardně tvořeni tak, že je změněno určité procento genů v genotypu a pro každého jedince je vždy znovu vypočítána hodnota fitness – mutace Normal (M_N). Tato mutace provádí příliš mnoho zbytečných evaluací. Pokročilejší druhy mutace, které se snaží provedené evaluace snížit, byly představeny v článku [2]:

> *Skip*(M_S) – stejné jako Normal, jen s tím rozdílem, že fitness se počítá jen v případě, kdy byl změněn aktivní gen. Pokud nebyl, nezmění se fenotyp ani hodnota fitness a potomek zdědí hodnotu fitness po rodiči. Tímto dojde ke znatelnému snížení počtu evaluací.
>
> *Accumulating Mutation*(M_{AM}) – mutace proběhne jako u Normal, ale pokud není změněn aktivní gen, mutuje se nově vytvořený jedinec znovu tak dlouho, než dojde ke změně aktivního genu. Pokud není horší než rodič, nahradí jej. V opačném případě je rodič nahrazen předposledním jedincem v této posloupnosti mutací (obr. 3.1).

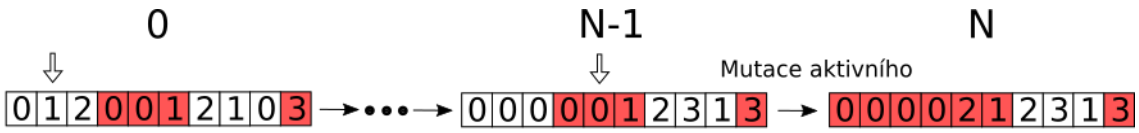


Obrázek 3.1: Accumulating Mutation – N-tý nahradí rodiče pokud není horší než on, jinak jej nahradí N-1

> *Single Active Mutation*(M_{SA}) – náhodně se vybere jeden gen a zmutuje se. To se opakuje tak dlouho, dokud není zmutován aktivní gen (obr. 3.2). Fenotyp se tedy vždy změní.

Mutace vstupu uzlu je provedena připojením na výstup náhodného uzlu, který mu předchází. Při mutaci funkce je přiřazena jiná funkce z tabulky. Pokud uzel představoval konstantu, je zde pravděpodobnost 50 %, že změní pouze svoji hodnotu.

Druh mutace a pravděpodobnost mutace jsou zvoleny parametry.



Obrázek 3.2: Single Active Mutation – N-tý nahradí rodiče

3.4 Ukončení

Program je primárně ukončen nalezením kandidátního řešení, které má hodnotu fitness rovnu nebo větší než je požadovaná hodnota fitness pro ukončení evoluce. Toto kandidátní řešení je pak označeno za řešení úplné. Pokud se řešení nedaří nalézt, existuje hodnota udávající počet generací, po kterých je evoluce zastavena a řešení zůstává nenalezeno.

Postačující fitness hodnota řešení a maximální možný počet generací jsou zadány parametry.

Kapitola 4

Implementace

Program je implementován v jazyce C++ se standardem C++11. Parametry programu a trénovací data jsou načítána ze vstupních souborů a výsledky programu jsou zaznamenávány do výstupního souboru.

4.1 Generování náhodných čísel

Pro účely stochastického prohledávání stačí náhodná čísla generovat generátorem pseudonáhodných čísel, jaký poskytuje funkce **rand**. Při dávkovém spouštění, kdy je program spuštěn vícenásobně v krátkém intervalu, nestačí inicializovat generátor pseudonáhodných čísel za pomoci funkce **time** na základě sekund. Proto je k inicializaci použita kombinace sekund a mikrosekund. Aktuální čas je získáván z knihovny *sys/time.h* na začátku běhu programu za pomoci funkce **gettimeofday**.

4.2 Měření času

V programu jsou měřeny dva časy, a to jak reálný čas, tak procesorový čas. Čas, který uběhl reálně, je určen pomoci rozdílu dvou hodnot získaných z funkce **gettimeofday**. Procesorový čas je získán ze struktury, kterou vrací funkce **getrusage**. Obě zmíněné funkce se nachází v knihovně *sys/time.h*.

4.3 Ukládání stavu programu

Kvůli velké časové náročnosti programu je potřeba počítat s možným přerušením běhu programu. Proto aplikace reaguje na signály následující zasílané operačním systémem [9]:

SIGTERM – je procesu zaslán v případě požadavku na ukončení procesu a může být ignorován.

SIGHUP – zasílán procesům při zavření terminálu.

SIGXCPU – přichází při překročení procesorového času vymezeného pro aplikaci. Po příchodu signálu má proces chvíli čas na reakci, než přijde *SIGKILL*.

SIGKILL – ukončí proces, nelze jej ignorovat.

Po detekci signálu se místo tvorby další generace zahájí procedura uchování dat poslední generace. Vytvoří se soubor s názvem *mezivypocet.dat*, do kterého je uloženo číslo generace, ve které byl přerušen výpočet. Dále je do něj zapsán počet provedených vyhodnocení kandidátních řešení, procesorový čas a reálný čas. Nakonec je do souboru zapsán obsah chromozomu jedince s aktuálně největší hodnotou fitness.

Jelikož soubor s uloženými daty o výpočtu neobsahuje žádné informace o trénovacích datech a nastavení programu, počítá s opětovným spuštěním bez změny parametrů. Při jejich změně jsou konečná data o průběhu evoluci chybná. Pokud při spuštění programu existuje soubor *mezivypocet.dat*, má to za následek změnu tvorby počáteční populace. Namísto náhodného tvoření jedinců je jeden jedinec načten ze souboru a určen jako rodič pro další populaci.

4.4 Vstup a výstup

Parametry a trénovací data jsou nahrávána ze dvou souborů s pevně danými jmény. Parametry evolučního algoritmu se nachází v souboru s názvem *settings.init*, trénovací data jsou obsažena v souboru se jménem *data.dat*.

4.4.1 Parametry

Soubor *settings.init* vypadá následovně:

`metoda_mutace(1.Normal_2.Skip_3.Accumulating_4.Single): 1` – číslicí v rozmezí 1 až 4 je volena strategie mutování jedinců. Číslem 1 je zvolena mutace Normal, 2 pro Skip, 3 pro Accumulating a nakonec číslo 4 znamená Single Active.

`hranice_fitness_funkce(<=1): 0.97` – minimální fitness hodnota, která je potřebná pro to, aby bylo kandidátní řešení prohlášeno za hledané řešení. Pohybuje se v rozmezí 0 (není potřeba, aby se nalezené řešení shodovalo s trénovacími daty v jediné hodnotě) až 1 (s trénovacími daty se musí shodovat všechny hodnoty).

`pravdepodobnost_mutace(min_0.0001): 0.0010` – s jakou pravděpodobností bude mutován každý gen v chromozomu. Minimální nastavitelná hodnota pravděpodobnosti je 0.0001 a maximální 1.

`maximalni_pocet_generaci: 1000000` – po kolika generacích má být algoritmus ukončen, bez ohledu na to, jak dobré řešení nalezl.

`velikost_populace: 8` – počet jedinců v populaci.

`pocet_uzlu: 32` – počet uzlů mřížky každého jedince.

4.4.2 Trénovací data

Trénovací data a jejich další specifikace se nachází v souboru s názvem *data.dat*:

`nazev_slozky: f3` – název složky, do které budou ukládány výsledky (podrobněji popsáno v 4.4.3).

`epsilon: 1.5` – udává rozdíl nalezené a trénovací hodnoty, s jakým bude ještě hodnota nalezená algoritmem považována za správnou.

`pocet_vstupu: 1` – počet nezávislých proměnných a tím pádem i počet sloupců trénovacích dat. Závislá proměnná je zde vždy jedna a tudíž přesný počet sloupců bude o jeden víc než je nezávislých proměnných.

`velikost_testovaci_sady: 200` – hodnota udávající počet trénovacích vektorů. Pokud je dále v souboru více trénovacích vektorů než udává tato hodnota, jsou přebývající data ignorována. V opačném případě jsou doplněny samé nuly.

`data:` – trénovací vektory, nejprve nezávislé hodnoty, po nich závislé. Sloupce dat jsou odděleny tabulátorem.

```
-10 162.2889592147
-9.9 144.9697594344
   ⋮
```

4.4.3 Výstup

Soubory s výsledky jsou ukládány do složky *./Data/nazev_slozky/*, kde *nazev_slozky* je určen v souboru *data.dat* (viz část 4.4.2). Pokud některá ze složek chybí, je automaticky vytvořena. Soubor s výsledky je pak pojmenován *Data(metoda_mutace–pravdepodobnost_mutace).dat*, kde *metoda_mutace* a *pravdepodobnost_mutace* jsou hodnoty ze souboru *settings.init* (viz část 4.4.1). Obsah souboru je následující:

```
44733600 31973 9.9133 8.15 1 (x0)+(((e^(sin(x0)))*(x0))*(x0));
```

První hodnota je počet provedených evaluací a druhá počet generací. Následují hodnoty reálného a procesorového času, po který evoluce probíhala. Páté číslo v pořadí je konečná hodnota fitness a za ní následuje předpis nalezené funkce $f(x0)$. Pokud nebylo řešení nalezeno, místo předpisu funkce se na konci řádku nachází slovo *Nenalezeno*.

4.5 Překlad a spuštění

Pro překlad a spouštění programu je k souborům se zdrojovým kódem přiložen *MAKEFILE*. Obsahuje následující příkazy:

`MAKE` – přeloží zdrojové kódy do jednoho spustitelného souboru s názvem *cgp*.
`MAKE run` – spustí jedenkrát soubor *cgp*.
`MAKE test` – *cpg* je prvně přeloženo a následně spuštěno několikrát po sobě. V jeho nastavení jsou prováděny změny dle skriptu *bcrun.sh*. Skript spustí program 50× pro 12 různých pravděpodobností mutace a všechny čtyři druhy mutací. Při mutaci single active je program spouštěn vždy jen jednou za 12 změn pravděpodobnosti a je ukládán do souboru s názvem *Data(4-0.8).dat*.
`MAKE clean` – smaže soubor *cgp*.

`MAKE pack` – smaže soubor *cgp* a zabalí zbylé soubory ve složce a podsložkách do zip archivu s názvem *cgp.zip*.

Soubor *bcrun.sh* obsahuje příkazy pro změnu pravděpodobnosti mutace a druhu mutace v souboru *settings.init* a následné několikanásobné spuštění.

Kapitola 5

Experimenty

Všechny čtyři druhy mutací jsou testovány na úlohách symbolické regrese, které byly představeny v článku [10]. Funkce jsou následující:

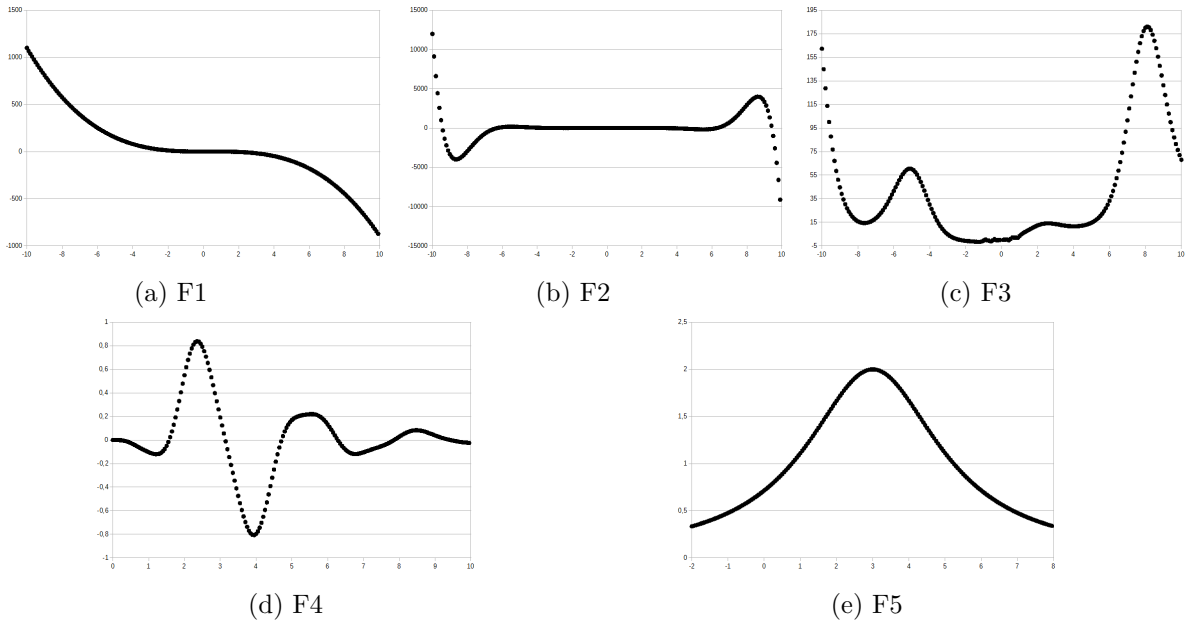
$$F1 : f(x) = x^2 - x^3, \qquad x \in \langle -10, 10 \rangle \tag{5.1}$$

$$F2 : f(x) = \mathrm{e}^{|x|} \sin(x), \qquad x \in \langle -10, 10 \rangle \tag{5.2}$$

$$F3 : f(x) = x^2 \mathrm{e}^{\sin(x)} + x + \sin(\frac{\pi}{x^3}), \qquad x \in \langle -10, 10 \rangle \setminus \{0\} \tag{5.3}$$

$$F4 : f(x) = \mathrm{e}^{-x} x^3 \sin(x) \cos(x) (\sin^2(x) \cos(x) - 1), \qquad x \in \langle 0, 10 \rangle \tag{5.4}$$

$$F5 : f(x) = \frac{10}{(x-3)^2 + 5}, \qquad x \in \langle -2, 8 \rangle \tag{5.5}$$



(a) F1 (b) F2 (c) F3

(d) F4 (e) F5

Obrázek 5.1: Grafy trénovacích dat

Pro každou z úloh bylo vybráno 200 rovnoměrně rozložených bodů z daného intervalu a byly vytvořeny sady trénovacích dat zobrazené na obrázku 5.1. Každý uzel v CGP má dva vstupy (i_1, i_2) a představuje vždy jednu z funkcí uvedenou v tabulce 3.1. Pokud má funkce jen jeden vstup, použije se pouze i_1. Parametry samotného CGP jsou zobrazeny v tabulce 5.1.

Počet vstupů (n_i)	1
Počet výstupů (n_o)	1
Rozměry mřížky ($n_c \times n_r$)	32×1
l-back (l)	32
Velikost populace	8
Minimální fitness hodnota pro úspěšné řešení	0,97
Povolená odchylka od trénovacích dat	F1, F2: 0,5 F3: 1,5 F4, F5: 0,025
Maximální počet generací	F1, F2, F3: 1 000 000 F4, F5: 5 000 000
Pravděpodobnost mutace jednotlivých genů (p_m)	0,001; 0,003; 0,005; 0,008; 0,01; 0,03; 0,05; 0,08; 0,1; 0,3; 0,5; 0,8

Tabulka 5.1: Nastavení CGP

Pro každou kombinaci úlohy, pravděpodobnosti mutace p_m a druhu mutace, byl program spuštěn 50krát. Vyjímku tvoří M_{SA}, kde díky nezávislosti na míře mutace stačilo spustit program pouze 50krát pro každou z úloh a v grafech tak bude mít zobrazenu jednu hodnotu pro všechny p_m. M_{SA} má v přepočtu $p_m = \frac{1}{96} \doteq 0{,}0104$ pro každý gen, neboť je vždy mutován jen jeden gen z celkového počtu 96 genů.

Druhy mutací, které jsou v experimentech použity (popsány v podkapitole 3.3), jsou následující:

- Normal (M_N)
- Skip (M_S)
- Accumulating Mutation (M_{AM})
- Single Active Mutation (M_{SA})

Experimenty byly spouštěny na výzkumném serveru *edesign3* s 64bitovým operačním systémem CentOS 6.5, procesorem Intel Xeon E5–2630 a 16 GB RAM.

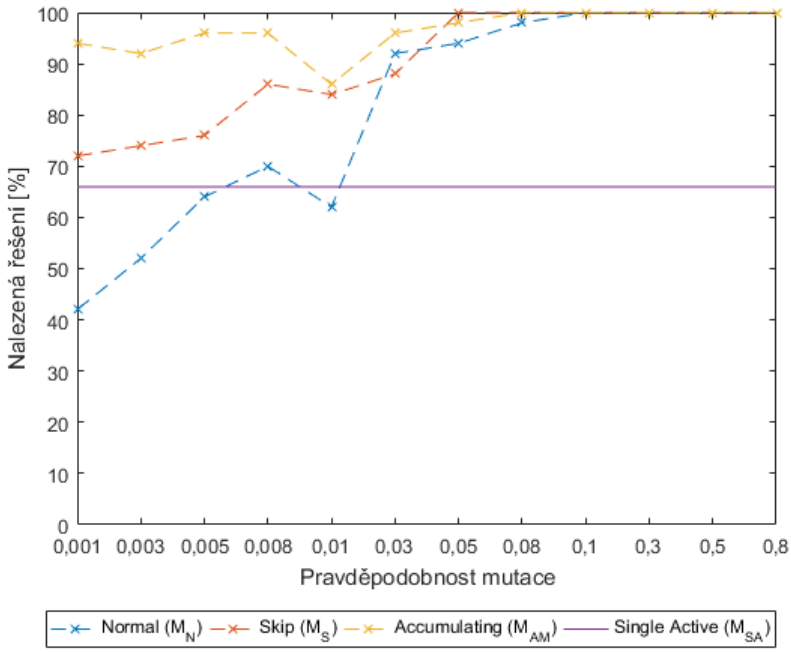
5.1 Výsledky experimentů

Hlavním ukazatelem úspěšnosti druhu mutace je procento běhů, ve kterých bylo nalezeno řešení a medián evaluací, které byly potřebné k úspěšnému nalezení řešení. Významnou hodnotou pro zvolené nastavení je $p_m = 0{,}01$, kde je průměrně mutován méně než jeden gen, a stává se tak, že mutace většinou nezmění chromozom a evoluce začíná stagnovat. Pokud p_m dále klesá, v populaci se vyskytuje stále méně změn, což vede ke značnému zpomalení evoluce a nalezení méně řešení za daný počet generací. Naopak, se zvyšující se

p_m počet změn v chromozomu roste a s nimi i úspěšnost. Od jisté míry mutace je však počet změn tak velký, že evoluci narušuje, začíná se chovat jako náhodné prohledávání a počet nalezených řešení se zmenšuje. M_N by měla provést největší počet evaluací při hledání řešení, neboť neobsahuje žádný systém na jejich redukci.

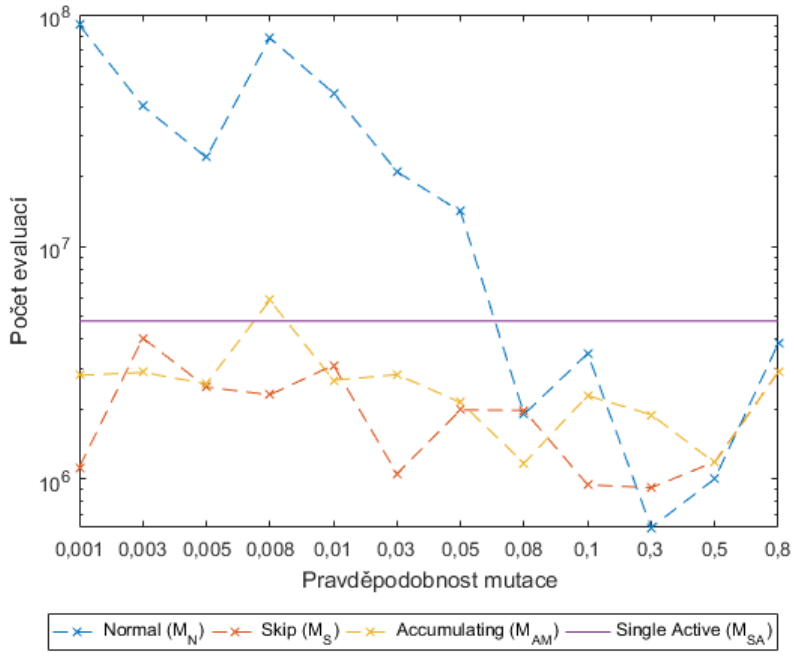
5.1.1 Úloha F1

Nejlehčí úloha *F1* (5.1) je řešitelná i za pomoci konvenčních matematických metod. Na grafu 5.2 je vidět procento úspěšných běhů v závislosti na míře mutace. Úspěšnost s mírou mutace roste a ani velká míra mutace nevede ke zhoršení (obvykle velká míra mutace narušuje vývoj jedince). Pro $p_m \leq 0{,}01$ se začíná projevovat výše zmíněná absence mutace u M_N a M_S. U M_{AM} se zhoršení neprojevuje tak značně díky opakovaným pokusům o mutaci. Pro nalezení řešení úlohy *F1* stačí i malý počet aktivních uzlů, a tak i jen občasná změna chromozomu dokáže nalézt řešení. Metoda M_{SA} nalezla řešení jen v 66% běhů a pro tuto úlohu vykazuje nejhorší výsledky (kromě M_N s velmi nízkou hodnotou p_m), protože její míra mutace je příliš malá.

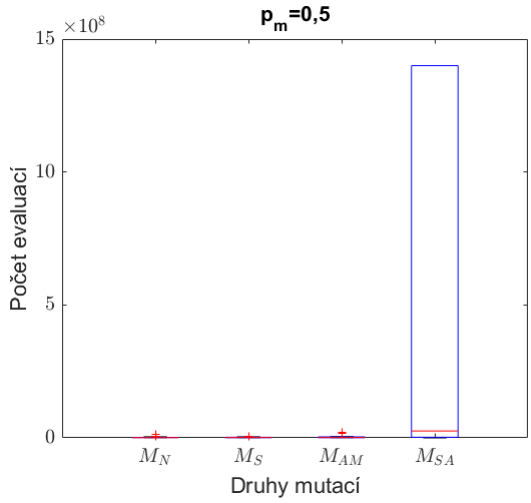


Obrázek 5.2: F1: Procento úspěšných běhů

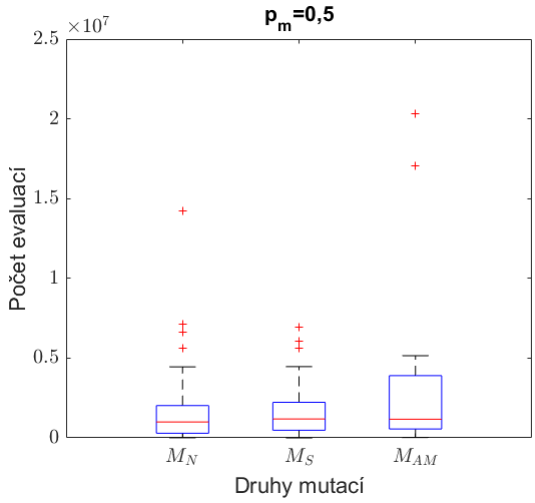
Graf 5.3 představuje medián evaluací potřebný pro nalezení řešení úlohy *F1*. Z důvodu nalezení řešení i po pár desítkách generací, nebo díky vygenerování řešení již v počáteční generaci, je počet evaluací u této úlohy spíše náhodný a neodpovídá přímo teoretickým předpokladům. I přesto jsou zde viditelné některé trendy, jako nejmenší počet evaluací pro všechny druhy mutací okolo $p_m = 0{,}3$ a zvyšující se počet evaluací se zmenšující se mírou mutace u M_N.



Obrázek 5.3: F1: Medián evaluací u úspěšných běhů



Obrázek 5.4: F1: Počty evaluací ze všech 50ti běhů



Obrázek 5.5: F1: Počty evaluací ze všech 50ti běhů – pouze první tři mutace

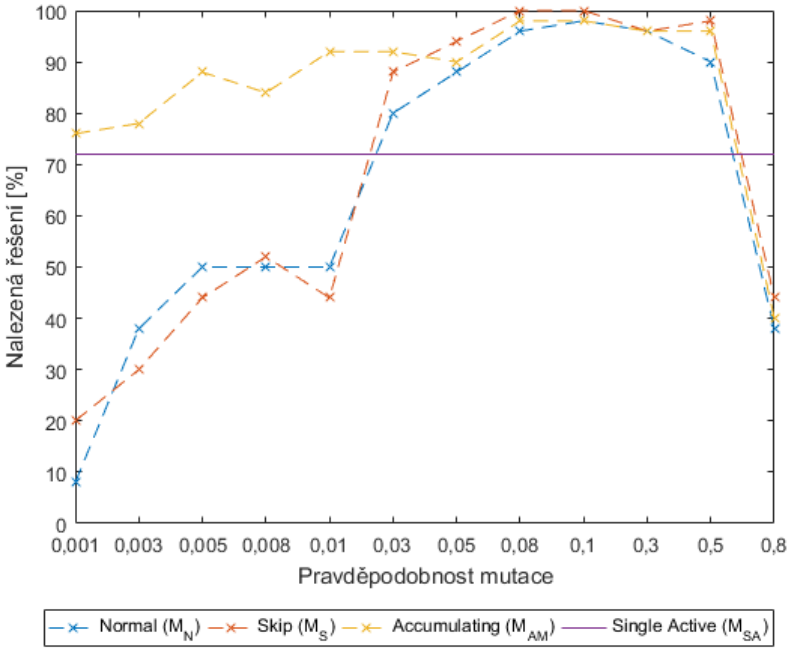
Pro $p_m \geq 0{,}05$ vykazují mutace závislé na pravděpodobnosti takřka stoprocentní úspěšnost a novým kritériem úspěšnosti jsou tak provedené evaluace. Nejvýhodnější nastavení pro M_N a M_S je $p_m = 0{,}3$, u M_{AM} se ukázalo jako nejlepší $p_m = 0{,}5$. Již v této úloze je vidět nevýhoda M_N, která je nejen v malé úspěšnosti, ale i řádovém rozdílu u počtu provedených evaluací (v průměru 13,6× víc než M_S a 10,5× víc než M_{AM}). Na kvartilovém grafu 5.4 pro $p_m = 0{,}5$ lze vidět pouze to, že mediány všech čtyř druhů mutace jsou ve stejných

řádech. Detailnější hodnoty nejsou rozpoznatelné kvůli velkému počtu evaluací u některých běhů M_{SA}, která jako jediná nevykazovala úspěšnost 100 % pro toto nastavení.

Graf 5.5 ukazuje už jen evaluace prvních tří druhů mutací. Počty evaluací pro M_N a M_S jsou takřka stejné, M_{AM} potřebuje v průměru víc evaluací (1,41× víc než M_N a 1,48× víc než M_S), ale její medián je takřka stejný jako u předchozích dvou.

5.1.2 Úloha F2

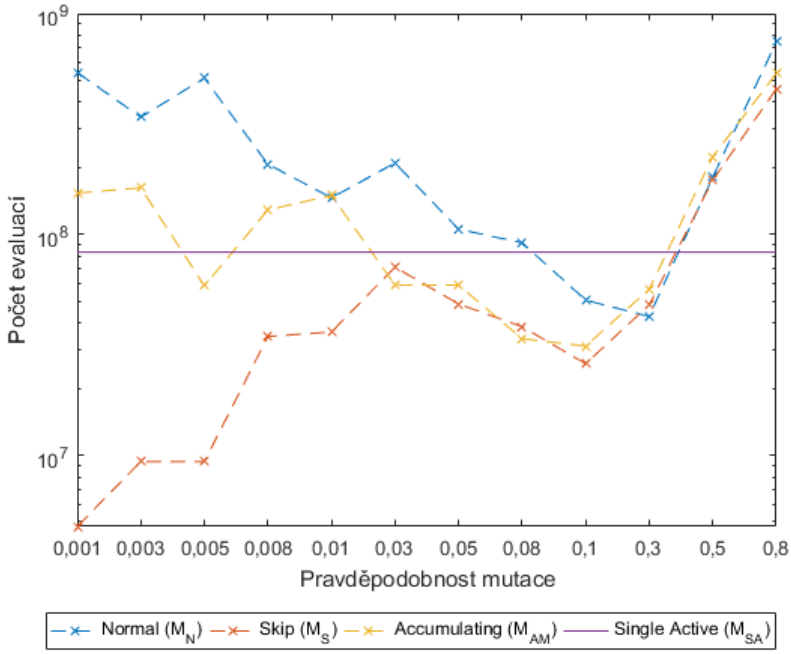
Úlohu *F2* (5.2) lze vyřešit za pomoci čtyř matematických operací a jedné konstanty, ale vyskytuje se v ní operace absolutní hodnota ze zadaného čísla, která není obsažena v tabulce funkcí, kterými CGP disponuje. Nalezený předpis bude tedy vždy pouze aproximací hledaného vztahu a lze předpokládat, že bude využito více uzlů, než je zde matematických operací, čímž úloha nabývá na složitosti. Graf 5.6, ukazující úspěšně nalezené řešení funkce *F2* pro jednotlivá nastavení programu, odpovídá vcelku přesně teoretickým předpokladům. Velká míra mutace a následné velké změny v chromozomu zde působí rušivě. Pro $p_m \leq 0{,}01$ je jasně viditelný pokles úspěšných běhů programu u M_N a M_S, která je při M_{AM} kompenzována opakovanou mutací genů. Největší počet úspěšných běhů se nalézá okolo nastavení $p_m = 0{,}1$. V případě $p_m = 0{,}01$, což odpovídá p_m M_{SA}, je M_{SA} úspěšnější než M_N (o 22%) a M_S (o 28%).



Obrázek 5.6: F2: Procento úspěšných běhů

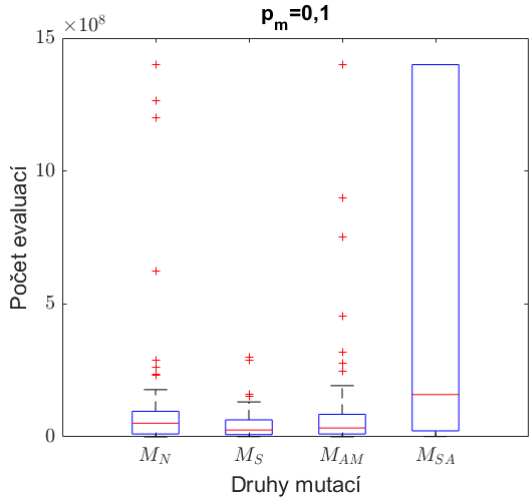
Medián evaluací potřebných k nalezení řešení (zobrazený na grafu 5.7) vykazuje nejlepší výsledky pro hodnotu $p_m = 0{,}1$. Zmenšování počtu evaluací M_S při $p_m \leq 0{,}01$ je dáno nale-

zením řešení jen v malém počtu běhů. Ze stejného důvodu u M_N nenarůstá počet evaluací tak závratným tempem, jak tomu bylo u úlohy *F1*.



Obrázek 5.7: F2: Medián evaluací u úspěšných běhů

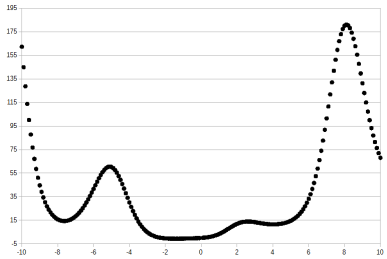
Největší úspěšnosti při nejmenším počtu evaluací bylo dosaženo při $p_m = 0{,}1$ (graf 5.8). Nejmenší počet nalezených řešení a největší průměrný počet provedených evaluací (jakožto i největší medián evaluací – $1{,}59 \times 10^8$) si drží M_{SA}. Na mediánu počtu evaluací si nejlépe v úloze *F2* vedla M_S, ale celkově nebyly další dva druhy mutací o mnoho horší.



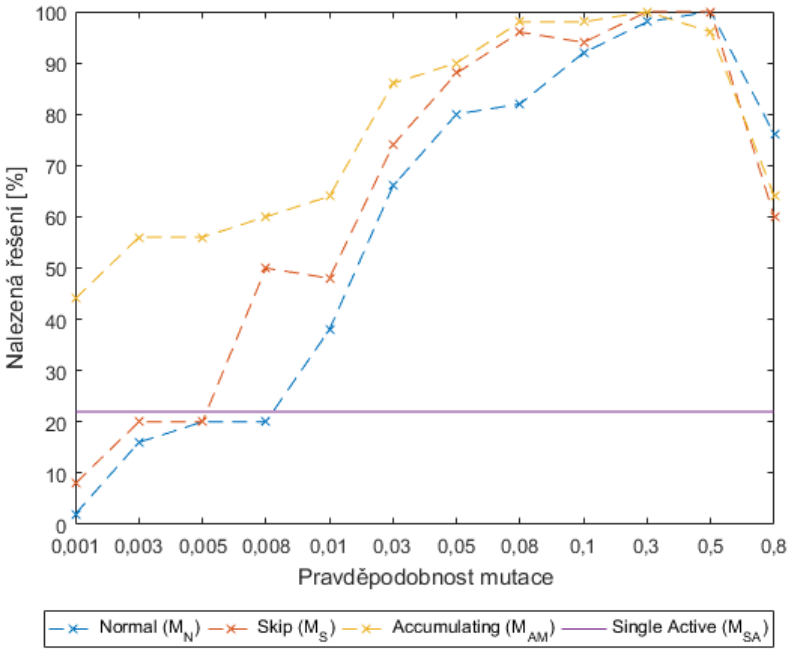
Obrázek 5.8: F2: Počty evaluací ze všech 50ti běhů

5.1.3 Úloha F3

Třetí úloha *F3* (5.3) obsahuje složitější člen na nalezení, kterým je $\sin(\frac{\pi}{x^3})$. Povolená odchylka od trénovacích dat je však nastavena tak, že dovoluje zanedbat celý člen $\sin(\frac{\pi}{x^3})$ (průběh funkce po zanedbání je zobrazen na grafu 5.9). Graf 5.10 se v mnohém podobá grafu 5.6 z úlohy *F2*. Velká míra mutace je rušivá a pro $p_m \leq 0{,}01$ nastává výrazné snížení počtu úspěšně nalezených řešení. Vzhledem ke zvýšené obtížnosti úlohy oproti úloze *F2* je celkový počet nalezených řešení menší. Nejvíce je to vidět u M_{SA}, která nalezla řešení v pouhých 22% běhů.

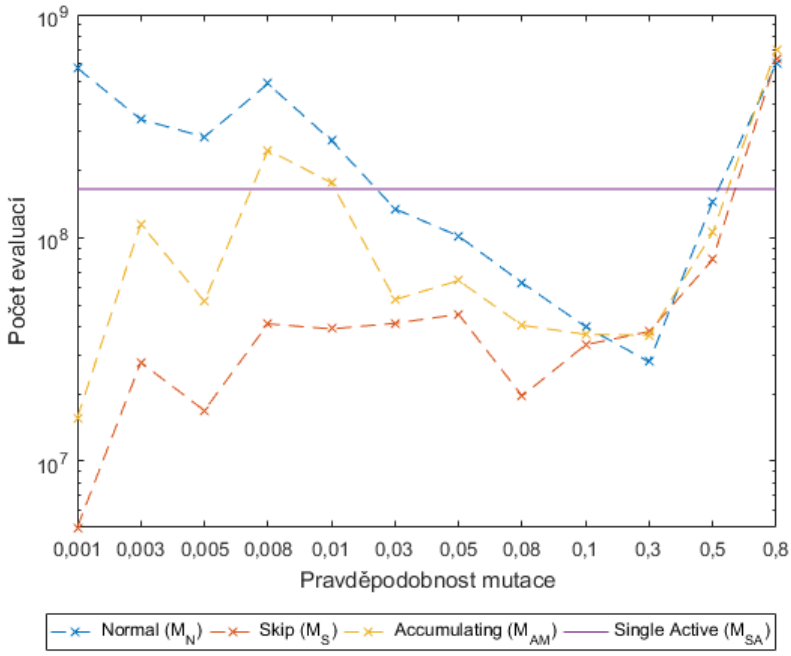


Obrázek 5.9: F3: Graf funkce bez členu $\sin(\frac{\pi}{x^3})$



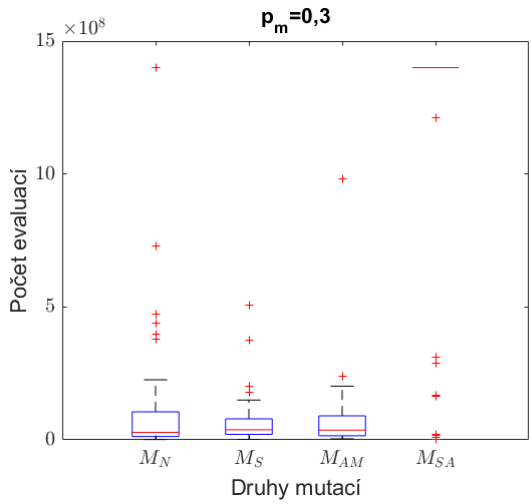
Obrázek 5.10: F3: Procento úspěšných běhů

Na grafu 5.11 se společné minimum evaluací pro mutace závislé na míře mutace nachází v $p_m = 0{,}3$.



Obrázek 5.11: F3: Medián evaluací u úspěšných běhů

Nejvhodnější z testovaných hodnot se ukázala být $p_m = 0{,}3$ (graf 5.12). Mediány evaluací prvních tří mutací jsou takřka na stejné úrovni (v rozmezí $2{,}8\times10^7$ až $3{,}8\times10^7$), nejmenší průměrnou hodnotu si drží M_S. M_{SA} vykazuje pro tuto úlohu velmi malou úspěšnost spolu s velkým počtem evaluací potřebných pro nalezení řešení a je tedy nevhodná k řešení této úlohy.

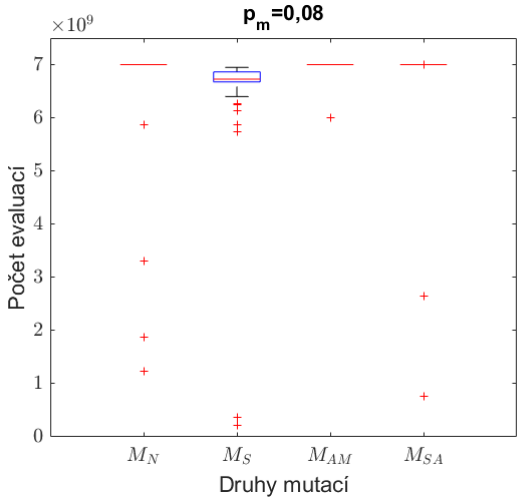


Obrázek 5.12: F3: Počty evaluací ze všech 50ti běhů

5.1.4 Úloha F4

Úloha *F4* (5.4) je první složitou úlohou v této sadě úloh. Obsahuje větší počet matematických operací a potřebuje tak i větší počet uzlů na reprezentaci. Kromě extrémně velké ($p_m \geq 0{,}5$) nebo malé ($p_m \leq 0{,}003$) míry mutace jsou řešení nacházena pro různé kombinace nastavení. Nalezení řešení je spíše náhodné. Nejvíce výsledků bylo nalezeno pro nastavení $p_m = 0{,}08$, a to 7 z celkem 150 běhů, tj. v 4,6 % případů.

Kvartilový graf 5.13 ukazuje celkové počty evaluací pro nejúspěšnější nastavení $p_m = 0{,}08$. Kromě běhů, kdy bylo nalezeno řešení, se všechny ostatní počty evaluací u M_N, M_{AM} a M_{SA} pohybují okolo $7{\times}10^9$. M_S se oproti tomu nedostane nikdy na tuto hranici, její medián má hodnotu $6{,}73{\times}10^9$ a projevuje se jako nejvhodnější druh mutace pro tuto úlohu.

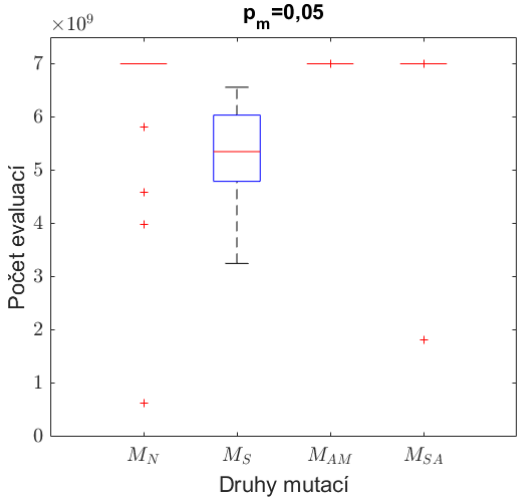


Obrázek 5.13: F4: Počty evaluací ze všech 50ti běhů

5.1.5 Úloha F5

Poslední úloha *F5* (5.5) se zaměřuje na nalezení předpisu obsahujícího větší konstanty, které nejsou pro CGP předem definovány. Nalezeny byly především složitější aproximace předpisu 5.5. Celkem bylo nalezeno pouze 11 řešení z 1850 běhů programu (úspěšnost 0,59%).

Kvartilový graf 5.14 ukazuje počty evaluací pro $p_m = 0{,}05$, kde bylo nalezeno 5 řešení (4× pro M_N a 1× pro M_{SA}). Opět, jako při úloze *F4*, jsou zde převážně neúspěšné běhy, které potřebovaly $7{\times}10^9$ evaluací u M_N, M_{AM} a M_{SA}. Medián evaluací pro M_S se vyšplhal pouze k hodnotě $5{,}35{\times}10^9$ ale M_S nenalezla jediné řešení. Avšak M_N řešení nalezla a M_S má stejnou pravděpodobnost na nalezení řešení. M_S je tedy nejvhodnější mutací pro tuto úlohu.



Obrázek 5.14: F5: Počty evaluací ze všech 50ti běhů

5.2 Shrnutí výsledků

Standardně používaná mutace M_N se ukázala jako nejnáročnější na počet evaluací, jak se předpokládalo. Pouze u jednodušších úloh a nejvýhodnějších nastavení vykazovala stejné výsledky jako M_S a M_{AM}. Při nasazení na složitější úlohy nebo při neznalosti ideálního nastavení, což je obvyklé při řešení nových problémů, vykazuje M_N nejhorší výsledky.

Experimenty ukázaly, že M_S se nejlépe hodí pro úlohy, které je pro CGP složité vyřešit – úlohy *F4* (úspora 3,90 % evaluací) a *F5* (úspora 23,54 % evaluací). Dokáže se za pomoci nejmenšího počtu evaluací dostat k maximálnímu povolenému počtu generací a tak nejrychleji ukončit běh programu, který nenalezl řešení.

M_{AM} dokáže do jisté míry kompenzovat malou míru mutace díky neustálým pokusům o mutaci genu, dokud není změněn aktivní gen. U lehčích úloh (*F1*, *F2*, *F3*) nalezne řešení v největším počtu případů. Je nejlepší volbou v případě neznámé optimální míry mutace pro danou úlohu, o níž je známo, že patří mezi úlohy, které CGP zvládá obstojně řešit.

Poslední mutace M_{SA} vykazovala průměrně nejhorší výsledky u všech testovaných úloh. Změní vždy minimálně jeden gen, ale to stejné dělá i M_{AM}, která má navíc možnost si vybrat při tvorbě potomka ze dvou chromozomů. Avšak vede si lépe než M_N a M_S v případech, kdy je míra mutace příliš malá a u těchto dvou druhů mutací již průměrně nedochází ke změně chromozomu při každém pokusu o mutaci.

Kapitola 6

Závěr

Hlavním cílem práce bylo navrhnout a implementovat program, který experimentálně otestuje různé druhy mutačních operátorů používaných v kartézském genetickém programování na úloze symbolické regrese. Pomocí volby správného druhu mutace lze snížit počet evaluací potřebných k doběhnutí programu. Občas tak jde zvýšit i úspěšnost nalezení řešení pro určité nastavení pravděpodobnosti mutace.

Navržený program byl implementován v jazyce C++. Mutace byly porovnány na pěti úlohách symbolické regrese stupňující se obtížnosti. Experimenty ukázaly, že druh mutace je dobré zvolit podle znalostí o řešené úloze a o nastavení. V případě, že je úloha lehce řešitelná za pomoci kartézského genetického programování, je nejlépe zvolit Accumulating Mutation, mutující chromozom tak dlouho, dokud není změněn aktivní gen. V opačném případě nejlepších výsledků dosahuje mutace Skip, která vypočítá hodnotu fitness jen v případě, že došlo ke změně fenotypu. V praxi tato skutečnost není většinou známa, ale vzhledem k využití symbolické regrese lze předpokládat, že se bude jednat o naměřená data. Jako taková ve většině případů neodpovídají teoretickému vzorci a jedná se o aproximace často obsahující konstanty. To tyto úlohy řadí mezi složitější a je vhodné použít mutaci Skip.

Při této práci jsem se seznámil se základy evolučních algoritmů a jejich implementací. Důkladněji jsem prostudoval mutační operátory u kartézského genetického programování. Rozšířil jsem si znalosti ohledně jazyka C++, a to především v oblasti zpracování signálů zasílaných procesům. V neposlední řadě jsem se naučil sázet odborný text za pomoci značkovacího jazyku LaTeX.

Literatura

[1] Fogel, D. B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Hoboken : John Wiley & Sons, 2006, 274 s., ISBN 0-471-66951-2.

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

[3] Hynek, J.: *Genetické algoritmy a genetické programování*. Praha : Grada, 2008, 200 s., ISBN 978-80-247-2695-3.

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

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

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

[7] Miller, J. F.; Thomson, P.: Cartesian Genetic Programming. In *Genetic Programming*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, s. 121–132, ISBN 978-3-540-46239-2.

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

[9] The IEEE and The Open Group: Signals [online]. [cit. 2. května 2016]. Dostupné z: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/signal.h.html.

[10] Šikulová, M.; Sekanina, L.: *Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, kapitola Coevolution in Cartesian Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 182–193, ISBN 978-3-642-29139-5.