VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

VYUŽITÍ OPERÁTORU KŘÍŽENÍ V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

USE OF THE CROSSOVER OPERATOR IN CARTESIAN GENETIC PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR BROMNIK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HURTA**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154344

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Bromnik Petr**
Program:	Informační technologie
Název:	**Využití operátoru křížení v kartézském genetickém programování**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Seznamte se s kartézským genetickým programováním (CGP) a s možnostmi jeho využití pro řešení vybraných úloh.
2. Seznamte se s problematikou použití operátoru křížení v CGP.
3. Zpracujte studii na výše uvedená témata.
4. Navrhněte a implementujte alespoň dvě metody operátoru křížení v CGP.
5. Experimentálně vyhodnoťte vlastnosti navržených metod a porovnejte je s existujícím přístupem.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího pokračování.

Literatura:

- Kalkreuth, R. Towards Discrete Phenotypic Recombination in Cartesian Genetic Programming. In: Rudolph, G., Kononova, A.V., Aguirre, H., Kerschke, P., Ochoa, G., Tušar, T. (eds) Parallel Problem Solving from Nature – PPSN XVII. PPSN 2022. Lecture Notes in Computer Science, vol 13399. Springer, Cham. https://doi.org/10.1007/978-3-031-14721-0_5
- Další dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hurta Martin, Ing.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	30.10.2023

Abstrakt

Cílem této práce je navrhnout a implementovat dvě nové metody křížení v kartézském genetickém programování (CGP) a porovnat je s existujícím přístupem. CGP je typ evolučního algoritmu využívající acyklické grafy k reprezentaci spustitelných programů. Většina CGP aplikací pracuje výhradně s operátorem mutace, ale snahy o nalezení vhodného operátoru křížení stále pokračují. V této práci jsou dvě nově navržené metody křížení porovnávány na pěti úlohách symbolické regrese oproti standardnímu přístupu $1 + \lambda$ založenému čistě na mutaci. Výsledky experimentů ukázaly, že tyto metody naleznou řešení za podobný počet fitness evaluací jako $1 + \lambda$, ve dvou případech dokonce významně dříve.

Abstract

The aim of this paper is to propose and implement two new crossover methods in Cartesian Genetic Programming (CGP) and compare them with existing approaches. CGP is a type of evolutionary algorithm that uses acyclic graphs to represent executable programs. Most CGP applications use the mutation operator only, but the effort to find a suitable crossover operator is still ongoing. In this paper, the two newly proposed crossover methods are compared on five symbolic regression problems against the standard $1 + \lambda$ procedure based purely on mutation. Experimental results show that these methods find solutions in a similar number of fitness evaluations as $1 + \lambda$ and, in two cases, even significantly earlier.

Klíčová slova

kartézské genetické programování, symbolická regrese, evoluční algoritmus, křížení, rekombinace

Keywords

cartesian genetic programming, symbolic regression, evolutionary algorithm, crossover, recombination

Citace

BROMNIK, Petr. *Využití operátoru křížení v kartézském genetickém programování.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hurta

Využití operátoru křížení v kartézském genetickém programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Hurty. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Bromnik
6. května 2024

Poděkování

Děkuji Ing. Martinu Hurtovi za cenné rady, vhodné připomínky, trpělivost a ochotu. Mé poděkováni patři též MetaCentru za poskytnutí výpočetních zdrojů, které přispěly k získání dat během experimentální části práce.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Řešení problémů se zdá být pouze lidskou záležitostí, ale pokud se podíváme kolem sebe, zjistíme, že různé problémy se vyskytují i v přírodě. Spolu s těmito problémy lze v přírodě najít i jistá řešení, ať už se jedná o postup mravenčích kolonií při hledání potravy či myšlení pomocí neuronů. Jedním z přírodních mechanismů, který různé problémy řeší, je evoluce. Dá se říct, že organismy jsou řešením na problémy, které vznikají kombinací prostředí a těchto organismů samotných. A právě na evoluci staví jeden z přístupů jak řešit výpočetní problémy. Jsou to evoluční algoritmy.

V evoluci, stejně jako v evolučních algoritmech, posouvá vývoj náhodná mutace a křížení organismů. Zatímco u mutace jsou změny náhodné (a mnohdy nevhodné), u křížení dochází k výměně informací, které v sobě jedinci již obsahují. Původně však křížení v přírodě neexistovalo a jedinci se množili čistě asexuálně. Podobně je tomu tak u jedné z variant evolučních algoritmů, kartézském genetickém programování (CGP). CGP na rozdíl od většiny jiných evolučních algoritmů křížení nepoužívá, jelikož doteď nebyla nalezena metoda, která by spolehlivě vedla k lepším výsledkům než při použití samotné mutace. Přesto ale snahy najít takovýto operátor křížení stále pokračují a někteří výzkumníci dosáhli i velmi slibných výsledků.

Cílem této práce je navrhnout dva nové operátory křížení pro CGP a porovnat je s existujícím přístupem. Teorie potřebná k návrhu metod křížení a experimentům je prezentována v kapitole 2. Tato kapitola popisuje evoluční algoritmy, genetické programování, CGP a symbolickou regresi. Navržené metody jsou popsány v kapitole 3. Pro experimentální účely bylo implementováno CGP v různých variantách, tato implementace je pak detailněji popsána v kapitole 4. Nakonec jsou navržené metody porovnávány na pěti problémech symbolické regrese se standardním přístupem. Výsledky těchto experimentů jsou zpracovány v kapitole 5.

Kapitola 2

Evoluční algoritmy

Evolučními algoritmy (EA) se chápe přístup přímého stochastického hledání řešení určitého problému. EA jsou inspirované přirozenou evolucí, tedy postupným vývojem organismů vzhledem k jejich prostředí s cílem přežít. I když se v běžných evolučních algoritmech nepoužívají žijící organismy či simulace reálného prostředí, prvky jako gen, jedinec, populace či mutace a selekce zůstávají. [1]

Evoluční algoritmy se, mimo jiné, skládají ze tří důležitých částí: tok řízení, mapování genotypu na fenotyp a množina genetických operátorů, typicky mutace a rekombinace [1]. Ve většině případů se pro nejlepší výsledky používají jak mutace pro prohledávání stavového prostoru, tak i rekombinace pro lokální optimalizaci.

S myšlenkou evolučního algoritmu přišlo více výzkumníků během druhé poloviny 20. století. Prvními z těchto výzkumníků byli Fogel a kol. [3], kteří přišli s návrhem evolučního programování (EP), zaměřeného na vývoj konečných automatů. Následovaly genetické algoritmy (GA) navržené Hollandem [4], který byl inspirován genetickým kódem organismů. Reichenberg [16] pak představil evoluční strategie (ES) zaměřené na technické problémy. Jako poslední, díky Kozovi a Polimu [10], přibylo genetické programování (GP). Dohromady se tyto čtyři techniky označují jako evoluční algoritmy [1]. Na základech zmíněného genetického programování pak stavěl mimo jiné i Miller [14], který přišel s návrhem kartézského genetického programování (CGP).

2.1 Důležité pojmy

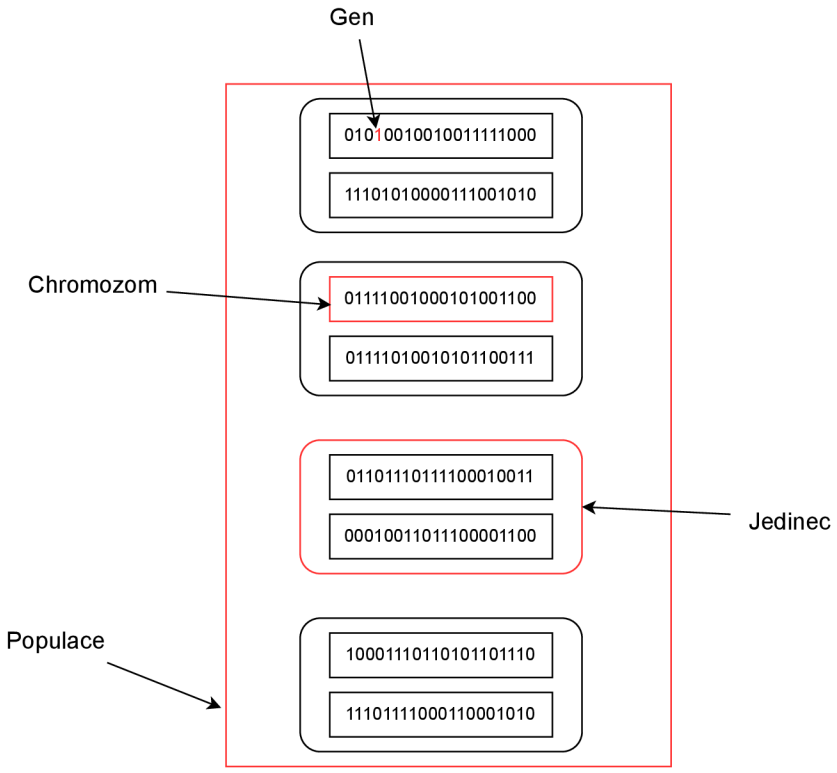
V oblasti evolučních algoritmů se často používají pojmy které mají původ v biologii. Jejich význam však nemusí být zcela zřejmý. Popisy pojmů vycházejí z [14], [1] a [18].

Populace – Množina jedinců. Její velikost se může lišit, v jednom běhu algoritmu ale bývá konstantní, viz obrázek 2.1.

Jedinec – Představuje možné řešení problému. Jeho reprezentace může být různá, použít se mohou binární řetězce, pole reálných hodnot či jiné.

Chromozom – Samostatná sekvence genů. Jedinec jich může obsahovat libovolné množství. Pokud jedinec obsahuje pouze jeden chromozom, tak je chromozom synonymem pro jedince.

Gen – Bývá atomický celek, který představuje jednu informaci, například 0 nebo 1 v binárním řetězci.



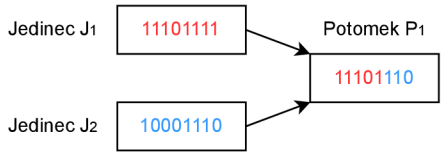
Obrázek 2.1: Populace o čtyřech jedincích. Každý jedinec obsahuje dva chromozomy, které jsou reprezentovány binárními řetězci. Chromozomy se skládají z jednotlivých genů, v tomto případě bitů.

Genotyp – Celková sekvence genů, které jsou v jedinci obsaženy. Může se opět zaměňovat s jedincem nebo chromozomem, v případě že jedinec obsahuje pouze jeden. Například v biologii představuje genetické instrukce zakódované v DNA.

Fenotyp – Dekódovaný genotyp, tedy řešení, které daný genotyp představuje. V biologii je jím zamýšlena fyzická forma organismů.

Mutace (bodová) – Genetický operátor, který změní jeden nebo více náhodných genů na jinou, validní náhodnou hodnotu.

Rekombinace - Nebo také křížení, je genetický operátor, který kombinuje geny již existujících jedinců v populaci. Příklad rekombinace lze vidět na obrázku 2.2. Jde o jednobodové křížení. Napravo a nalevo od tzv. "bodu křížení" jsou vyměněny geny.



Obrázek 2.2: Jednobodové křížení dvou jedinců reprezentovaných binárními řetězci. Délka chromozomu je v tomto případě osm, jako bod křížení byla vybrána pozice mezi pátým a šestým genem. Vlevo od této pozice dědí potomek geny jedince J_1, vpravo pak geny jedince J_2.

Fitness – Číselná hodnota, která udává, jak dobré je kandidátní řešení neboli jedinec.

Fitness funkce – Funkce, která přiřazuje jedincům jejich fitness hodnotu. Někdy také označovaná jako objektivní funkce.

Generace – Označení pro populaci v určité iteraci algoritmu.

Selekce – Selekce znamená výběr jedinců. Má za úkol zvýšit pravděpodobnost výběru kandidátních jedinců s lepší fitness hodnotou. Jednou z takových selekcí rodičů je turnajová selekce, kterou lze vidět na obrázku 2.3. U obecné turnajové selekce je určen parametr velikosti turnaje K a parametr počtu potřebných rodičů N. Následně je N-krát proveden náhodný výběr K jedinců, kdy vždy je za rodiče vybráno řešení s nejlepší fitness z daných jedinců. Výběr je však prováděn s navrácením, jedinec tedy může být vybrán do několika turnajů.

2.2 Princip evolučního algoritmu

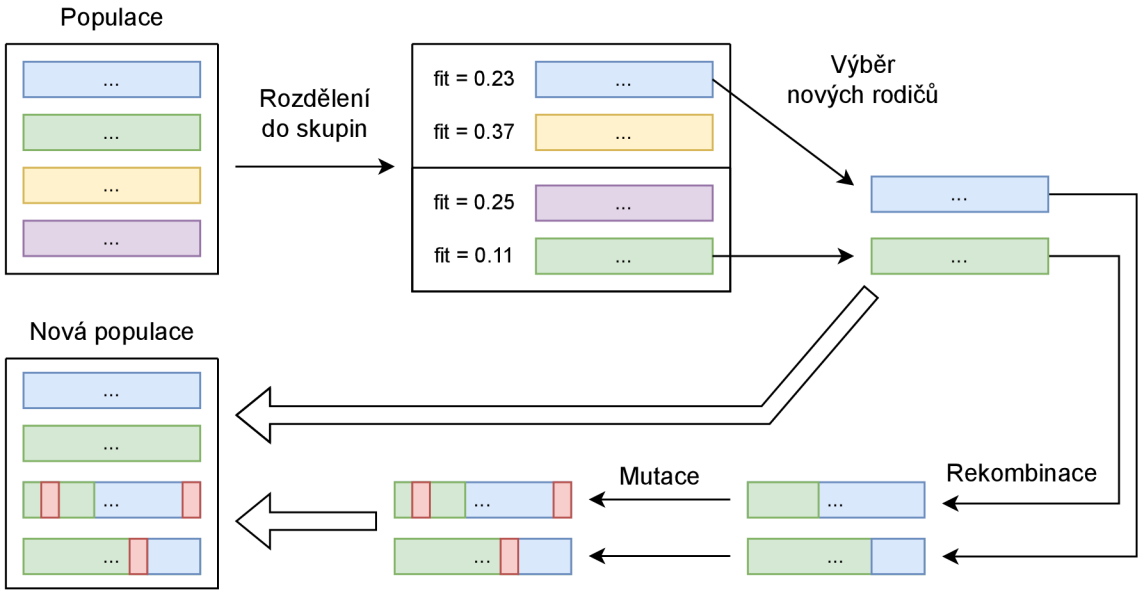
Zjednodušený průběhu EA je popsán algoritmem 1, inspirován [18]:

Algoritmus 1: Evoluční algoritmus

```
1: Vytvoř počáteční populaci
2: Ohodnoť jedince fitness funkcí
3: while není splněna ukončující podmínka do
4:     Vytvoř novou generaci pomocí genetických operátorů
5:     Ohodnoť jedince fitness funkcí
6: end
```

EA začíná vytvořením počáteční generace buď náhodně, nebo s pomocí heuristiky. Nově vytvoření jedinci jsou pak ohodnoceni fitness funkcí. Pak pokračuje cyklem, který se provádí, dokud není nalezeno dostatečně dobré řešení nebo není dosaženo výpočetní omezení dané uživatelem. [18]

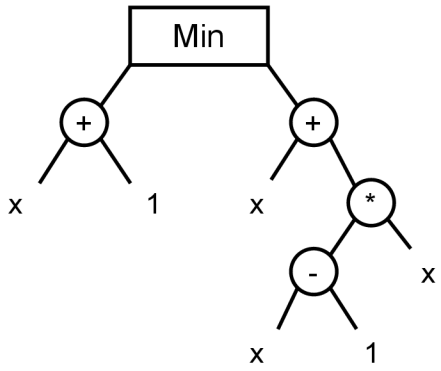
Za dostatečně dobré řešení se považuje takové, které má fitness menší nebo rovno přípustné hranici (v případě, že chceme fitness hodnotu minimalizovat). Omezení pak může být například počet generací, nebo počet fitness evaluací. V cyklu se generuje nová generace pomocí genetických operátorů. Jedním z možných algoritmů selekce se vyberou rodiče, kteří budou začleněni do nové populace. Křížením těchto rodičů se vytvoří potomci, na které je pak aplikován operátor mutace. Každému jedinci v nové populaci je přiřazena hodnota fitness a cyklus se opakuje. Příklad EA využívajícího turnajovou selekcí je na obrázku 2.3.



Obrázek 2.3: Příklad jedné iterace EA využívající turnajovou selekci kde $K = N = 2$. V tomto případě není možnost navrácení, každý jedinec se tedy zúčastní pouze jednoho turnaje. Potomci vznikají rekombinací rodičů a mutací. Nová populace se pak skládá z vybraných rodičů a jejich nových potomků.

2.3 Genetické programování

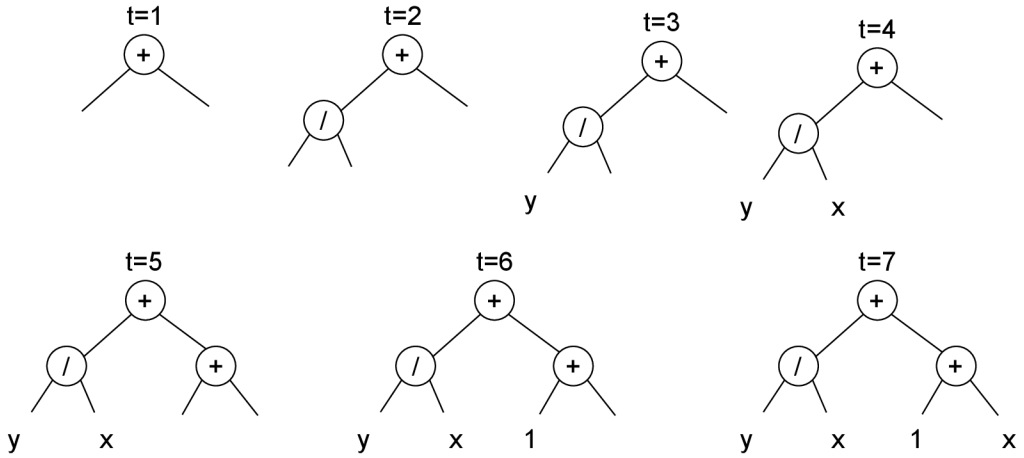
Je systematická metoda, která umožňuje počítačům automaticky řešit problém vycházející z vysokoúrovňového zadání. Každý jedinec znázorňuje počítačový program. Tyto programy nejsou vyjádřeny kódem, ale syntaktickými stromy. Stromy jsou složeny z hran a uzlů, které indikují instrukce k vykonání. Příklad takového stromu je na obrázku 2.4. Vnitřní uzly se nazývají funkce a listové pak terminály. Strom je tvořen z podstromů, neboli větví, které představují další podprogramy. Při určování fitness těchto jedinců reprezentovaných stromy, je program spouštěn zleva doprava, depth-first metodou. Kromě rekombinace a mutace, patří do vyhledávacích operátorů i duplikace a mazání genů. [10]



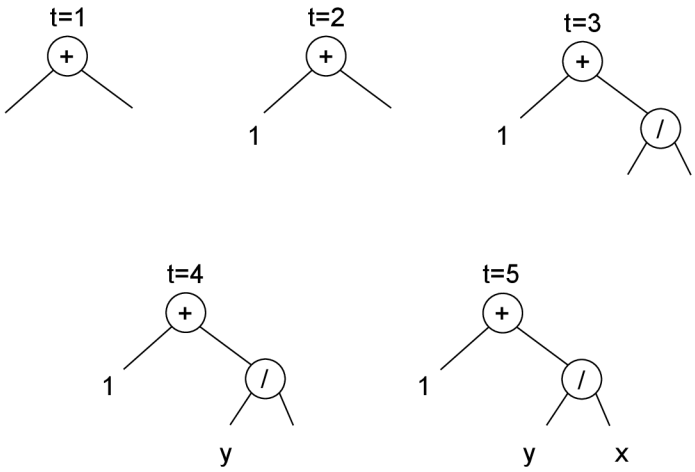
Obrázek 2.4: Syntaktický strom pro výraz $min(x + 1, x + x \times (x - 1))$.

GP začíná s počáteční populací programů, složených z vhodných funkcí a terminálů pro daný problém. Typicky se jedinci této počáteční populace generují rekurzivním vytvářením stromu programu, který se skládá z náhodných výběrů primitivních funkcí a terminálů zadaných uživatelem. [10]

Tzv. Full inicializační metoda používá funkce z funkční sady, dokud není dosažena maximální hloubka stromu. Pod touto úrovní se již mohou používat pouze terminály. Příklad vytváření jedince metodou Full je na obrázku 2.5. Full vždy vytváří takový strom, že jeho větve dosahují maximální hloubku. Metoda Grow je pak variantou metody Full, která využívá celou primitivní sadu, dokud nenarazí na hloubkové omezení stromu. Poté opět používá pouze terminály stejně jako metoda Full. Příklad vytváření jedince metodou Grow je na obrázku 2.6. [10]. Grow vytváří stromy, jejichž větve mohou dosahovat různé hloubky, jsou však také omezeny maximem, které zadává uživatel. [10]



Obrázek 2.5: Vytváření jedince v počáteční generaci pomocí metody Full, kde maximální hloubka stromu je $H_m = 2$. Jedinec zde znázorňuje výraz $(y/x) + (1 + x)$, t reprezentuje čas.

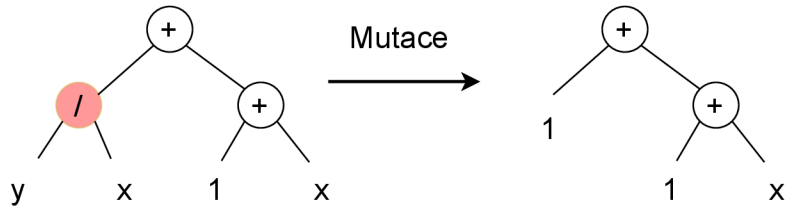


Obrázek 2.6: Vytváření jedince v počáteční generaci pomocí metody Grow, kde maximální hloubka stromu je $H_m = 2$. Jedinec zde znázorňuje výraz $1 + (y/x)$, t reprezentuje čas.

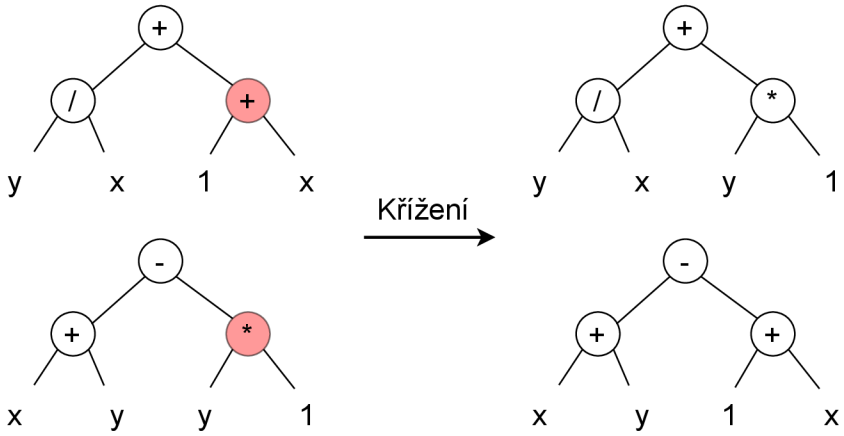
Kombinací metody Full a metody Grow je pak metoda Ramped half-n-half (RHH), která postupně inkrementuje hloubku generovaných stromů od dvou až po maximum zadané uživatelem. Pro každou z těchto hloubek RHH vytváří polovinu potomků metodou Full a druhou polovinu metodou Grow. Výsledkem je pak pestrá množina stromů různé hloubky a tvarů. [11]

Vytváření počáteční populace je pouze slepé náhodné zkoumání prostoru řešení, kterým se pokládá základ pro budoucí generace. Typicky mají jedinci v počáteční populaci velmi chabé fitness. Nicméně mezi nimi bývají jedinci, kteří mají lepší fitness než ostatní, čehož GP využívá. [10]

Při použití mutačního operátoru (obrázek 2.7) je náhodně vybrán jeden z uzlů jedince a je smazán celý podstrom, kde vybraný uzel představuje kořen. Následně je tento podstrom nahrazen novým podstromem, který je vytvořen podobným postupem jako při tvorbě počáteční populace. Příklad operátoru křížení v GP, který vytváří dva potomky, lze pak vidět na obrázku 2.8. V obou rodičích je náhodně vybrán jeden z uzlů jako bod křížení. Podstromy, ve kterých je vybraný uzel kořenem, se následně mezi rodiči vymění. [10]



Obrázek 2.7: Příklad použití operátoru mutace v genetickém programování. Jako bod mutace byl vybrán náhodný uzel (označen červeně). Následně byl podstrom tohoto uzlu smazán a nahrazen jiným náhodným podstromem.



Obrázek 2.8: Příklad použití varianty operátoru křížení v genetickém programování, která vytváří dva potomky. V obou jedincích byl vybrán náhodný uzel jako bod křížení (označeny červenou). Následně byly vyměněny podstromy rodičů s kořeny ve vybraném uzlu.

2.4 Symbolická regrese

Regresní analýza je proces pro odhad vztahů mezi proměnnými. Soustředí se na vztah mezi závislou proměnnou a jednou či více nezávislými proměnnými. Přesněji řečeno, regresní analýza pomáhá odhalit, co se stane se závislou proměnnou, pokud se změní nějaká z nezávislých proměnných.

Existuje více modelů regrese, nejjednodušší z nich je lineární regrese. Ta předpokládá, že funkční vztah mezi nezávislou proměnnou a parametry je lineární. Tento funkční vztah lze popsat rovnicí 2.1:

$$Y_j = \beta_0 + \beta_1 X_{1j} + \beta_2 X_{2j} + \cdots + \beta_p X_{pj} + \varepsilon_j, \qquad (j = 1, \ldots, n) \tag{2.1}$$

kde Y_j je závislá proměnná, X_{pj} jsou nezávislé proměnné, β_i jsou regresní koeficienty a ε_j jsou chyby modelu [17]. Y a X_p jsou v tomto případě vektory, proto jsou jejich komponenty značeny pomocí j.

Symbolická regrese je typ regresní analýzy, která má velmi blízko odvětví GP. Dá se říct, že je to základní úloha, kterou GP ze svého principu řeší. Na rozdíl od lineární regrese nepředpokládá lineární vztahy. Symbolická regrese se snaží najít takový model a parametry, které co nejlépe vystihují vztah mezi vstupními a cílovými hodnotami.

Datová sada pro symbolickou regresi obsahuje vstupní a cílové hodnoty. Cílem pak je najít takovou funkci, které namapuje každý bod ze vstupních hodnot do příslušné cílové hodnoty. Například řešení pro problém symbolické regrese $f(x) = x^2 - 2x$ by v ideálním případě namapovalo každý možný bod x do příslušného bodu $f(x)$ [14]. V realitě ale symbolická regrese pracuje pouze s body vektorů, které jsou obsaženy v datové sadě. V tomto případě by tyto vektory mohly být například:

$$V_x = [0, 1, 2]$$
$$V_{f(x)} = [0, -1, 0]$$

Symbolická regrese se pak snaží najít funkci, která mapuje následovně:

$$f(x) = \left\{ \begin{array}{ll} 0 & x = 0 \\ -1 & x = 1 \\ 0 & x = 2 \end{array} \right\}$$

Pokud takovou funkci nalezne, není garantované, že to bude právě ta, která byla požadovaná. Obzvlášť při práci s malou datovou sadou jako je tato. Naleznout bychom mohli jak funkci $f(x) = x^2 - 2x$, tak i třeba $f(x) = -(x \mod 2)$, jelikož se na těchto datech chovají identicky. Tato chyba se dá omezit větším počtem vstupních a cílových hodnot. Kromě toho může nalezené řešení mít nadbytečné množství prvků, které ve výsledku nic nemění. Například $f(x) = x^2 - \sin x + \sin x - 3x + x$. Zároveň může představovat problém šum v datech. Data pak nemusí přesně odpovídat výsledkům hledané funkce.

Kromě vstupních a výstupních hodnot používáme při návrhu řešení problémů symbolické regrese i množinu funkcí a terminálů. Tyto funkce se mohou skládat například ze sčítání, odčítání, násobení a chráněného dělení (dělení nulou může vrátit například 1). Ale mohou se rozšířit například o sinus, cosinus a další [14].

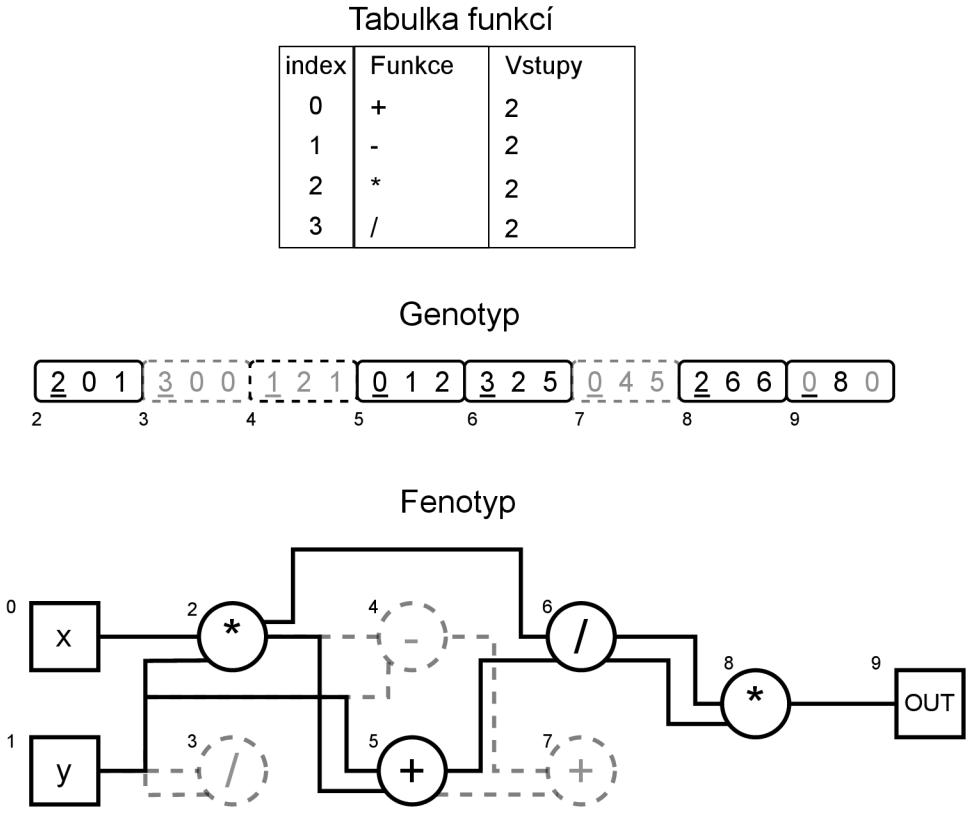
2.5 Kartézské genetické programování

V kartézském genetickém programování (CGP) se používá dvourozměrné pole uzlů, nikoliv stromy. Programy jsou pak reprezentovány jako orientované acyklické grafy. Genotyp se typicky skládá z jednoho chromozomu, který je rozdělen do uzlů. Ty pak obsahují jednotlivé geny.

2.5.1 Reprezentace kandidátních řešení

Genotyp má v CGP fixně danou velikost, na rozdíl od fenotypu, jehož velikost se může lišit. Minimálně může mít délku nula, maximálně pak stejnou jako počet uzlů v genotypu. Když se totiž provádí dekódování genotypu na fenotyp, ne všechny uzly musí být součástí aktivní cesty. Tedy zapojení uzlů, které vede mezi vstupem a výstupem jedince. Takovéto uzly které nejsou v aktivní cestě, se označují jako neaktivní nebo nekódující.

Uzly mohou být funkční (výpočetní), vstupní nebo výstupní. Každý z funkčních uzlů představuje nějakou funkci, které je složená ze dvou typů genů. Funkční gen odkazuje do tabulky funkcí (například plus, mínus, děleno) a je obsažen v každém uzlu pouze jednou. Zbytek genů jsou takzvané spojovací geny, které určují vstup pro danou funkci. Spojovací geny představují adresu jiného uzlu, většinou index v poli. Vstupní geny slouží jako vstupní data programu. Výstupní uzly obsahují index genu, ze kterého se má brát výstup programu. Ukázku jedince v CGP lze najít na obrázku 2.9.



Obrázek 2.9: Příklad jedince v CGP s parametry $lback = 2, n_c = 4, n_r = 2$. Podtržené geny ukazují do tabulky funkcí. Šrafovaná čára představuje neaktivní uzel, který se neprojeví ve fenotypu. Tento jedinec představuje funkci $(\frac{xy}{y+xy})^2$.

Počet vstupních genů je dán nejvyšší kardinalitou mezi funkcemi v množině dostupných funkcí. Všechny uzly mají tedy stejný počet genů [14]. Ne všechny vstupní geny se ale musí využít. Pokud máme uzly o třech vstupech, ale daný uzel reprezentuje binární operaci plus, použije pouze první dva vstupy.

V CGP jsou tři základní parametry: počet řádků n_r, počet sloupců n_c a počet úrovní (sloupců) zpět, o které může uzel brát vstup. Označuje se jako tzv. "levels back", nebo také *l-back*. Pokud by *l-back* bylo například jedna, jako vstup by uzly mohly používat pouze sloupce přímo před sebou nebo uzly vstupní. Počet výpočetních uzlů L_n pak dostaneme jako $n_c \times n_r$ [14].

2.5.2 Prohledávací algoritmus

Nejrozšířenější varianta CGP používá evoluční algoritmus $1 + \lambda$ který je popsán níže (algoritmus 2). Jedničkou se myslí jeden rodič a λ značí počet potomků. "+"zde znamená, že do nové generace přežívá i rodič. Opakem by pak bylo značení ",", kde nová generace obsahuje pouze potomky. Nejvyužívanější variantou $1 + \lambda$ je algoritmus $1 + 4$, kde je jeden rodič a čtyři potomci. Důležitou součástí algoritmu je fakt, že potomek se stane novým rodičem, i když má stejnou fitness jako rodič. Jednou z charakteristik $1 + \lambda$ je, že využívá tzv. "neutrálního driftu". Mutace totiž zasahuje i geny, které nejsou aktivní (těch dokonce bývá podstatně více, než genů aktivních) a nepodílí se tedy na fenotypu. Tyto nenápadné změny v neaktivních genech pak můžou vést k velkému rozdílu ve chvíli, kdy se stanou aktivními. Tímto způsobem se $1 + \lambda$ může dostat z lokálního optima [14].

Algoritmus 2: Evoluční strategie 1 + lambda

```
1:  Vytvoř počáteční populaci
2:  Ohodnoť populaci fitness funkcí
3:  Vyber jedince s nejlepším fitness a povyš jej na rodiče
4:  while není nalezeno řešení nebo není dosažen limit generací do
5:      forall i takové že i < λ do
6:          Vytvoř potomka i mutací rodiče
7:      end
8:      Ohodnoť populaci fitness funkcí
9:      Vyber nového rodiče následujícím způsobem:
10:     if alespoň jeden z potomků má lepší nebo stejnou fitness jako aktuální rodič
         then
11:         if vícero potomků má shodnou nejlepší fitness then
12:             Nový rodič je vybrán náhodně z těchto potomků
13:         else
14:             Novým rodičem je vybraný potomek
15:         end
16:     else
17:         Rodič zůstává nezměněn
18:     end
19: end
```

2.6 Křížení v CGP

Křížení se ve standardním CGP nepoužívá. Ve studii o účinnosti řešení booleovských funkcí pomocí CGP pro něj Miller nenašel využití [13]. Možnou příčinou je genetický drift, který umožňuje mutaci dělat jak velké tak malé změny a tím překonávat lokální optima [15]. To ovšem neznamená, že od vzniku CGP nebyly žádné snahy o zakomponování křížení. Návrhů se objevilo hned několik, některé z nich i se slibnými výsledky. Spoustu z nich je ale nutné otestovat na širším spektru úloh, aby bylo jasné, zda a kdy se jimi vyplatí nahradit standardní algoritmus $1 + \lambda$.

Reprezentace s reálnými čísly

Clegg et. al. [2] v roce 2007 předložili novou metodu křížení. Inspirovali se křížením z GA, které využívá reprezentaci reálnými čísly. Pro použití tohoto operátoru bylo však třeba upravit genotyp. Genotyp byl zakódován tak, že všechny geny byly reálné číslo v intervalu $\langle 0, 1\rangle$. Při dekódování genotypu se pak reálná čísla převáděla do klasické reprezentace genotypu celými čísly.

Při experimentech bylo křížení kombinováno s většími populacemi a turnajovou selekcí. Jimi navržená metoda křížení se ukázala být prospěšná na obou testovaných problémech symbolické regrese.

Kuželové křížení

Při zkoumání modulární reprezentace CGP v roce 2008 Kaufmann a Platzner [9] navrhli kuželové křížení v kombinaci s genetickým algoritmem. Jde o vytváření nového chromozomu tím, že kužel z dárcovského chromozomu je transplantován do klonu příjemce.

Výsledky experimentů ukázaly, že tato technika fungovala lépe než základní technika používaná v modulárním CGP pouze na dvou ze tří porovnávacích úloh (návrh elektrických obvodů).

Vícechromozomová reprezentace

V roce 2011 navrhli Walker et. al. vícechromozomovou reprezentaci CGP [19], která zahrnovala křížení. Tato reprezentace spočívá v tom, že genotyp je rozdělen do několika stejně dlouhých sekcí, které umožňují program rozdělit do vícero menších řešení. Použita pak byla upravená verze $1 + \lambda$, která současně fungovala i jako operátor křížení. Jelikož fitness každého chromozomu mohla být zjištěna nezávisle, nový rodič se sestavil z chromozomů s nejlepší fitness hodnotou.

Metoda byla experimentálně ověřena na vývoji elektrických obvodů, kde nová reprezentace obecně předčila klasické řešení s jedním chromozomem. U jednoho z problémů pak bylo možné získat řešení až 392× rychleji.

Výměna podgrafů

Kalreuth et. al. [7] přišli s návrhem na křížení výměnou podgrafů v roce 2017. Tato metoda je podobná jednobodovému křížení. Liší se ale v tom, že aktivní uzly jsou na obou stranách bodu křížení zachované. Na začátku křížení výměnou podgrafů je třeba najít obecný bod křížení C_P. Ten je definován výběrem menšího bodu křížení z C_{P1} a C_{P2}, kde C_{P1} a C_{P2} jsou náhodné funkční uzly ležící v aktivní cestě rodičů P_1 a P_2. Genetický materiál

pro sekci ležící před C_P je zkopírován z rodiče P_1 do genotypu potomka. Tento genetický materiál zahrnuje funkční uzly od začátku genotypu P_1 do funkčního uzlu daného C_P, včetně neaktivních uzlů. Pro genetický materiál sekce za bodem C_P se provede ten samý postup s genetickým materiálem rodiče P_2, s tím rozdílem, že tento materiál leží mezi C_P a výstupy. Po provedení těchto kroků následují úpravy výsledného potomka, za účelem napojení těchto dvou kopírovaných sekcí. Jinak by mohl nastat případ, kdy dříve aktivní uzly se stanou neaktivními. Jako evoluční strategii použili turnajovou selekci.

Ohodnocování algoritmu se provádělo na širší škále problémů. Přesněji pak symbolická regrese, návrh elektrických obvodů a návrh obrazových filtrů. Jejich algoritmus byl porovnán pouze v různých mírách křížení, nikoliv oproti klasickému $1 + \lambda$. Není tedy jasné, jak by jejich nový algoritmus obstál oproti zaběhnuté metodě založené čistě na mutaci [15].

Blokové křížení

V roce 2018 navrhli Kalkreuth a Husa blokové křížení [5]. Je inspirováno hlavně kuželovým křížením zmíněným v sekci 2.6 a výměnou podgrafů v sekci 2.6. Zaměřili se na jednodimenzionální pohled na genotyp. Po selekci nových rodičů z jejich genotypu jsou vybrány bloky, které splňují všechna následující kritéria:

- blok obsahuje požadovaný počet uzlů,
- všechny uzly v bloku jsou přímo napojené přes vstupy a výstupy,
- všechny uzly v bloku jsou aktivní.

Následně je vybrán v každém jedinci jeden náhodný blok. Tyto bloky jsou pak prohozeny. Pozice jednotlivých uzlů v blocích se mohou v rámci nového jedince změnit, vzájemné zapojení uzlů v bloku ale zůstává nezměněno. Pokud alespoň jeden z rodičů neobsahuje žádný blok splňující tato pravidla, křížení se neprovede. Tato metoda křížení má lineární složitost, jelikož je prováděna spolu s evaluací aktivní cesty genotypu.

Kalkreuth a Husa se v jejich implementaci rozhodli pro bloky složené ze tří uzlů. Porovnání s $1 + \lambda$ probíhalo přes různé kombinace parametrů na úlohách symbolické regrese a booleovských funkcí. Nebyla však nalezena žádná kombinace parametrů, která by byla lepší na všech problémech.

Diskrétní fenotypická rekombinace

Diskrétní fenotypická rekombinace byla navržena Kalkreuthem v roce 2022 [6]. Adaptoval diskrétní rekombinaci výměnou funkčních genů v aktivních uzlech. Po výběru dvou rodičů se zjistí maximální a minimální počet aktivních uzlů, které každý z těchto jedinců obsahuje. Důvodem je, že délka fenotypu (a tedy i aktivních uzlů) se může lišit od jedince k jedinci. Tato metoda křížení pak iteruje nad minimálním množství aktivních uzlů. U každého z nich je učiněno rozhodnutí, zda se funkční geny vymění či ne. V případě, že se délka fenotypu liší, kratší z fenotypů se uměle prodlouží tak, aby měl požadovanou délku a výměna mohla pokračovat.

Tento způsob křížení, zkombinovaný s turnajovou selekcí, byl následně testován na úlohách symbolické regrese oproti klasickému algoritmu $1 + \lambda$. V této oblasti se ukázalo, že navržené křížení vedlo menšímu množství fitness evaluací a k více úspěchům u komplexnějších úloh.

Kapitola 3

Návrh

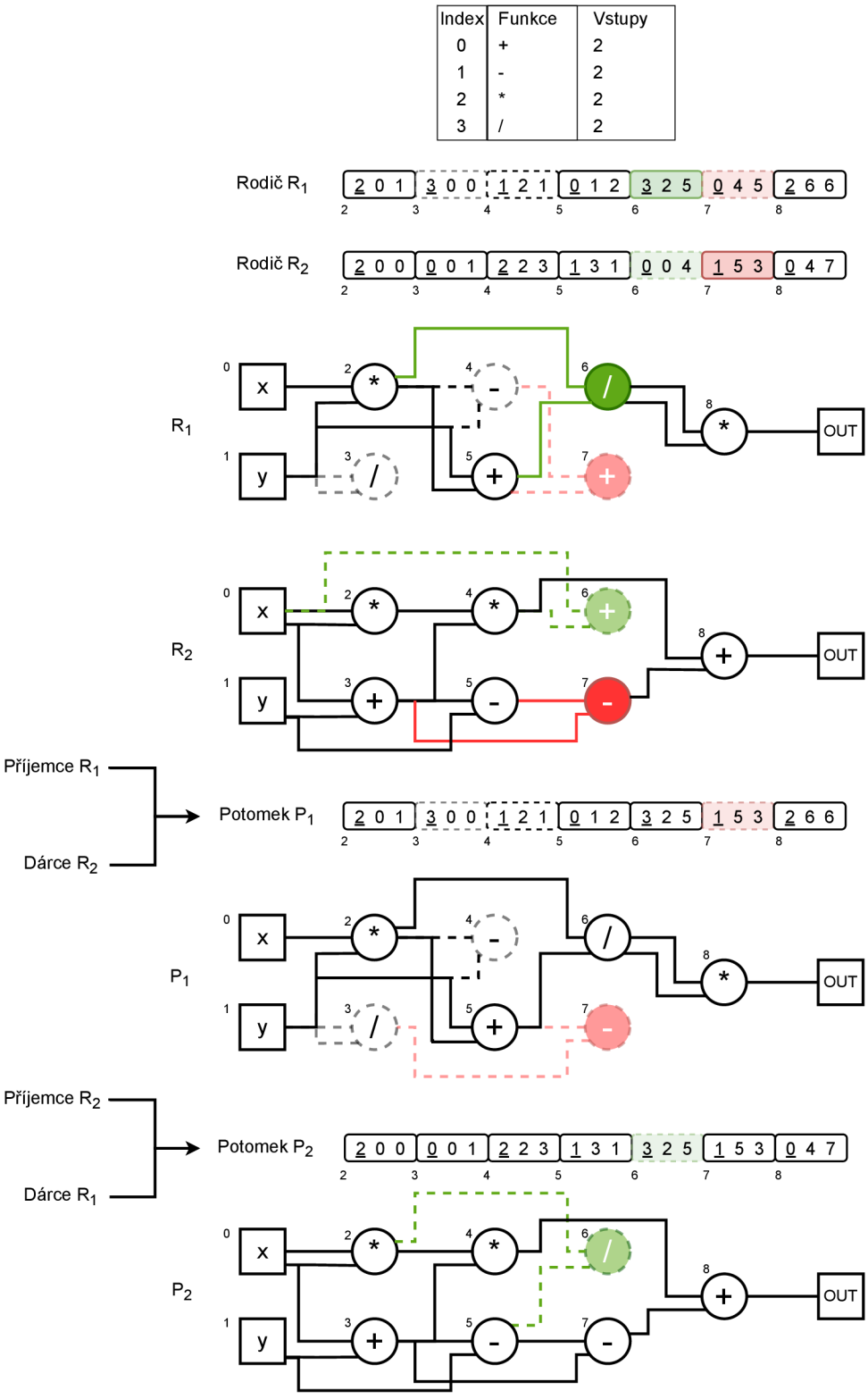
V rámci této práce jsem navrhl dvě nové metody křížení v CGP. Implantaci aktivních genů do neaktivních (IAGdN) a výměnu podgrafů náhodnou cestou (VPNC). Většina metod křížení, se kterými jsem se seznámil, kříží jedince na různých částech genotypu. V rámci navržených variant jsem se ale rozhodl prozkoumat možnost, kde křížení probíhá, či minimálně začíná, na stejném místě v obou jedincích. Touto cestou jsem se vydal, abych omezil zbytečné narušení fenotypu při křížení.

3.1 Implantace aktivních genů do neaktivních

Tato metoda křížení nemění přímo fenotyp jedince. Rodičům jsou přiřazeny role dárce a příjemce, které se při vytváření potomků střídají. Začíná výběrem dvou jedinců a vytvořením seznamu pozic uzlů, kde jeden rodič má uzel aktivní a druhý rodič má uzel neaktivní. Pokud takováto pozice neexistuje, křížení se neprovede. Pokud jich existuje více, náhodně se vybere jedna nebo více z nich podle parametru pro míru křížení. Následně se implantuje aktivní uzel dárce do neaktivního uzlu příjemce. Příklad možného výsledku tohoto křížení lze vidět na obrázku 3.1.

Při návrhu jsem se inspiroval diskrétním křížením. Myšlenkou za tímto algoritmem je, že mutace stále odvádí většinu práce. Křížení tady slouží pouze jako možné postrčení určitým směrem, pokud se mutace rozhodne uzel zahrnout do aktivní cesty. Původně jsem zamýšlel tento algoritmus zkombinovat s $1 + \lambda$, kde mutace může dělat výrazné skoky díky genetickému driftu. Nakonec jsem se ale rozhodl použít turnajovou selekci pro větší diverzitu rodičů.

Potenciální problém tohoto algoritmu vidím v tom, že tento algoritmus křížení může zamořit neaktivní uzly opakujícími se funkcemi. Je to však pouze spekulace, kterou by bylo třeba experimentálně ověřit.



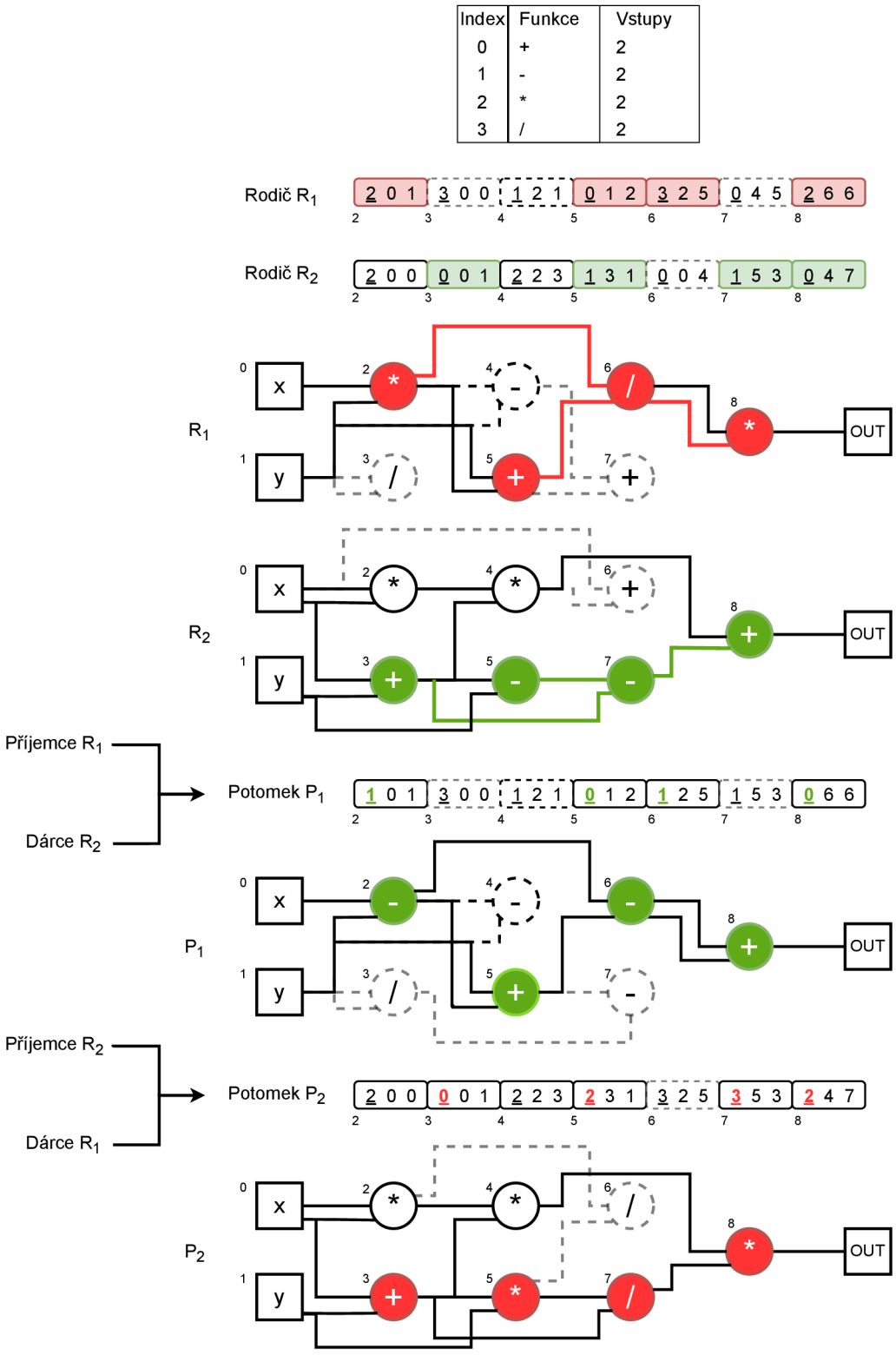
Obrázek 3.1: Příklad aplikování IAGdN křížení. V rodičích R_1 a R_2 je více dvojic uzlů splňující podmínky pro křížení. Z těchto dvojic byly náhodně vybrány uzly s indexem 6 a 7 . Následně se geny neaktivních uzlů na těchto indexech nahradily geny aktivních uzlů druhého rodiče. Tímto způsobem se vygenerovali potomci P_1 a P_2.

3.2 Výměna podgrafů náhodnou cestou

Výměna podgrafů náhodnou cestou na rozdíl od předchozího algoritmu v sekci 3.1 fenotyp mění. Rodiče jsou rozděleni na dárce a příjemce, tyto role se pak prohazují pro každého nově generovaného potomka. Algoritmus opět vytvoří seznam uzlů se stejným indexem, u kterých platí určité pravidlo. Tentokrát je pravidlem, že musí být oba uzly aktivní. Minimálně jedna takováto dvojice se najde vždy, jsou to výstupní uzly. Na vstupních uzlech není co měnit, proto nejsou nahrazovány. Jedna z těchto dvojic je náhodně vybrána, následně se funkční gen z uzlu dárce zkopíruje do uzlu příjemce. Pro každý vstup, který používají uzly obou jedinců, se následně s určitou pravděpodobností rozhodne, zda se bude pokračovat dále. Pokud se rozhodne že ano, pak jsou další dvojicí pro tento algoritmus ty uzly, které byly na daném vstupu. Pokud je na vstupu vstupní uzel, pak se tento vstup přeskočí. Algoritmus končí pokud ve všech cestách narazí na vstupní uzly nebo pokud se na žádném dalším uzlu rozhodne pokračovat. Příklad možného křížení lze najít na obrázku 3.2.

Při návrhu tohoto algoritmu jsem se inspiroval diskrétním fenotypickým křížením zmíněným v sekci 2.6. Chtěl jsem metodu rozšířit o výměnu podgrafů, které spolu souvisí. Zároveň jsem chtěl využít toho, že napojení uzlů zůstává nezměněno. Toto křížení jsem se rozhodl zkombinovat s turnajovou selekcí stejně jako v diskrétním fenotypickém křížení [6]. Původní návrh této metody křížení neobsahoval role příjemce a dárce, výměna probíhala současně v obou jedincích. Pro možnost větší flexibility v množství generovaných potomků (původní verze generovala pouze sudý počet potomků) jsem se rozhodl tuto část změnit na úkor rychlosti.

Je více možností jak náhodně rozhodovat o tom, kterou cestou se křížení vydá. Zvolil jsem základní možnost rovnoměrného rozdělení u každého zpracovávaného uzlu. Během návrhu jsem ale zvažoval i jiné varianty, které by mohly být vhodné. Nabízí se například použít rozdělení jiné. Zajímavou možností by také bylo zůstat u rovnoměrného rozdělení s určitou úpravou. S každým zanořením algoritmu by se pravděpodobnost na pokračování dalším uzlem postupně snižovala z počáteční pravděpodobnosti 1,0. Tímto způsobem by se spíše vyměňovaly podgrafy podobné velikosti, ale šance, že se vymění pouze samostatný uzel, by byla nulová.



Obrázek 3.2: U rodičů R_1 a R_2 se náhodně vybrala dvojice na indexu 8 z ostatních dvojic aktivních uzlů. Následně se algoritmus vydal náhodnou cestou přes vstupy uzlů. Při generování potomka P_1 byl dárcem R_2 a příjemcem R_2. Při generování potomka P_2 byly role prohozeny. Pro přehlednost je náhodná cesta při vytváření potomků stejná.

Kapitola 4

Implementace CGP

Pro zhodnocení svých návrhů jsem se rozhodl implementovat vlastní CGP, místo přebírání cizího kódu a dodatečného přidávání křížení. Toto rozhodnutí jsem udělal nejen proto, abych měl větší kontrolu nad průběhem algoritmu, ale také abych se více seznámil s vnitřním fungováním CGP. Při implementaci jsem částečně vycházel z pseudokódů uvedených v [14].

Jako programovací jazyk jsem zvolil Python, jelikož umožňuje rychlý vývoj prototypů. I když jsou programy v Pythonu obecně pomalejší než například programy napsané v C, tento problém lze částečně vykompenzovat vhodným použitím knihoven jako je `numpy`[1], pro urychlení časově náročných výpočtů.

Implementace byla mířená na experimenty se symbolickou regresí. Změna některých parametrů potřebná pro řešení jiných typů úloh by mohla tedy vyžadovat úpravy v implementaci.

Každý jedinec obsahoval jeden chromozom, termín chromozom/jedinec se tedy v tomto případě dá zaměnit. Chromozom byl implementován jako pole uzlů. Každý uzel pak byl polem celých čísel, které sloužily jako ukazatele na jiné uzly, nebo do tabulky funkcí.

4.1 Automatické testy

Jako testovací framework jsem zvolil `unittest`[2]. Automatické testy jsem psal v průběhu implementace spolu s přidáváním nových funkcí.

Unit testy pro jednotlivé funkce byly využity pro ověření správnosti jednotlivých částí programu. Tyto typy testů zahrnovaly například zda jsou zmutované uzly vždy validní nebo jestli se správně počítá výstup z jedince a hodnota fitness.

Integrační testy jsem pak použil později při ověřování, zda je program funkční. Tato kontrola například zahrnovala, zda program najde během kratší doby (většinou 10^4 fitness evaluací) řešení jednoduchého problému, jako třeba $(x + x) \times x$. Po doběhnutí programu bylo zkontrolováno, zda našel řešení a zda je řešení opravdu validní. Tedy jestli řeší daný problém ve správných mezích a má správnou fitness hodnotu. Tento postup se opakoval pro algoritmy $1 + \lambda$ i turnajové selekce s mými metodami křížení. U každého z těchto algoritmů se testoval jak program s jedním, tak více vstupy různých číselných typů.

[1] Knihovna umožňující rychlé numerické výpočty s poli https://numpy.org

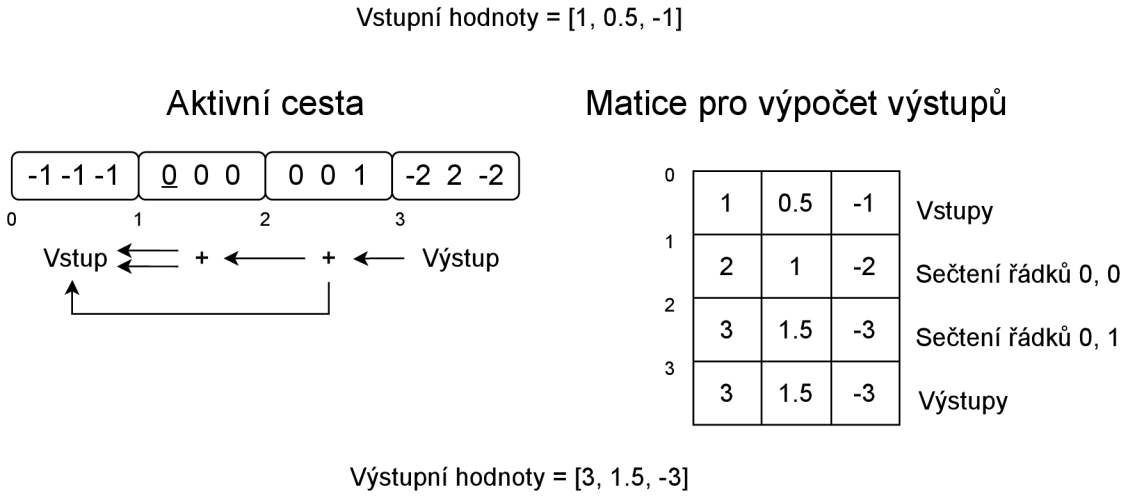
[2] Vestavěná knihovna Pythonu určená pro testování https://docs.python.org/3/library/unittest.html

4.2 Reprezentace populace

I když jsem většinu kódu psal ve funkcích, implementoval jsem jednu třídu pro populaci. Objektový přístup jsem zde zvolil proto, abych lépe uchovával informace, které jsou třeba ve více funkcích. Dalším důvodem bylo zamezení duplicity a lepší přehlednost kódu. Třída populace vytváří první populaci náhodně. Další jedince již nevytváří, pouze je ukládá či vrací. Zároveň také uchovává jejich aktivní cestu, fitness hodnotu a má přehled o tom, který jedinec je potomek a který rodič. Při ukládání nové generace se pak nevyhodnocuje fitness jedince, pokud již je známá. Toto platí zejména pro rodiče, kteří vždy pochází minimálně z předchozí generace a jejich fitness se tak nemusí vyhodnocovat znovu. Toto vede ke značnému zrychlení programu, jelikož vyhodnocování fitness je časově náročná operace.

4.3 Vyhodnocování fitness hodnoty

Vyhodnocování fitness hodnoty sestává ze tří základních funkcí: získání aktivní cesty, získání výstupu této aktivní cesty a výpočtu hodnoty fitness. Získání aktivní cesty jsem implementoval podle pseudokódu v [14]. Algoritmus nejprve označí výstupní uzly jako aktivní. Následně prochází přes jejich vstupy do dalších aktivních uzlů, kde postupuje stejně. Tento postup se opakuje, dokud nezbývají pouze primární vstupy programu. Výpočet výstupu kandidátního CGP programu pro zadaný vstup zahrnuje velké množství výpočtů. Proto je zde použita knihovna `numpy`. Vstupní data jsou předány jako dvourozměrně `numpy` pole. Každý řádek zde obsahuje hodnoty pro jednu nezávislou proměnnou. Každý sloupec představuje výpočet výstupu pro jednu hodnotu nezávislé proměnné. Tyto hodnoty jsou používány vstupními uzly jedince. Pro výpočet výstupních hodnot kandidátního řešení se vytvoří `numpy` matice. Každý řádek této matice znázorňuje jeden uzel. Na prvních řádcích jsou vstupy, pak funkční uzly a nakonec výstup. Každý řádek je díky `numpy` operacím nad poli počítán jako celek. Tím pádem se zjistí výstup jedince pro všechny vstupy najednou. Pro lepší představu je jednoduchý příklad na obrázku 4.1.



Obrázek 4.1: Příklad výpočtu výstupu z aktivní cesty. Každý sloupec matice představuje výpočet výstupu pro jednu hodnotu nezávislé proměnné. Řádky pak značí jeden uzel v aktivní cestě.

Uzly v aktivní cestě samozřejmě nejdou vždy za sebou podle indexu, proto probíhá mapování indexu uzlu na index řádku v matici. Vstupy operací jsou omezeny, aby se zamezilo přetékání a nevalidním hodnotám. Pokud alespoň jeden z operandů nesplňoval některou z podmínek, operace vrátila nulu. Operace a jejich podmínky byly následující:

- $x + y : x, y \in \langle -1e30, 1e30 \rangle$,
- $x - y : x, y \in \langle -1e30, 1e30 \rangle$,
- $x \times y : x, y \in \langle -1e30, 1e30 \rangle$,
- $x \mod y : y \neq 0$,
- $\sin x$: bez omezení,
- $\cos x$: bez omezení,
- $e^x : x \leq 50$,
- $\ln |x| : x \neq 0$

Pro výpočet fitness funkce se pak používá sumu absolutní hodnoty rozdílu mezi očekávaným výstupem a výstupem jedince pro stejné vstupy. Postup lze tedy popsat jako:

$$fit = \sum_{i=0}^{n} |x_i - y_i|, \tag{4.1}$$

kde n je počet výstupů, x_i je očekávaný výstup a y_i je reálný výstup jedince.

Řešení se považuje za dostatečně dobré, pokud se každý výstup jedince liší maximálně o ε od správného řešení. ε je implicitně nastavená na $0,01$. Původně se fitness počítala střední kvadratickou chybou. Změnu jsem provedl po zhodnocení práce jiných autorů, kteří se zabývali symbolickou regresí v CGP. Podobně byla fitness vyhodnocována například v [6] nebo jiných publikacích zmíněných v [15].

4.4 Varianty CGP a křížení

Implementoval jsem tři varianty CGP. První varianta využívala $1 + \lambda$ pro čistě mutací řízený běh. Zbylé varianty používaly turnajovou selekci, lišily se ale v použité metodě křížení. Při psaní $1 + \lambda$ jsem se inspiroval pseudokódy v [14]. U druhé a třetí varianty jsem použil deterministickou turnajovou selekci se dvěma turnaji pro výběr dvou rodičů. Každý z jedinců mohl být součástí pouze jednoho turnaje, bez možnosti navrácení. Vítězem turnaje se pak stal jedinec s nejnižší fitness hodnotou. Bylo tedy zajištěno, že nejlepší jedinec v populaci bude vždy jedním z nových rodičů.

Obě metody mají mimo jiné parametr pro míru křížení, který se pohybuje v rozsahu $\langle 0, 1 \rangle$. U implantace aktivních genů do neaktivních tento parametr udává podíl dvojic uzlů splňující potřebné podmínky, u kterých se provede implantace. U výměny podgrafů náhodnou cestou pak tento parametr říká, jaká je u každého vstupu šance, že metoda bude pokračovat dále.

Výměna podgrafů náhodnou cestou je pak implementována rekurzivně spolu s polem příznaků, které udávají, zda se již funkce na daném uzlu provedla.

4.5 Spouštění běhu a experimentů

Jeden běh programu se spouští voláním funkce `evolve`. Její vstupy a výstupy jsou vypsané v tabulce 4.1.

Vstupy	**Výstupy**
Velikost populace (včetně rodiče)	Nejlepší nalezené řešení
Počet sloupců v genotypu	Fitness nejlepšího řešení
Počet řádků v genotypu	Počet uplynulých generací
Dvourozměrné pole vstupů	Počet provedených fitness evaluací
Pole správných výstupů	Slovník nejlepších fitness podle generace
Přijatelnou hranici fitness, neboli ε	Bool – nalezeno dostačující řešení
Maximální počet fitness evaluací	
Míra mutace – $\langle 0, 1 \rangle$	
Seed pro generování náhodných čísel	
Typ algoritmu který se má provést	
Míra křížení – $\langle 0, 1 \rangle$	

Tabulka 4.1: Vstupy a výstupy funkce `evolve`, která spouští jeden běh vybraného algoritmu dle zadaných parametrů.

Do zmíněného slovníku fitness hodnot podle generace se přidávají hodnoty pouze tehdy, kdy je nalezena nová lepší fitness hodnota.

Experimenty lze spustit skriptem `experiment.py`. Tento skript používá již předem připravené hodnoty pro funkce zmíněné v sekci 5. Parametry pro spouštění experimentů jsou globální proměnné na začátku souboru, které lze upravit podle potřeby (počet běhů na funkci, spuštění samostatné funkce apod.). Tento skript spustí všechny běhy paralelně pomocí knihovny `multiprocessing`[3]. Každému běhu je přiřazen unikátní identifikátor ve formě řetězce knihovnou `uuid`[4]. Tento identifikátor slouží pro rozpoznání jednotlivých běhů a zároveň k vygenerování seedu pro generátor náhodných čísel. Tyto seedy jsou potřebné, jelikož jinak dědí všechny běhy stejný seed z rodičovského procesu.

Po dokončení každého běhu se zapíše jeho výstup do dvou souborů: `data-general.csv` a `data-run-details.csv`. V `data-general.csv` lze najít obecné informace o jednotlivých bězích. Zásadní je počet fitness evaluací, typ algoritmu a úloha. V `data-run-details.csv` jsou pak pro každý běh detaily průběhu fitness podle generací.

4.6 Dokumentace

Dokumentace se generuje automaticky spuštěním skriptu `generate-docs.sh`. Využil jsem knihovnu `pdoc`[5], který vytváří dokumentaci na základě docstringů každé funkce. Každý modul má svou vlastní HTML stránku. Ke každé funkci a metodě lze najít krátký popis, vstupy, výstupy a případně chyby, které může vyhodit.

[3]Vestavěná knihovna jazyka Python umožňující paralelismus https://docs.python.org/3/library/multiprocessing.html

[4]Knihovna pro generování unikátních identifikátorů https://docs.python.org/3/library/uuid.html

[5]Knihovna generující dokumentaci z docstringů https://pdoc.dev

Kapitola 5

Experimenty

Nové metody křížení byly porovnávány oproti základní variantě využívající genetický operátor mutace a prohledávací algoritmus $1 + 4$, tedy často používanou variantu $1 + \lambda$. Metody byly porovnány na pěti úlohách symbolické regrese. Tyto úlohy spolu s jejich datovými sadami vycházejí z doporučení zmíněných v [12]. Tyto úlohy vypadaly následovně:

Název	**Objektivní funkce**	**Vstupní hodnoty**	**Množina funkcí**	**Konstanty**
Koza 3	$x^6 - 2x^4 + x^2$	U[-1, 1, 20]	$+, -, \times,$ mod , sin, cos, e^n, ln $\lvert n \rvert$	Žádné
Nguyen 6	$\sin(x) + \sin(x + x^2)$	U[-1, 1, 20]	$+, -, \times,$ mod , sin, cos, e^n, ln $\lvert n \rvert$	Žádné
Nguyen 8	$\sqrt{x}$	U[0, 4, 20]	$+, -, \times,$ mod , sin, cos, e^n, ln $\lvert n \rvert$	Žádné
Nguyen 9	$\sin(x) + \sin(y^2)$	U[-1, 1, 100]	$+, -, \times,$ mod , sin, cos, e^n, ln $\lvert n \rvert$	Žádné
Nguyen 10	$2\sin(x)\cos(y)$	U[-1, 1, 100]	$+, -, \times,$ mod , sin, cos, e^n, ln $\lvert n \rvert$	Žádné

Tabulka 5.1: Tabulka problémů symbolické regrese použitých v experimentech. U$[a, b, c]$ zde představuje uniformní náhodné hodnoty s krokem c, v intervalu od a do b včetně.

Ukončující podmínkou běhu byl počet evaluací fitness. Běh se ukončil, pokud tento počet dosáhl 10^7. Toto rozhodnutí jsem udělal na základě výsledků publikace Kalkreutha z roku 2022 o diskrétní fenotypické rekombinaci [6]. V této publikaci sice bylo omezení nastaveno na 10^8 fitness evaluací, ale běhy funkcí, které s touto publikací sdílím, ve většině případů doběhly podstatně dříve. Jejich výsledky jsem však využil pouze orientačně, jelikož použité CGP parametry se neshodovaly s parametry použitými v této práci.

Pro každý z těchto problému symbolické regrese bylo spuštěno 90 běhů, tedy 30 pro každou variantu CGP. Parametry pro jednotlivé varianty CGP řešící problém symbolické regrese jsou vypsány v tabulce 5.2.

Problém	**Algoritmus**	N_c	N_r	λ	M [%]	K [%]	R	P
Koza 3	$1 + \lambda$	150	1	4	7	50	-	-
	IAGdN	150	1	-	7	50	2	2
	VPNC	150	1	-	7	50	2	2
Nguyen 6	$1 + \lambda$	100	1	4	2	50	-	-
	IAGdN	100	1	-	2	50	2	2
	VPNC	100	1	-	2	50	2	2
Nguyen 8	$1 + \lambda$	150	1	4	15	50	-	-
	IAGdN	150	1	-	15	50	2	2
	VPNC	150	1	-	15	50	2	2
Nguyen 9	$1 + \lambda$	150	2	4	15	50	-	-
	IAGdN	150	2	-	15	50	2	2
	VPNC	150	2	-	15	50	2	2
Nguyen 10	$1 + \lambda$	60	2	4	20	50	-	-
	IAGdN	60	2	-	20	50	2	2
	VPNC	60	2	-	20	50	2	2

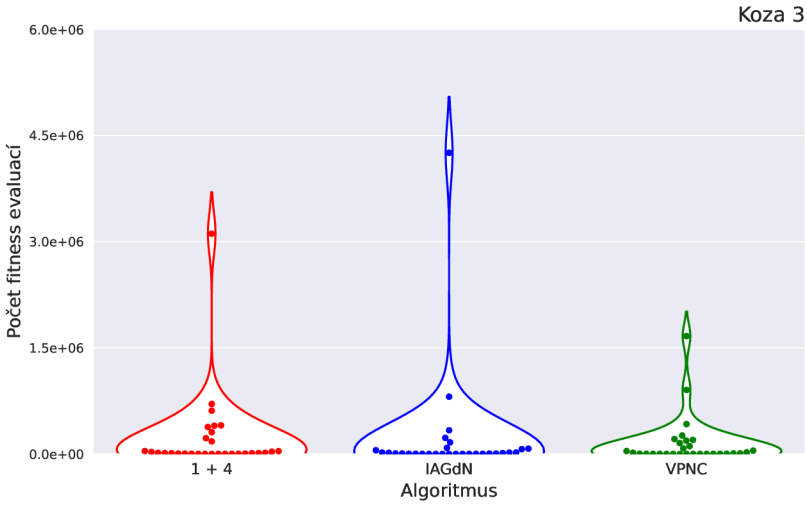
Tabulka 5.2: Parametry použité při spouštění experimentů. N_r a N_c jsou počet řádků a počet sloupců. M a K značí míru mutace a míru křížení v procentech. R a P je pak počet rodičů a počet potomků.

Parametry N_c a M byly nastaveny podle studie [8] od Kaufmanna a Kalkreutha. Přesněji pak podle sekce hodnotící parametry CGP s $1 + 4$ pro řešení úloh symbolické regrese. Parametr *l-back* byl implicitně nastaven na ∞. Jelikož nové metody křížení neprošly meta-optimalizací a kopírují většinu parametrů optimalizovaných pro běhy čistě s mutací, experimenty jsou výhodněji nastavené pro běhy CGP s $1 + 4$.

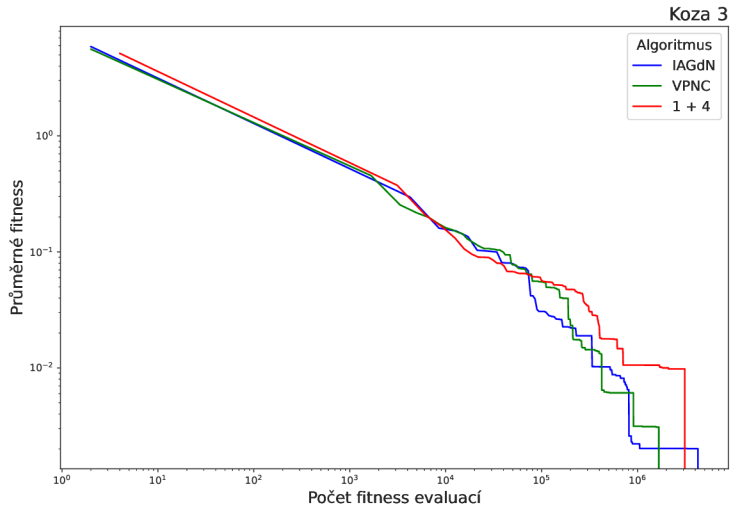
Jelikož nelze zaručit normální rozdělení vzorků, pro určení statistické významnosti byl použit jednostranný Mann-Whitneyův U test. Přesněji se u obou navržených metod křížení testovala nulová hypotéza, že počet evaluací potřebných k nalezení řešení je stochasticky menší než u CGP s $1 + 4$. Jako signifikantní rozdíl jsem bral p-hodnotu menší nebo rovno 0,05.

5.1 Koza 3

Pro problém Koza 3 bylo nalezeno dostatečně dobré řešení ve všech bězích, žádný z nich tedy nedosáhl hranice 10^7 fitness evaluací. Běhy využívající IAGdN a VPNC si vedly o něco lépe než standardní CGP s $1 + 4$. VPNC mělo nejlepší průměrný počet evaluací na nalezení řešení (145 607). IAGdN často nalezlo řešení za krátký čas, celkový průměr mělo ale 206 904 fitness evaluací, tedy horší než VPNC a blízko $1 + 4$, které potřebovalo v průměru 219 363 fitness evaluací. Výsledky jsou graficky znázorněny na obrázcích 5.1 a 5.2. Výsledky u této úlohy ale nejsou dostatečně statisticky signifikantní.



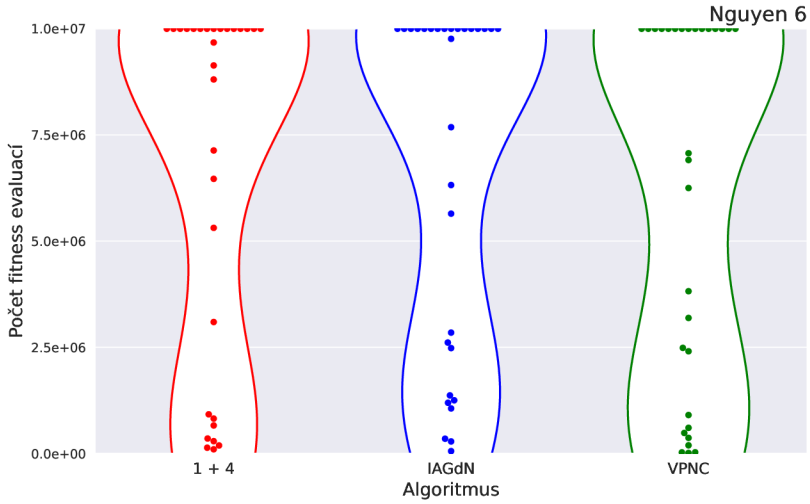
Obrázek 5.1: Počet fitness evaluací potřebných na vyřešení problému Koza 3 znázorněny houslovými diagramy. Jednotlivé běhy jsou zde vyznačeny jako body.



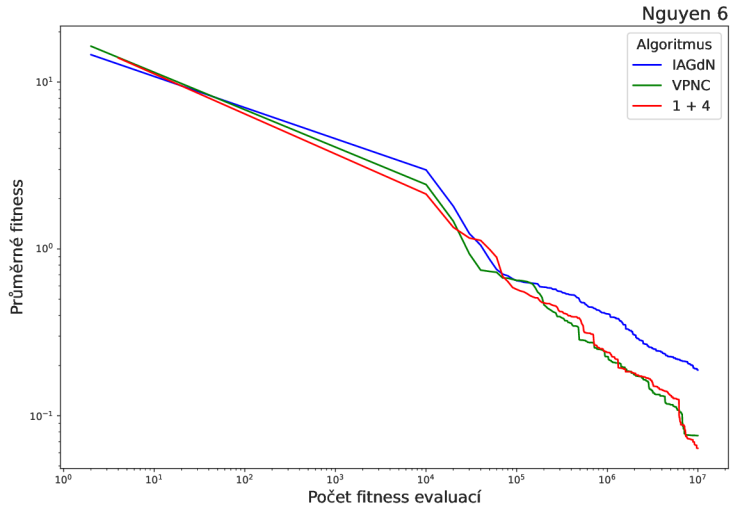
Obrázek 5.2: Graf zobrazující průběh jednotlivých variant CGP na problému Koza 3. Osy x a y jsou logaritmické. Osa y představuje fitness, kterou měl nejlepší jedinec v daném počtu fitness evaluací. Tato fitness je průměrem všech běhů.

5.2 Nguyen 6

Problém Nguyen 6 byl oproti ostatním úlohám nejnáročnější. Jak lze vidět na obrázku 5.3, přibližně polovina úloh nenašla řešení před dosažením 10^7 fitness evaluací. VPNC i 1 + 4 našli řešení přesně v 50 % běhů, IAGdN pak ve 47 % běhů. Všechny testované algoritmy si zde vedly podobně, v průměru potřebovaly něco přes 6 000 000 fitness evaluací na nalezení řešení. IAGdN si zde vedlo o něco hůře než ostatní v míře úspěchu i v jednotlivých kvartilech.



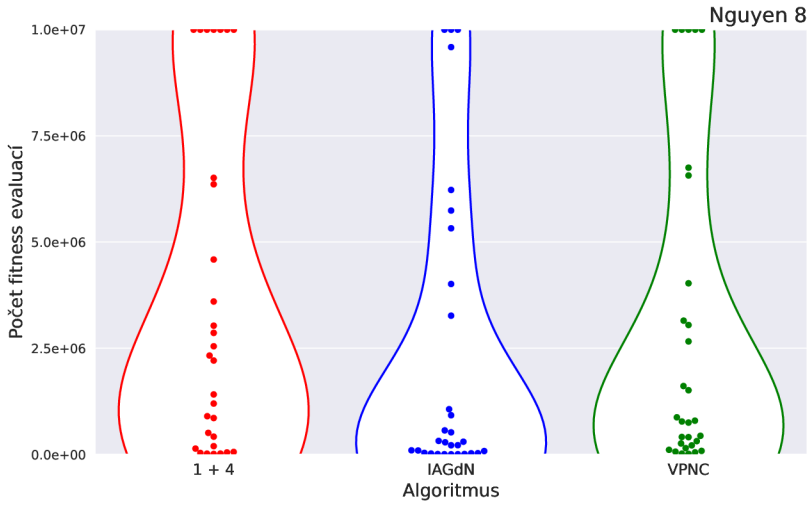
Obrázek 5.3: Počet fitness evaluací potřebných na vyřešení problému Nguyen 6 znázorněný houslovými diagramy. Jednotlivé běhy jsou zde vyznačeny jako body.



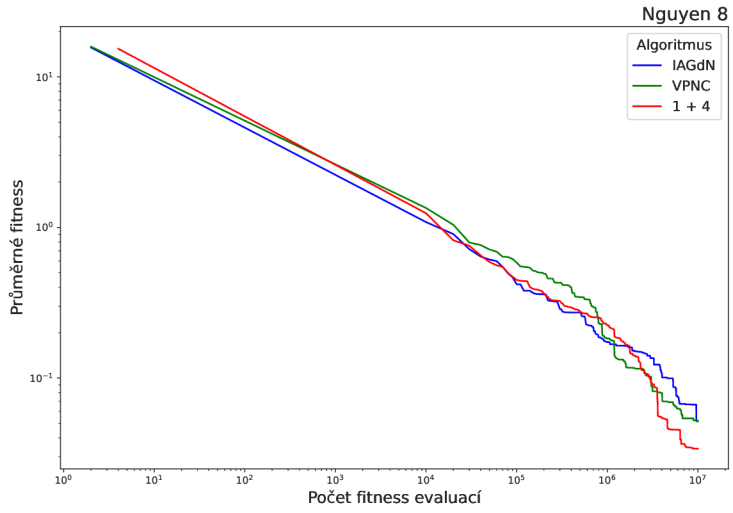
Obrázek 5.4: Graf zobrazující průběh jednotlivých variant CGP na problému Nguyen 6. Osy x a y jsou logaritmické. Osa y představuje fitness, kterou měl nejlepší jedinec v daném počtu fitness evaluací. Tato fitness je průměrem všech běhů.

5.3 Nguyen 8

U problému Nguyen 8 nové metody křížení předčily standardní 1 + 4 v počtu fitness evaluací potřebných k dosažení řešení, jak lze vidět na obrázku 5.5. IAGdN dokonce se statistickou významností, tedy s p-hodnotou 0,028 v porovnání s 1 + 4. Na grafu 5.6 jde ale vidět, že 1 + 4 měla u běhů které dosáhly hranice 10^7 fitness evaluací lepší průměrnou fitness než VPNC i IAGdN. Je tedy možné, že při větším limitu fitness evaluací by odlehlé běhy 1 + 4 našly řešení dříve než VPNC a IAGdN.



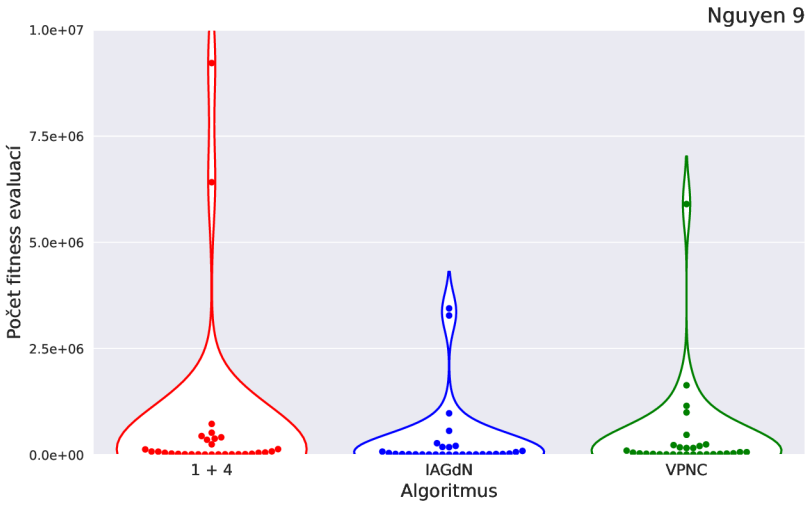
Obrázek 5.5: Počet fitness evaluací potřebných na vyřešení problému Nguyen 8 znázorněný houslovými diagramy. Jednotlivé běhy jsou zde vyznačeny jako body.



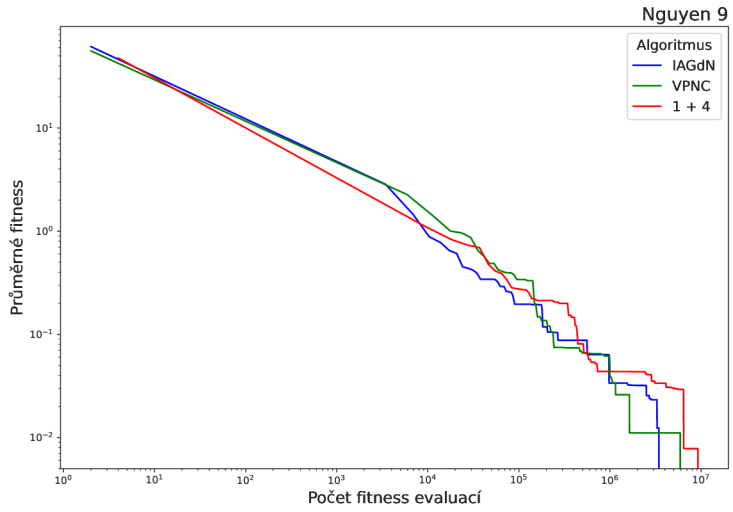
Obrázek 5.6: Graf zobrazující průběh jednotlivých variant CGP na problému Nguyen 8. Osy x a y jsou logaritmické. Osa y představuje fitness, kterou měl nejlepší jedinec v daném počtu fitness evaluací. Tato fitness je průměrem všech běhů.

5.4 Nguyen 9

U úlohy Nguyen 9 IAGdP předčilo 1 + 4 v počtu fitness evaluací potřebných k nalezení řešení s p-hodnotou 0,077, tedy téměř na významné úrovni. Všechny metody našly řešení v daném limitu, jak lze vidět na obrázcích 5.7 a 5.8. Nejvyšší průměrný počet fitness evaluací potřebný k nalezení řešení mělo 1 + 4 (1 992 619). Jeho průměr byl přibližně dvakrát vyšší než průměr nově navržených metod (852 120 pro IAGdN a 1 108 969 pro VPNC).



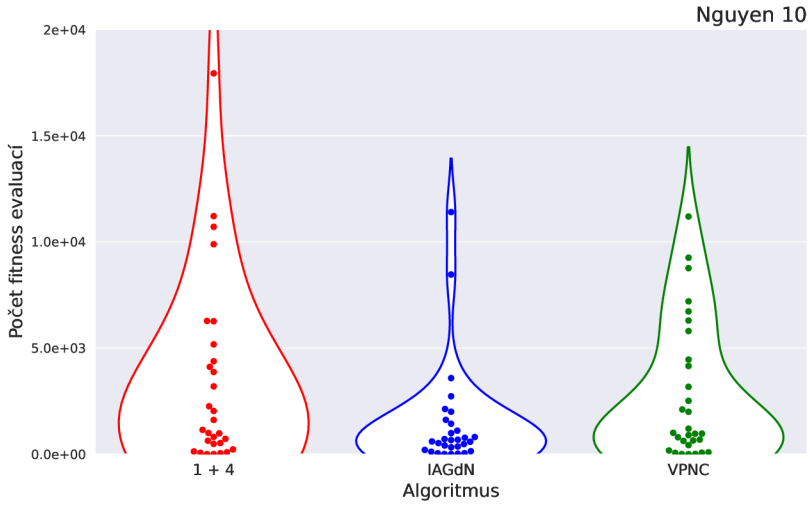
Obrázek 5.7: Počet fitness evaluací potřebných na vyřešení problému Nguyen 9 znázorněný houslovými diagramy. Jednotlivé běhy jsou zde vyznačeny jako body.



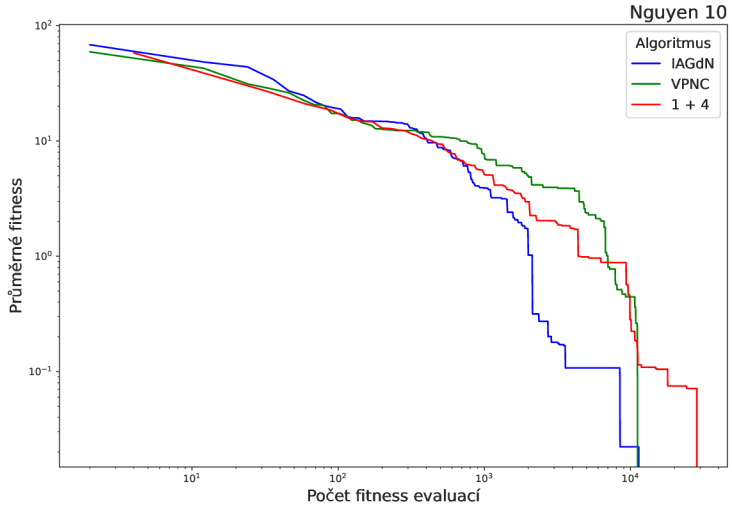
Obrázek 5.8: Graf zobrazující průběh jednotlivých variant CGP na problému Nguyen 9. Osy x a y jsou logaritmické. Osa y představuje fitness, kterou měl nejlepší jedinec v daném počtu fitness evaluací. Tato fitness je průměrem všech běhů.

5.5 Nguyen 10

Nguyen 10 byl ze zkoumaných problémů nejjednodušší, všechny běhy našly řešení před dosažením 2×10^4 fitness evaluací. Na této úloze dosáhlo IAGdN významně lepších výsledků, než $1 + 4$. IAGdN potřebovalo v průměru 2 492 fitness evaluací pro nalezení řešení, oproti $1 + 4$, která v průměru potřebovala 6 268. Rozdíl mezi jednotlivými algoritmy lze vidět na obrázcích 5.9 a 5.10.



Obrázek 5.9: Počet fitness evaluací potřebných na vyřešení problému Nguyen 10 znázorněný houslovými diagramy. Jednotlivé běhy jsou zde vyznačeny jako body.



Obrázek 5.10: Graf zobrazující průběh jednotlivých variant CGP na problému Nguyen 10. Osy x a y jsou logaritmické. Osa y představuje fitness, kterou měl nejlepší jedinec v daném počtu fitness evaluací. Tato fitness je průměrem všech běhů.

5.6 Zhodnocení výsledků experimentů

Celkové výsledky experimentů jsou vypsány v tabulce 5.3. Ve třech z pěti úloh se řešení podařilo najít vždy. Řešení pro problém Nguyen 6 našly metody VPNC a 1 + 4 s mírou úspěchu 0,5, IAGdN pak s mírou 0,47 (rozdíl byl pouze v jednom z běhů). U problému Nguyen 8 pak na tom s mírou úspěchu byla nejhůře 1 + 4 ($MU = 0,77$), po ní VPNC ($MU = 0,83$) a v nejvíce případech našla řešení metoda IAGdP ($MU = 0,9$).

Rozdíly v průměrném počtu fitness evaluacích potřebných k nalezení řešení byly u většiny úloh minimální, s výjimkou úloh Nguyen 9 a Nguyen 10. V obou případech měla nejmenší průměr metoda IAGdN. Na vyřešení problému Nguyen 10 potřebovala metoda IAGdN v průměru téměř třikrát méně evaluací a u problému Nguyen 9 přibližně dvakrát méně fitness evaluací než CGP s algoritmem 1 + 4. Jedna z navržených metod (IAGdN) dokázala v úlohách Nguyen 8 a Nguyen 10 najít řešení signifikantně dříve, než standardní CGP. Navržené metody křížení ale celkově potřebovaly podobný nebo pouze o trochu menší počet fitness evaluací než standardní CGP s 1 + λ. Je však třeba brát v potaz, že parametry, se kterými byly spouštěny nové metody křížení, pro ně nebyly optimalizované.

Problém	**Algo**	**P**	**MU**	**SO**	**1Q**	**2Q**	**3Q**	**p-hod**
Koza 3	1 + λ	219 363	1,0	580 373	3 710	14 548	212 217	-
	IAGdN	206 904	1,0	781 194	1 490	7 475	64 455	0,107
	VPNC	145 607	1,0	341 013	1 887	8 492	143 923	0,171
Nguyen 6	1 + λ	6 769 443	0,5	4 204 389	1 463 809	9 838 476	10 000 000	-
	IAGdN	6 763 320	0,47	4 073 183	2 512 791	10 000 000	10 000 000	0,618
	VPNC	6 157 796	0,5	3 958 804	1 279 219	8 534 340	10 000 000	0,382
Nguyen 8	1 + λ	3 660 136	0,77	3 958 804	248 337	2 267 304	6 473 106	-
	IAGdN	2 297 414	0,9	3 539 683	28 849	290 020	3 824 594	0,028
	VPNC	2 833 029	0,83	3 707 186	221 840	782 570	3 807 797	0,257
Nguyen 9	1 + λ	647 511	1,0	1 992 619	8 601	46 482	322 594	-
	IAGdN	316 055	1,0	852 120	4 193	14 804	153 929	0,077
	VPNC	391 152	1,0	1 108 969	7 975	38 616	194 739	0,382
Nguyen 10	1 + λ	4 141	1,0	6 268	494	1 378	4 973	-
	IAGdN	1 431	1,0	2 492	231	631	1 350	0,023
	VPNC	2 741	1,0	3 225	475	987	4 378	0,338

Tabulka 5.3: Shrnutí výsledků získaných z experimentů. Algo je zkratka pro algoritmus. P je průměrný počet fitness evaluací, po které trval běh algoritmu. MU je míra úspěchu (success rate), tedy jak často se podařilo najít řešení. SO je směrodatná odchylka počtu fitness evaluací. xQ značí kvartily počtu fitness evaluací. Pod p-hod jsou zapsány p-hodnoty pro nulovou hypotézu, že daný algoritmus potřebuje stochasticky méně fitness evaluací na nalezení řešení než 1 + λ.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat dvě metody křížení v kartézském genetickém programování a porovnat je s existujícím přístupem. Zároveň práce zahrnovala nastudování a popsání evolučních algoritmů, GP, CGP, problematiky křížení a symbolické regrese. Tyto metody jsem navrhl na základě předchozích publikací, ve kterých jsem viděl možnost zlepšení. Kromě této teorie bylo také třeba implementovat různé varianty CGP pro účely experimentů a tuto implementaci popsat. Následně jsem nově navržené metody porovnával se standardním přístupem na úlohách symbolické regrese a výsledky zhodnotil.

Celkově navržené metody dosáhly podobných výsledků jako standardní postup. Metoda křížení VPNC si vedla o trochu lépe než standardní $1 + 4$ v počtu fitness evaluací potřebných k nalezení řešení, nešlo však o významný rozdíl. Metoda IAGdN pak potřebovala v úloze Nguyen 6 o trochu více času než standardní $1 + 4$, ale v úlohách Nguyen 8 a Nguyen 10 předčila $1 + 4$, řešení našla s významně menším počtem fitness evaluací. Celkově výsledky naznačují, že IAGdN může být v určitých případech lepší volbou, než standardní $1 + 4$. Tyto experimenty však byly více nakloněny standardnímu přístupu.

Mnou navržené metody nabízejí spoustu možností, které by se daly v budoucnu prozkoumat. U výměny podgrafů náhodnou cestou je například možné vyzkoušet různé rozložení pravděpodobnosti. Také by bylo zajímavé porovnat, zda je výměna začínající na stejném indexu přínosná oproti výměně na náhodném indexu. Nabízí se i vyzkoušet tyto metody křížení v kombinaci s různými evolučními strategiemi. Práce pokračující ve zkoumání těchto metod by mohla najít optimální parametry a porovnat metody i na jiných problémech, například na návrhu elektrických obvodů.

Na konec bych rád řekl, že tato práce mi dala náhled do evolučních algoritmů, které mě fascinovaly již před začátkem mého studia informačních technologií. Právě tato fascinace byla nakonec jeden z důvodů, proč jsem si tento obor vybral.

Literatura

[1] Bartz Beielstein, T.; Branke, J.; Mehnen, J. a Mersmann, O. Evolutionary algorithms. *WIREs Data Mining and Knowledge Discovery*, Květen 2014, sv. 4, č. 3, s. 178–195.

[2] Clegg, J.; Walker, J. A. a Miller, J. F. A new crossover technique for Cartesian genetic programming. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2007, s. 1580–1587. GECCO '07. ISBN 978-1-595-93697-4. Dostupné z: https://doi.org/10.1145/1276958.1277276.

[3] Fogel L. J., W. M. J. Artificial intelligence through a simulation of evolution. In: *Biophysics and Cybernetic Systems*. Washington DC: Spartan books, 1965.

[4] Holland, J. H. Genetic Algorithms and the Optimal Allocation of Trials. *SIAM Journal on Computing*, 1973, sv. 2, č. 2, s. 88–105.

[5] Husa, J. a Kalkreuth, R. A Comparative Study on Crossover in Cartesian Genetic Programming. In: Castelli, M.; Sekanina, L.; Zhang, M.; Cagnoni, S. a García Sánchez, P., ed. *Genetic Programming*. Cham: Springer International Publishing, 2018, s. 203–219. ISBN 978-3-319-77553-1.

[6] Kalkreuth, R. Towards Discrete Phenotypic Recombination in Cartesian Genetic Programming. In: Rudolph, G.; Kononova, A. V.; Aguirre, H.; Kerschke, P.; Ochoa, G. et al., ed. *Parallel Problem Solving from Nature – PPSN XVII*. Cham: Springer International Publishing, 2022, s. 63–77. ISBN 978-3-031-14721-0.

[7] Kalkreuth, R.; Rudolph, G. a Droschinsky, A. A New Subgraph Crossover for Cartesian Genetic Programming. In: McDermott, J.; Castelli, M.; Sekanina, L.; Haasdijk, E. a García Sánchez, P., ed. *Genetic Programming*. Cham: Springer International Publishing, Duben 2017, s. 294–310. ISBN 978-3-319-55696-3.

[8] Kaufmann, P. a Kalkreuth, R. Parametrizing Cartesian Genetic Programming: An Empirical Study. In: Kern Isberner, G.; Fürnkranz, J. a Thimm, M., ed. *KI 2017: Advances in Artificial Intelligence*. Cham: Springer International Publishing, 2017, s. 316–322. ISBN 978-3-319-67190-1.

[9] Kaufmann, P. a Platzner, M. Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2008, s. 1219–1226. GECCO '08. ISBN 978-1-605-58130-9. Dostupné z: https://doi.org/10.1145/1389095.1389334.

[10] Koza, J. R. a Poli, R. Genetic Programming. In: Burke, E. K. a Kendall, G., ed. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Boston, MA: Springer US, 2005, s. 127–164. ISBN 978-0-387-28356-2. Dostupné z: https://doi.org/10.1007/0-387-28356-0_5.

[11] Langdon, W. B.; Poli, R.; McPhee, N. F. a Koza, J. R. Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications. In: Fulcher, J. a Jain, L. C., ed. *Computational Intelligence: A Compendium*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 927–1028. ISBN 978-3-540-78293-3. Dostupné z: https://doi.org/10.1007/978-3-540-78293-3_22.

[12] McDermott, J.; White, D. R.; Luke, S.; Manzoni, L.; Castelli, M. et al. Genetic programming needs better benchmarks. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2012, s. 791–798. GECCO '12. ISBN 978-1-450-31177-9. Dostupné z: https://doi.org/10.1145/2330163.2330273.

[13] Miller, J. F. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, s. 1135–1142. GECCO'99. ISBN 978-1-55860-611-1.

[14] Miller, J. F. *Cartesian Genetic Programming*. 2011. vyd. Springer International Publishing, červen 2003. ISBN 978-3-642-17309-7.

[15] Miller, J. F. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*. USA: Kluwer Academic Publishers, Červen 2020, sv. 21, 1–2, s. 129–168.

[16] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. 1971. Disertační práce. Technische Universität Berlin.

[17] Sarmento, R. a Costa, V. Introduction to Linear Regression. In: *Comparative Approaches to Using R and Python for Statistical Data Analysis*. Leden 2017, s. 111–115. ISBN 9781522519898.

[18] Sekanina, L. Evoluční návrh hardware. In: *Umelá inteligencia a kognitívna veda II.* Vydavateľstvo STU, 2010, s. 437–465. Edícia výskumných textov FIIT STU. ISBN 978-80-227-3284-0. Dostupné z: https://www.fit.vut.cz/research/publication/9234.

[19] Walker, J. A.; Völk, K.; Smith, S. L. a Miller, J. F. Parallel evolution using multi-chromosome cartesian genetic programming. *Genetic Programming and Evolvable Machines*, Prosinec 2009, sv. 10, č. 4, s. 417–445. ISSN 1573-7632. Dostupné z: https://doi.org/10.1007/s10710-009-9093-2.