VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA GENETICKÉ PŘÍBUZNOSTI APROXIMATIVNÍCH OBVODŮ

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH KREJČÍK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Krejčík Vojtěch**
Program: Informační technologie
Název: **Analýza genetické příbuznosti aproximativních obvodů**
Genetic Relatedness Analysis of Approximate Circuits
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou aproximativních obvodů a knihovnou EvoApproxLib.
2. Pro zvolený typ evolučně navržených obvodů z EvoApproxLib (např. 8-bitové bezznaménkové násobičky) proveďte základní analýzu jejich vlastností.
3. Definujte vhodné ukazatele příbuznosti obvodů (např. počet použitých hradel, typ použitých hradel, existence specifických podobvodů apod.).
4. Pomocí nadefinovaných ukazatelů příbuznosti zjistěte, do jaké míry jsou si jednotlivé implementace podobné na úrovni reprezentace (genotypu) a jak tato podobnost souvisí s jejich chybou a dalšími vlastnostmi na úrovni fenotypu.
5. Výsledky vhodně graficky prezentujte. Pokuste se o zobecnění dosažených výsledků.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Cílem práce je analýza velké knihovny aproximativních obvodů (EvoApproxLib), která byla vytvořena evolučním algoritmem a kterou v této práci chápeme jako zdroj genetických dat. Konkrétně se jedná o hledání příbuznosti v souboru obsahujícím 24912 osmibitových aproximativních násobiček, které byly evolučně vytvořeny ze šesti různých rodičovských plně funkčních implementací operace násobení. Jako ukazatele příbuznosti byly zvoleny počty hradel a existence 16 specifických podobvodů. Na základě těchto ukazatelů (příznaků) byly natrénovány různé klasifikátory pro zařazení násobičky do jedné ze šesti tříd odpovídající rodičovským implementacím. S těmito ukazateli se podařilo dosáhnout úspěšnosti klasifikace až 77%. Výsledky této práce ukazují, že kombinace specifických podobvodů jsou silným indikátorem, ze kterého rodičovského obvodu daný aproximativní obvod pochází.

Abstract

The goal of this thesis is analyzing a large library of approximate circuits (EvoApproxLib) which was created using an evolutionary algorithm and used as a source of genetic data for the purposes of this thesis. More specifically it is a relatedness search in a file containing 24 912 8-bit approximate multipliers which were created by evolution from six different fully functioning parent implementations of multiplication. Gate counts and existence of 16 specific subcircuits were used as relatedness indicators. Various classifiers for assigning multipliers to one of six classes corresponding to parent implementations were trained based on these indicators. A classification success rate of up to 77% was achieved using said indicators. The results of this work show that combinations of specific subcircuits are a strong indicator for identifying which parent circuit the given approximate circuit comes from.

Klíčová slova

Kartézské genetické programování, evoluční algoritmy, klasifikace, aproximace, aproximativní počítání.

Keywords

Cartesian genetic programming, evolutionary algorithms, classification, approximation, approximate computing.

Citace

KREJČÍK, Vojtěch. *Analýza genetické příbuznosti aproximativních obvodů.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Analýza genetické příbuznosti aproximativních obvodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Krejčík
16. května 2022

Poděkování

Děkují vedoucímu práce panu prof. Ing. Lukášovi Sekaninovi, Ph.D za odborné vedení a ochotu při pomoci vypracování této bakalářské práce.

Obsah

Kapitola 1

Úvod

Evoluční algoritmy se ukazují jako dobrý nástroj pro vytvoření nových technických řešení, která jsou schopna porážet člověkem navržené designy. Využívá se k tomu například při návrhu kombinačních obvodů, kde se používá typ evolučního algoritmu nazvaný kartézské genetické programování. I přesto, že takto navržené obvody můžou být energeticky efektivnější a menší, tak to pro některé aplikace nestačí. V případech, kdy jsme ochotni obětovat přesnost výpočtu, tak je možné obvody výrazně optimalizovat pro spotřebu energie, zpoždění, nebo velikost.Takový přístup se nazývá aproximativní počítání .

Cílem této práce je analyzovat struktury evolučně navržených aproximativních obvodů a hledat ukazatele vzájemné příbuznosti jednotlivých obvodů. Analyzované obvody pochází z knihovny EvoApproxLib [10] vytvořené zde na Fakultně informačních technologií Vysokého učení technického. Cíle analyzování obvodů vytvořených evolučními algoritmy jsou podobné cílům vědců, kteří analyzují genetický kód organismů. Obvody zde nebudu simulovat, ale budu hledat statistické závislosti mezi podobvody a naměřenými parametry obvodů, jako jsou spotřeba, nebo přesnost. Hlavním zaměřením práce je hledání příbuznosti jednotlivých obvodů a znaků příbuznosti.

Bakalářská práce má následující strukturu, V kapitole 2 představím základní informace o aproximativním počítání, metriky kterými se měří míra aproximace, a přehled metod návrhu aproximativních obvodů. V kapitole 3 je úvod do strojového učení s přehledem jednotlivých přístupu. Konkrétně tam jsou popsané metody pro klasifikaci, které jsem využil pro své řešení. Kapitola 4 představuje základní pojmy a koncepty, kterých se využívá v evolučních algoritmech a jak souvisí s evolucí v přírodě. Představení konkretní evolučního algoritmu je v následující kapitole 5, tímto algoritmem je kartézské genetické programování, kterým byly navržené analyzované obvody. Popis struktury obvodů z knihovny a zároveň přehled obvodů v ní obsažených je podrobně popsán v kapitole 6. Přehled využitých knihoven, algoritmů a datových struktur a celková implementační část řešení je popsána v kapitole 7. Samotné vypracování analýzy obvodu, výsledky klasifikace a grafická reprezentace výsledku je v kapitole 8.

Kapitola 2

Aproximativní obvody

Stále se zvyšující nárok na energetickou účinnost výpočetní techniky dal vzniku aproximativnímu počítání. Využívá se v oblastech, kde vyžadujeme vysokou energetickou účinnost, ale jsme schopni udělat kompromis v přesnosti výsledku. Může jít o zpracování audiovizuálních dat, kde výsledná nepřesnost je pod rozeznávací úrovní člověka, nebo typ nepřesnosti, kterou si je lidský mozek schopný domyslet. Dalšími oblastmi jsou zpracování velkých dat, zpracovaní analogových dat se šumem a některé oblasti strojového učení.Tento typ optimalizace je, dle využití, možné udělat na úrovni softwaru, ale i hardwaru. Softwarové metody využívající aproximace jsou například metody strojového učení jako K-means, nebo hluboké neuronové sítě [1].

Aproximativní obvody jsou optimalizovány i pro jiné metriky než je spotřeba. Vzhledem k vlastnostem moderních výrobních procesů čipů, využití aproximace je výhodné i z pohledu snížení komplexity a velikosti čipů a reaguje tak na snižující se spolehlivost jejich výroby [7].

Je možné aproximovat aritmetické obvody, paměti, ale i složitější bloky jako jsou obrazové filtry. Proto pro zvolení vhodných částí systému k aproximaci, je třeba daný systém podrobně analyzovat a zvolit k aproximaci části, kde bude benefit největší a chyba způsobená aproximací nejmenší.

2.1 Chybové metriky

Chybovost způsobená aproximací je většinou měřena jako rozdíl oproti přesnému systému na určité množině vstupů. Způsob měření chybovosti silně závisí na daném systému. Níže představím pouze několik metrik vhodných pro kombinační obvody, kde testování probíhá simulováním obvodu. Další specifičtější metriky představím v kapitole 5 o CGP. Informace v této kapitole vychází z [13].

2.1.1 Aritmetické chyby

$O^{(i)}_{approx}$ reprezentuje výstup aproximovaného/testovaného obvodu, $O^{(i)}_{orig}$ reprezentuje výstup přesného obvodu a n je počet vstupů.

Metrika maximální chyba (anglicky worst case error):

$$e_{wst} = \max_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}| \tag{2.1}$$

Za využití této metriky při optimalizaci systému (podmínkou $e_{wst} \leq \epsilon$) je zaručeno, že se výsledek aproximovaného systému bude lišit od správného maximálně o ϵ.

Maximální relativní chyba:

$$e_{rel} = \max_{\forall i} \frac{|O^{(i)}_{orig} - O^{(i)}_{approx}|}{O^{(i)}_{orig}} \tag{2.2}$$

Průměrná chyba (anglicky average error magnitude) je definovaná jako průměr rozdílu výsledků přesného a aproximovaného systému:

$$e_{avg} = \frac{e_{tot}}{n} = \frac{\sum_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|}{n} \tag{2.3}$$

Pravděpodobnost chyby (anglicky error probability) je definována jako poměr vstupních vektorů, kde se výsledek mezi přesným a aproximovaným systémem liší (výraz - $O^{(i)}_{orig} \neq O^{(i)}_{approx}$ nabývá hodnot 0, nebo 1):

$$e_{prob} = \frac{\sum_{\forall i}, O^{(i)}_{orig} \neq O^{(i)}_{approx}}{n} \tag{2.4}$$

2.1.2 Ověřování přibližné ekvivalence

Pro větší aritmetické obvody je nepraktické používat výše zmíněné metriky, protože výpočet chyby pomocí simulace pro větší množinu vstupů je výpočetně náročný. Pro takové obvody se využívají jiné techniky, které jseou mimo zaměření této práce, lze se o nich dočíst například zde [13].

2.2 Metody návrhu

Cílem tzv. funkcionální aproximace je navrhnout lehce odlišný obvod od přesné implementace tak, aby splňoval požadavky na akceptovatelnou chybu a spotřebu, nebo jiné parametry, které prioritizujeme. Obvykle se pro funkcionální aproximaci využívá nástrojů, které na základě heuristických postupů dojdou k výsledným obvodům. Mezi tyto metody patří SALSA, ASLAN, SASIMI, ABACUS, ABM. Přehled těchto metod je možné opět najít zde [13]. Další možností je využití metod evolučního návrhu, převážně CGP o které bude popsáno v kapitole 5.

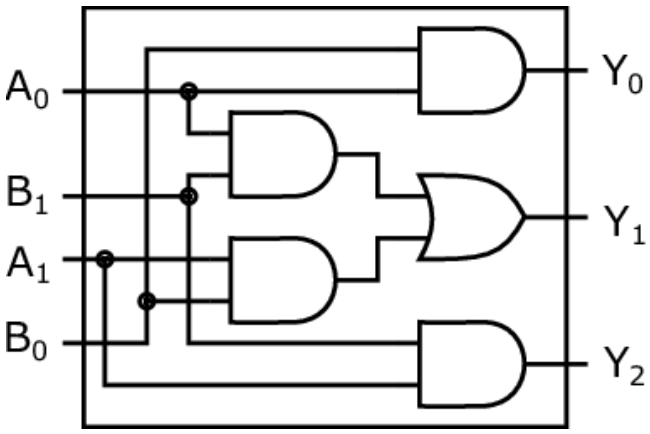
Druhou možností, jak zlepšit vlastnosti obvodu za cenu nepřesnosti, je over-scaling. Místo toho, aby byl obvod provozován na napětí, ke kterému byl určen, je mu napětí sníženo, to vyústí v nepřesnost, která je závislá ne jednotlivých obvodech a konkrétním napětí. Další snížení příkonu muže přinést změna frekvence, na které obvod funguje. Over-scaling může být kombinován s funkcionální aproximací.

Vytvoření obvodu těmito způsoby je možné buďto ručně, nebo automatizovaně.

Příklad ručně navrženého aproximovaného obvodu

Příkladem ručně navrženého obvodu může být násobička na obrázku 2.1 popsána v [8]. Tento obvod násobí dvě 2-bitová čísla, ale výsledek pouze tříbitový. Výsledky pro všechny vstupy, lze přesně reprezentovat pomocí tří bitů, kromě vstupu 3x3. Správný výsledek pro tento vstup by byl 9 (binárně 1001), ale výsledek tohoto obvodu bude 7 (111). Tabulka výstupů pro všechny vstupy je ukázaná v právě části obrázku 2.1. Výsledek je tedy v 15 ze 16 případů správný. Výměnou za nepřesnost je komplexita obvodu redukována, plocha,

kterou obvod zabírá, je téměř poloviční. Tento obvod lze použít jako blok pro vytvoření větších násobiček.



B \ A	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	7

Obrázek 2.1: **Aproximovaný obvod** - násobící dvě dvoubitová čísla. Vlevo je návrh jeho designu a vpravo výsledky pro jednotlivé vstupy. Pro vstup 3x3 je výsledek označen červeně, protože je nepřesný, je 7 místo 9. Obrázek je převzatý z [11].

Kapitola 3

Strojové učení

Strojové učení je oblastí umělé inteligence, která se zabývá možnostmi "učení se"pro počítače. Algoritmy z této oblasti nejsou programovány do posledního detailu, aby plnily danou funkci, ale jejich parametry jsou optimalizovány s využitím dostupných dat, aby byly schopny předpovídat správný výsledek v praxi. Algoritmy strojového učení nachází využití například v rozpoznávání řeči, počítačovém vidění a v mnoha dalších aplikací. Tyto algoritmy lze rozdělit do tří skupin, dle způsobu trénování a trénovacích dat. Úvod této kapitoly vychází z [15] a [12].

3.1 Způsoby učení

3.1.1 Učení s učitelem

Tato třída algoritmů se vyznačuje tím, že trénovací množina dat je tvořena dvojicemi vstupu a očekávaného výstupu. Cílem je namapovat tyto vstupy na výsledky tak, aby fungovaly i na dosud neviděných vstupech, tedy dosáhnout generalizace. Podrobně popíšu jen algoritmy, které jsem ve své práci využil, ale významných jich je mnohem víc, jmenovitě to jsou například lineární regrese, logistická regrese, rozhodovací stromy, random forest, k-means a umělé neuronové sítě.

3.1.2 Učení bez učitele

Trénovací množina dat je neoznačená, to znamená, že algoritmus neví, jaký je správný výsledek a snaží se objevit vzory v datech.

3.1.3 Posilované učení

Místo trénovací množiny dat, zpětnovazební učení pracuje s agenty v prostředí. Agenti se pohybují v prostředí, dle předem stanovené množiny akcí a jsou odměňováni dle kvality řešení, kterého dosáhli (případně na základě každého kroku).

3.2 Klasifikace

Jednou z úloh, pro které je využíváno metod strojového učení, je klasifikace. Cílem je rozhodnout o vzorku, do jaké třídy, z předem definované množiny tříd, patří. Většina algoritmů

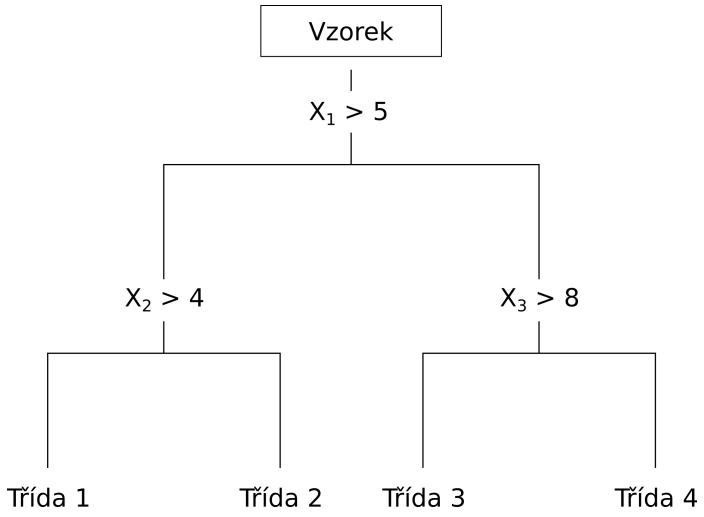
implementující klasifikaci fungují na principu učení s učitelem, kde očekávaným výstupem je právě třída vzorku. Algoritmus implementující klasifikaci je nazýván klasifikátor.

3.2.1 Random Forest

Random forest (česky náhodný les) je, jak název napovídá, metoda kombinující zkonstruování několika rozhodovacích stromů pro klasifikaci a regresi.Tato sekce vychází z následujících článků [18] a [16].

Rozhodovací stromy

Rozhodovací stromy klasifikují na základě veličin známých o vzorku, viz 3.1. Rozhodování je reprezentováno uzly stromového grafu, kde dojde k větvení. Průchod grafem končí, když se dostaneme k listu. Listy reprezentují výslednou třídu, do které byl vzorek klasifikován.



Obrázek 3.1: **Rozhodovací strom** Ukázka jednoduchého rozhodovacího stromu, kde x_1 , x_2 a x_3 jsou příznaky vzorku.

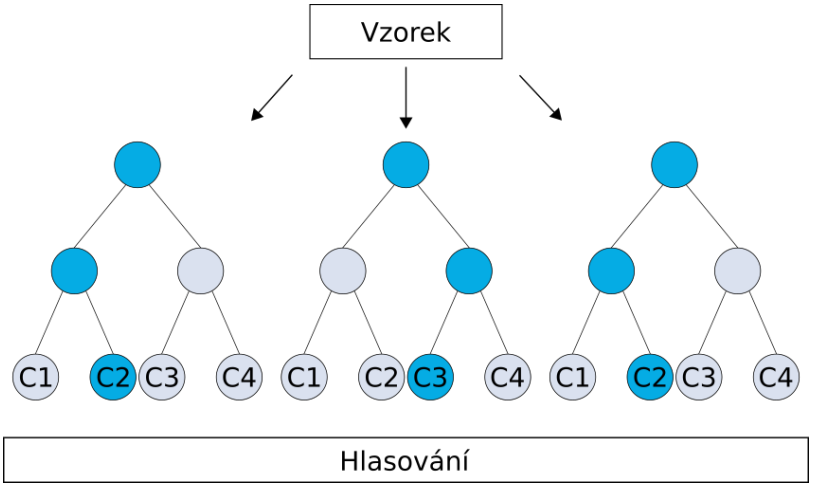
Bagging

Bagging (akronym z anglického pojmu bootstrap aggregating) je metoda, na které je založeno vytváření náhodných lesů. Trénovací množina dat se náhodně rozdělí do n množin a pro každou z nich je nezávisle vytvořený rozhodovací strom. Vizualizace průběhu klasifikace je na obrázku 3.2.

Random forest je oproti základním algoritmům pro rozhodovací stromy odolný proti přetrénování (tzv. overfitting) a přesto, že trénovaní modelu i vyhodnocení výsledku je výrazně složitější, než u rozhodovacích stromů, stále se jedná o relativně rychlý algoritmus, který poskytuje dobré výsledky pro různá data.

3.2.2 Analýza hlavních komponent

Analýza hlavních komponent (dále jako PCA z anglického principal component analysis) je metoda sloužící k dekorelaci dat a snížení počtu dimenzí s co nejnižší ztrátou informace. Minimalizovaní počtu dimenzí při zachování co největšího množství informací je žádané pro



Obrázek 3.2: **Random forest** - zjednodušena reprezentace průběhu rozhodování algoritmu random forest. Listy stromu označené jako C1, C2, C3 a C4 jsou třídy, do kterých je klasfikováno. Třída, do které vzorek patří, je vyhodnocena rozhodovacími stromy 1, 2, a 3. Tmavě modře je vyznačená cesta grafem, tedy samotné rozhodování. Konečný výsledek je ten, který se objevil nejvíce-krát, v tomto případě je to třída 2.

mnoho metod strojového učení. Použití PCA může výrazně urychlit učení algoritmů, nebo umožnit použití jiných algoritmů, které nebyly vhodné na původní data. Tato kapitola o PCA je zpracována podle [5] a [14].

Postup

1. Utvoření matice X pro $m \times n$, kde m je počet vzorků a n počet atributů pro každý vzorek

2. Standardizace dat:
   PCA je velmi náchylné na rozptyl hodnot jednotlivých atributů. Například atribut nabývajících hodnot 0 - 1000 by výrazně dominoval atributy jehož hodnoty by se pohybovaly pouze v rozmezí 0 - 10. Tomuto problému lze předejít standardizací dat podle následujícího vzorce.
   $$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\sigma(x_j)} \tag{3.1}$$
   Pro každou hodnotu $x_{i,j}$, kde i je číslo řádku a j číslo sloupce matice, je vypočtena. $\bar{x}_j$ jako průměrná hodnota sloupce j a $\sigma(x_j)$ je střední odchylka sloupce j. Výslednou matici nazveme X'.

3. Sestrojení kovarianční matice
   $$Cov(X') = \begin{bmatrix} Cov(A_1, A_1) & .. & Cov(A_1, A_n) \\ ... & .. & ... \\ Cov(A_n, A_1) & .. & Cov(A_n, A_n) \end{bmatrix} \tag{3.2}$$
   A_i jsou jednotlivé sloupce matice X' a Cov(...) značí kovarianci. Kovariance je definovaná následovně:
   $$Cov(X, Y) = E[(X - E[X])(Y - E[Y])] \tag{3.3}$$

kde E značí střední hodnotu.

4. Výpočet vlastích čísel a vlastních vektorů z kovarianční matice.

$$|Cov(X') - \lambda I| = 0 \tag{3.4}$$

Výpočtem λ získáme vlastní čísla. Pomocí nich poté pro každé vlastní číslo λ_i spočítáme vlastní vektor, kde x_i je vlastní vektor.

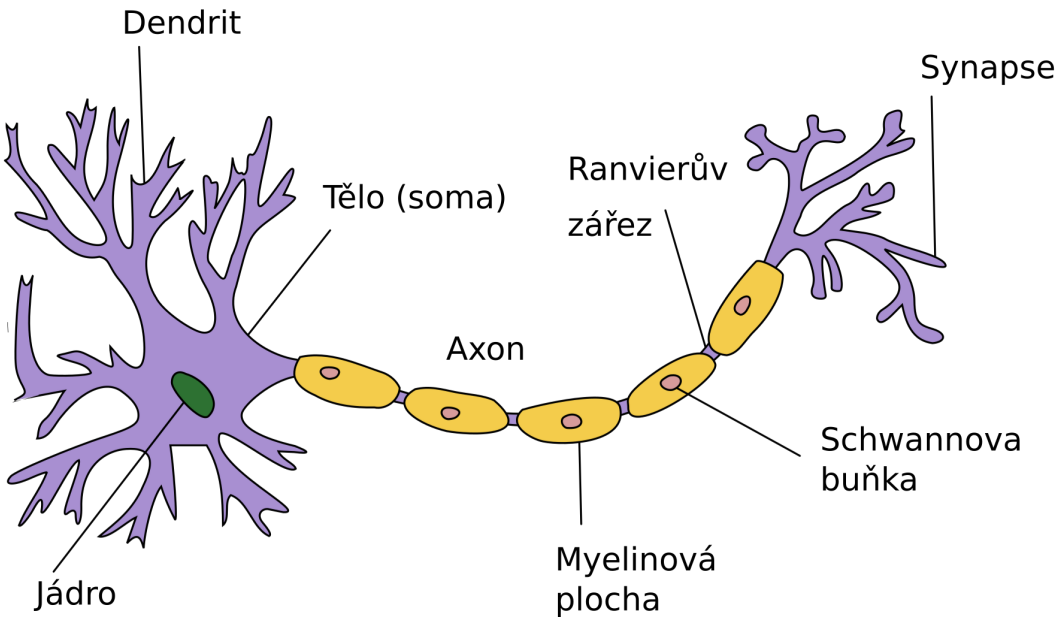
$$(Cov(X') - \lambda_i I)x_i = 0 \tag{3.5}$$

5. Spočítané vlastní vektory se seřadí dle jejich vlastních čísel a poté použijeme prvních n vektorů, kde n je počet komponent, které chceme použít.

3.2.3 Vícevrstvý perceptron

Vícevrstvý perceptron (anglicke multialyer perceptron, dále jako MLP) je třída dopředných umělých neuronových sítí. Tato kapitola vychází z článku [3].

Neuron



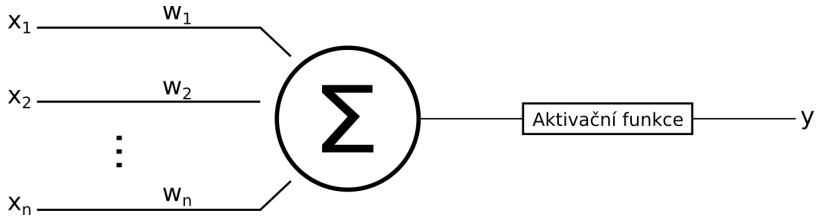
Obrázek 3.3: **Obrázek neuronu** pro ilustraci jednotlivých částí buňky. Převzato z [6].

Název a částečně i fungování těchto neuronových sítí vychází z nervové buňky, neboli neuronu, vysoce specializované buňky tvořící nervovou tkáň, viz obrázek 3.3. Neurony vynikají schopnosti přijímat signály z jiných neuronů, zpracovat je a posílat signál dalšímu neuronu. Neuron se skládá z těla buňky (soma), dendritů - výběžku, které akceptují signály z jiných neuronů a jednoho axonu. Axon je vlákno zakončené takzvaným axonálním zakončením, které slouží ke předání informací dalšímu neuronu. Axon je mnohonásobně delší než samotné tělo buňky. Spojení dvou neuronů se nazývá synapse.

Učení neuronu leze chápat jako nastavování vah jednotlivých synapsí, což je koncept, kterým se inspirují umělé neuronové sítě.

Perceptron

První pokus o využití principu neuronu nastal v 40. letech 20. století, kdy vědci Warren McCulloch a Walter Pitts vytvořili model neuronu. Na základě vstupů a vah byl jeho výstup pozitivní nebo negativní. Na rozdíl od pozdějších modelu, tento postrádal schopnost se učit a zcela závisel na počátečním nastavení vah. Později Frank Rosenblatt rozšířil tento model o algoritmus umožňující učení a nazval ho perceptron.



Obrázek 3.4: **Perceptron** - pro ilustraci jednotlivých částí.

$$y = \begin{cases} 1, & \text{pokud } \sum w_i x_i - T > 0 \\ 0, & \text{jinak} \end{cases} \tag{3.6}$$

kde T je rozhodovací hranice určená aktivační funkcí a $\sum w_i x_i$ je vážená suma vstupů.

Aktivační funkce

Cílem aktivační funkce je namapovat váhovanou sumu vstupů do intervalu <-1, 1>, nebo <0, 1>.

- Sigmoid funkce, nebo také logistická křivka.

$$f(x) = \frac{1}{(1 + exp(-x))} \tag{3.7}$$

- ReLU (Rectified Linear Unit) - často používaná aktivační funkce pro jednoduchost a rychlost učení.

$$f(x) = max(0, x) \tag{3.8}$$

Základní perceptron je vhodný pouze pro binární klasifikaci dat, která jsou lineárně oddělitelná. Pro učení perceptron používá gradientní sestup, který zajišťuje úpravu vah, pro lepší výsledek.

MLP se zbavuje limitací základní perceptronu. Umožňuje klasifikaci i dat, které nelze oddělit lineární funkcí, nebo klasifikovat více tříd. Dosahuje toho za využití více perceptronů v několika vrstvách. Má vstupní vrstvu, jednu, nebo více skrytých vrstev a výstupní vrstvu, která pro binární klasifikaci obsahuje jeden perceptron, nebo tolik kolik je klasifikačních tříd. Výstup perceptronu z jedné vrstvy je spojený se všemi perceptrony z následující vrstvy. Výsledky z každé vrstvy se propagují do další, než je dosaženo výstupní vrstvy.

Učení MLP

Učení probíhá pomocí tzv. zpětné propagace chyby (backpropagation), která iterativně upravuje váhy vstupů v síti ve směru od výstupní vrstvy ke vstupní.Typicky průběh učení

sítě začíná inicializací vah na náhodnou hodnotu. Poté síť vyhodnocuje trénovací data a výsledek sítě je ohodnocen pomocí ztrátové funkce. Cílem učení je minimalizovat ztrátovou funkci pomocí gradientního sestupu.

3.2.4 Support vector machine

Support vector machine (dále jako SVM) je metodou využívající učení s učitel pro klasifikaci. Základní algoritmus slouží pro binární klasifikaci, kde učení algoritmu spočívá v hledání nadroviny pro rozdělení prostoru a tím klasifikaci dat. Více informací o kalsifikaci pomocí SVM je v článku [4].

Kapitola 4

Evoluční algoritmy

Evoluční algoritmy jsou heuristikou pro prohledávání stavového prostoru, kde stavový prostor je množina všech kandidátních řešení problému. Tyto algoritmy jsou inspirované biologickým procesem evoluce, který byl popsán Darwinem. Využívá se například koncept vycházející z přirozeného výběru z Darwinovi teorie a moderních myšlenek. Darwinova teorie byla později rozšířena Gregorem Mendelem, což dalo vzniku neodarwinismu. Od něho evoluční algoritmy přebírají náhodné mutace genetické informace. Kapitola vychází z článku [2], doplněná o informace z knihy [9].

4.1 Pojmy

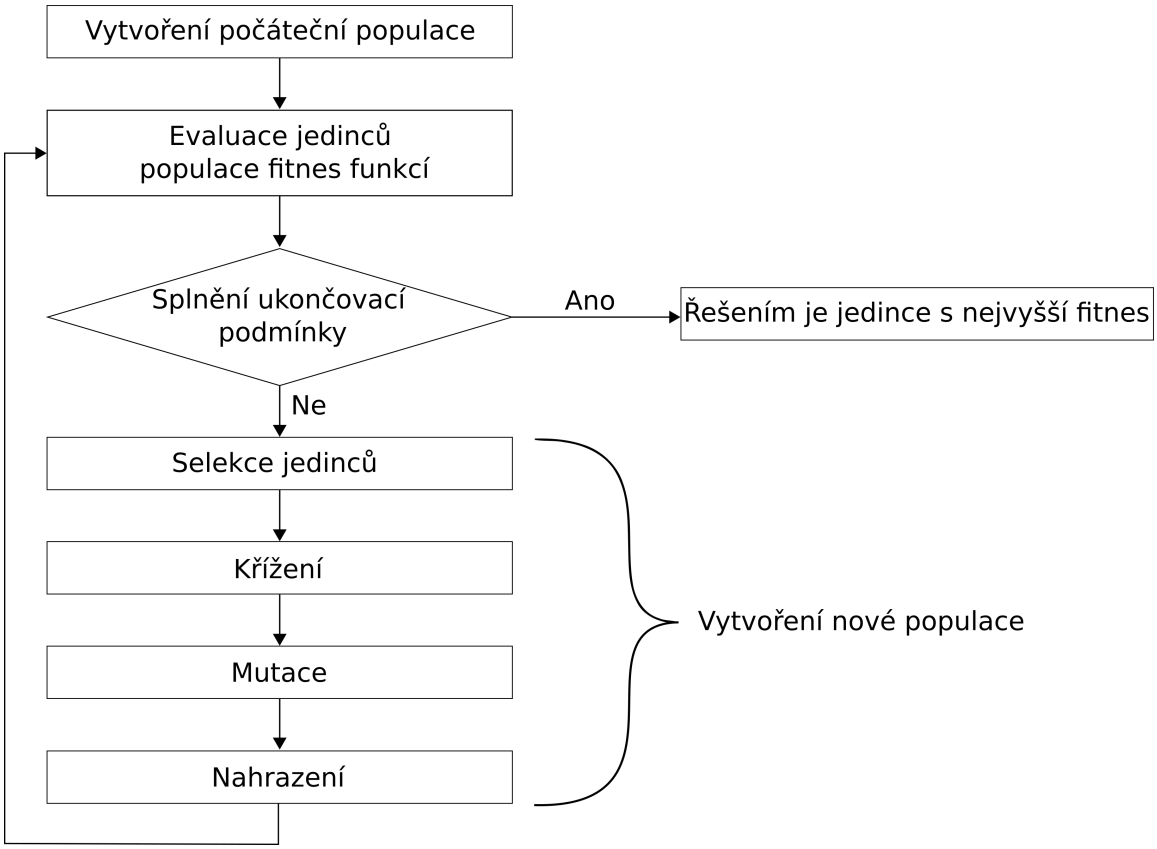
Vzhledem k tomu, že značná část pojmů používaná k popisu fungování evolučních algoritmů je převzatá z biologie, tak zde bude vysvětlen jejich informatický význam.

- **Gen** - základní stavební jednotka chromozomu, jeho hodnota (tzv. alela) patří do definované abecedy (např. binární čísla, celá čísla atd.)
- **Chromozom, genotyp, jedinec** - jedno zakódované kandidátní řešení, které se skládá z genů.
- **Fenotyp** - je sestaven na základě chromozomu (příklad - fenotyp je obvod, zatímco chromozom říká, jak poskládat součástky po sobě pro vytvoření daného obvodu)
- **Populace** - skupina jedinců dané velikosti.
- **Fitness** - kvalita řešení.
- **Fitness funkce** - získává fitness hodnotu pro jedince.
- **Evaluace** - ohodnocení populace, nebo jedince fitness funkcí
- **Selekce** - výběr jedinců z populace.
- **Křížení, Mutace** - genetické operátory.
- **Generace** - výsledek jedné iterace evolučního algoritmu

Dále zde budu používat pojmy jako rodič a potomek, kde potomek je jedinec vytvořený pomocí mutace, nebo křížením z jiného jedince - rodiče.

4.2 Průběh evoluce

Cyklus průběhu evolučních algoritmů je popsán obrázkem 4.1. V závislosti na konkrétním algoritmu můžou být jednotlivé kroky odlišné, nebo úplně chybět. Mnoho algoritmů spoléhá buďto jen na mutaci, nebo křížení.



Obrázek 4.1: **Cyklus evolučních algoritmů** - obecný průběh evolučních algoritmů. V závislosti na implementaci konkrétního přístupu může chybět mutace, nebo křížení.

4.2.1 Návrh

Abychom mohli řešení problému hledat pomocí evolučních algoritmů, je nezbytné problém zakódovat. Způsobů kódování je více a volba závisí na konkrétním problému, který řešíme. Chromozom může být zakódován jako binární řetězec, gramatika, stromová struktura a mnoho dalších. V závislosti na algoritmu, chromozom může být fixní, nebo proměnlivé délky.

Druhou podstatnou věcí, než se rozhodneme řešit problém evolucí, je vymyslet fitness funkci. Fitness funkce, respektive evaluace kandidátních řešení, bývá typicky výpočetně nejnáročnější částí evolučních algoritmů. Pro návrh programu může jít o několik spuštění s různými vstupy, pro obvod o jeho simulaci, viz kapitola 2.1 a v případě karoserie auta o simulaci aerodynamiky. Výpočet fitness funkce je potřeba provést pro každého jedince z každé generace. Počet jedinců, který se může za dobu evoluce pohybovat v řádech sta tisíců, v kombinaci s přílišnou složitostí fitness funkce tedy bývají největším důvodem, proč nejsou evoluční algoritmy aplikovatelné na daný problém.

4.2.2 Inicializace

Inicializací je myšleno vytvoření počáteční populace. Na jedince z počáteční populace může být referováno jako na seed (v angličtině semínko). Chromozom počáteční populace bývá typicky složen z genů, kterým byla přiřazena náhodná hodnota. Pro komplexnější problémy se dají použít již funkční řešení, kde bude cílem evoluce je vylepšit. Další možností je použití tzv. embrya. Embryo je nějaký základní blok, který evoluce bude následně rozšiřovat a upravovat.

V případě evolučních algoritmů prvně dojde k namapování chromozomu na fenotyp a ten je poté ohodnocen.

4.2.3 Evaluace

Protože populace algoritmu neexistuje v žádném reálném prostředí, kde by byl tlak na přežití pouze nejsilnějších jedinců, simulujeme toto prostředí pomocí analýzy a hodnocení jedinců. Hodnocení jedinců zajišťuje fitness funkce. Takto ohodnocen musí být každý jedinec. V závislosti na řešeném problému, bývá k ohodnocení používám software specifický pro daný obor.

V případě genetických algoritmů první dojde k namapování chromozomu na fenotyp a ten je poté ohodnocen.

4.2.4 Výběr rodičů

Výběr rodičů (selekce) je zásadní pro všechny evoluční algoritmy. Cílem je vyfiltrovávat omezené množství jedinců, kteří se budou reprodukovat. Metod výběru je několik, ale většina je založena na preferování jedinců s vyšší hodnotou fitness v kombinaci s náhodností.

- Řazení dle pořadí fitness - jedinci jsou seřazeni dle jejich fitness hodnoty. Jejich šance na výběr je přímo úměrná tomu, jak vysoko se umístili.
- Turnajový výběr - šance jedince být rodičem je založena na tom, kolik jiných jedinců v turnaji byl schopný porazit (tedy má větší fitness).
- Ruleta - šance na výběr jedince je založená na podílu fitness jedince a sumy fitness celé populace.

4.2.5 Plodnost

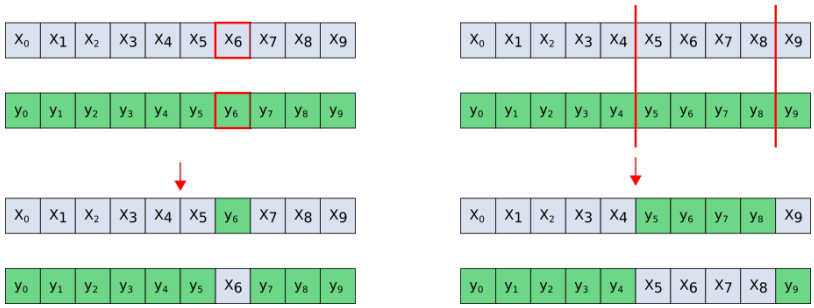
Plodnost jedince určuje, kolik potomků bude mít. Tato hodnota můžeme být nastavena pevně pro celý algoritmus, nebo může být zvolena pro každého rodiče samostatně na základě jejich fitness hodnoty.

4.2.6 Reprodukce

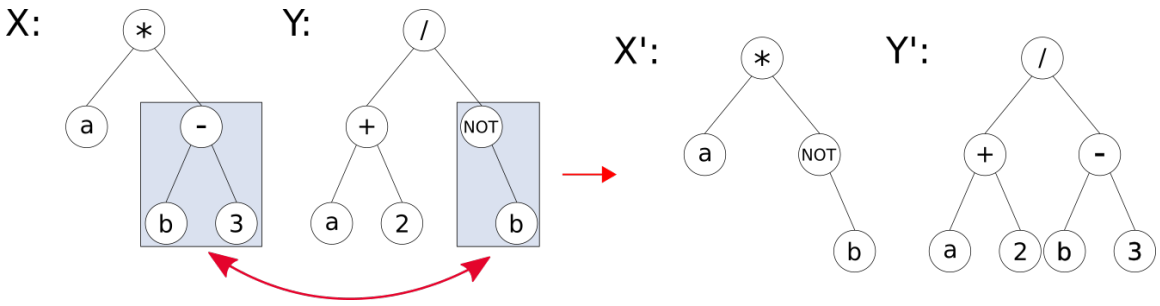
Reprodukce je další nepostradatelnou částí evolučních algoritmů. Je zodpovědná za vytvoření nových jedinců a zajišťuje, že potomci zdědí vlastnosti rodičů, ale zároveň budou odlišní tak, aby bylo možné dosáhnout nových řešení. Využívá se k tomu konceptů křížení, nebo mutace nebo obojího.

Křížení

Křížení vyžaduje minimálně dva rodiče, jejichž chromozomy se nějakým způsobem zkombinují. Může jít o prohození jednoho náhodného genu, nebo o prohození náhodně dlouhého řetězce genů, opět záleží na konkrétním algoritmu, viz obrázky 4.2 a 4.3.



Obrázek 4.2: **Křížení 1** - ukázka dvou způsobů křížení na chromozomem kódovaných jako řetězce znaků. Vlevo je ukázka bodového křížení, kde se prohodí geny na náhodně zvolené pozici. Napravo se prohazuje úsek genů. Začátek i konec úseku bývá volen náhodně.



Obrázek 4.3: **Křížení 2** - Ze dvou rodičovských stromů vzniknou dva potomci prostým prohozením náhodně zvolených podstromů.

Mutace

Mutace v kontextu evolučních algoritmů většinou znamená , že se náhodně vybraný gen chromozomu změní na náhodnou hodnotu. Například v evoluční strategii náhodná hodnota může pocházet z normálního rozložení.

4.2.7 Nahrazení

Protože populace má fixní velikost, tak je potřeba za každého nového jedince odstranit jednoho jedince z původní populace. Jednoduší algoritmy obmění pokaždé celou populaci, ale běžněji se nahrazuje pouze část předem definované velikost. Výběr jedinců, kteří zůstanou probíhá na základě fitness podobně jak bylo popsáno v části o selekci Může zde fungovat tzv. elitismus, který způsobí, že nejlepší jedinec z populace se automaticky dostane do té nové.

4.2.8 Ukončení

Ukončení algoritmu proběhne po splnění ukončovací podmínky.Tou může být určitý počet iterací, nebo dosáhnutí řešení problému.

Kapitola 5

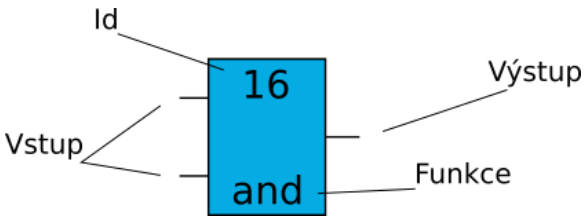
Kartézské genetické programování

Kartézské genetické programovaní, dále jako CGP (zkratka z anglického Cartesian Genetic Programming) je, jak název napovídá, typem genetického programování. Název kartézské referuje na uspořádání uzlů chromozomu v dvou-dimenzionální mřížce. Pro potřeby této práce budu popisovat CGP v souvislosti s návrhem kombinačních obvodů, ale většina informací bude obecně platná. Informace z této kapitoly vychází z následující knihy [9].

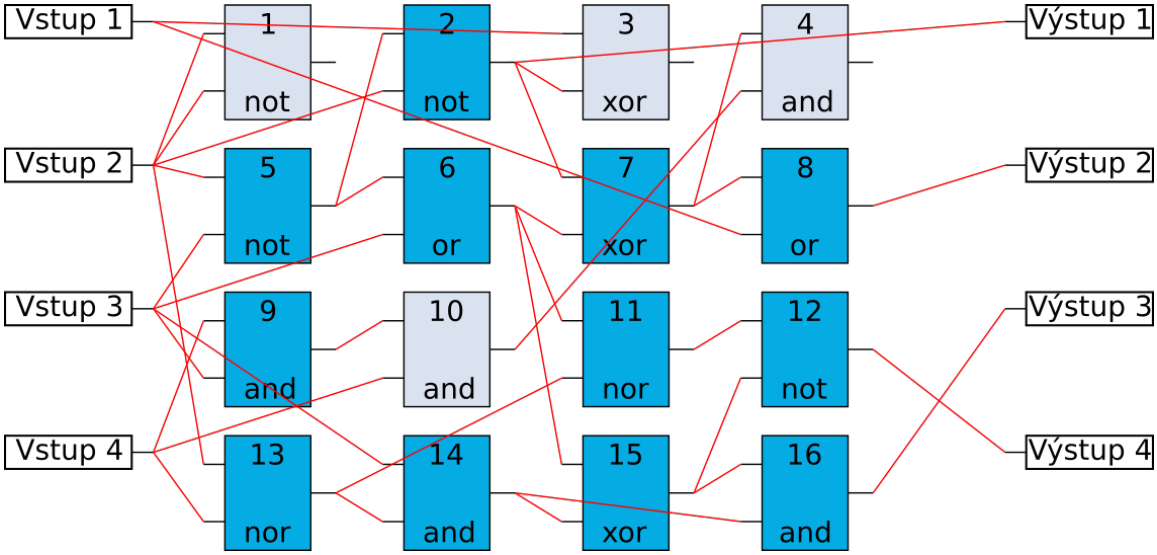
5.1 Reprezentace problému v CGP

Chromozom je reprezentovaný acyklickým grafem uspořádaným v dvou-dimenzionální mřížce. Pro správné fungování je potřeba mít CGP nakonfigurované pomocí následujících parametru:

- počet vstupů,
- počet výstupů,
- počet řádků mřížky,
- počet sloupců mřížky,
- počet vstupů uzlů,
- L-back parametr,
- množina funkcí, kterých může uzel nabývat.



Obrázek 5.1: **Uzel CGP** - ukázka, jak může vypadat uzel CGP pro elektrický obvod. Vstupy jsou v CGP reprezentovány jako Id uzlů, jejichž výstup je použit.



Obrázek 5.2: **Obvod v CGP** - ukázka chromozom jedince v CGP pro elektrický obvod. Tmavě modré uzly jsou aktivní a světle modré neaktivní, tedy nepřipojené na výstup.

Samotný chromozom jedince je zakódovaný jako řetězec celých čísel. Každý uzel grafu se v chromozomu skládá z id, z $k+1$ genů, kde k je počet vstupů uzlu(v tomto případě dvou), tedy označení připojených uzlů a funkce, kterou uzel vykonává. Na konci chromozomu je d genů popisujících zapojení výstupů obvodu, viz obrázky 5.1 a 5.2.

5.1.1 L-back

Pro zajištění acykličnosti grafu jo možné do uzlu připojit uzel pouze z předchozích sloupců. Nejnižší hodnoty, kterou L-back parametr může nabývat, je 1, kdy bude možné propojit pouze předchozí sloupec. Nejvyšší hodnota je max, kde není žádné omezení. Výjimkou jsou primární vstupy, které je možné připojit kamkoliv.

5.1.2 Genetické operátory

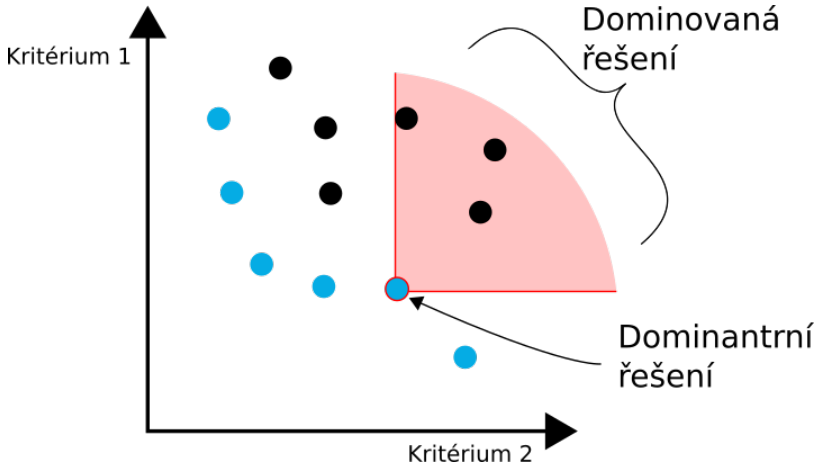
Z genetických operátorů se používá jen mutace a má tedy zásadní vliv na vývoj jedince. Může jít o mutaci pravděpodobnostní, kdy má každý uzel stejnou pravděpodobnost na mutaci. Alternativou je náhodný výběr k celých čísel z chromozomu, které se nahradí náhodnou validní hodnotou. Mutace tedy může měnit vstupy uzlů, jejich funkce a výstup kterých uzlů se použije jako výstup celku. Tímto vznikají neaktivní uzly. Neaktivní uzly se můžou mutací stát znovu aktivními a tedy není důvod nemutovat i je samotné. Taková mutace je pak neutrální, nemá vliv na fenotyp a tedy ani na fitness.

5.1.3 Převod chromozomu na fenotyp

CGP patří mezi evoluční algoritmy, kde chromozom neodpovídá konečnému řešení, ale je z něj potřeba odvodit fenotyp. Neaktivní uzly jsou takové, které nejsou ani nepřímo připojené k výstupu celku. Pro oddělení aktivních uzlů se chromozom prochází postupně od výstupu, než se dostaneme na vstup.

5.1.4 Více-kriteriální optimalizace

Někdy nám nestačí optimalizovat řešení pomocí jedné fitness funkce (u aproximativního obvodu nás nezajímá například jen přesnost, ale také spotřeba obvodu). Fitness funkce pro náš problém tedy může být suma váhovaných dílčích fitness. Řešení, která mají nejlepší kombinace všech parametrů jsou takzvaně Pareto optimální viz kapitola 5.3.



Obrázek 5.3: **Pareto linie** - Modře vyznačená řešení jsou dominantní, leží na tzv. Pareto-linii. Červeně označené řešení je lepší v obou kriteriích než ta řešení v červeně vyznačené oblasti. Cílem je minimalizovat obě kritéria.

f

Kapitola 6

Zdrojová data - knihovna aproximativních obvodů

Analyzovanými daty je knihovna 8-bitových aproximativních bez-znaménkových násobiček EvoApproxLib[1]. Tato knihovna byla vytvořena na Fakultě informačních technologií na Vysokém učením technickém v Brně. V knihovně je obsaženo 24 912 násobiček vygenerovaných pomocí algorimtu založeném na kartézském genetickým programování, které se odlišují v míře aproximace, spotřebě a velikostí. Jednotlivé implementace jsou k dispozici ve Verilog, C, Matlab a jako kód genotypu. Další informace o charakteristice jedinců jsou v json formátu. Pro potřeby analýzy si vystačím s genotypem a charakteristikou obvodů.

6.1 Reprezentace chromozomu

Genotyp/chromozom je v souborech s příponou .chr. První řádek souboru obsahuje seznam vstupů – dva 8bitové operandy, druhý řádek 16-bit výstup. Třetí řádek obsahuje parametry obvodu a popis jednotlivých hradel v následujícím formátu:

- {16 vstupů, 16 výstupů, počet hradel, X, X, X, X}
  - X jsou další parametry CGP
- ([ID hradla], vstup 1, vstup 2, funkce)
  - vstup 1 a vstup 2 jsou ID hradla, jehož výstup byl použitý jako vstup
  - funkce, které mohou hradla zastávat:
    - 0 -IDA (identita)
    - 1 - INVA (inverze prvního vstupu)
    - 2 - AND2
    - 3 - OR2
    - 4 - XOR2
    - 5 - NAND2
    - 6 - NOR2
    - 7 - XNOR2

[1] https://ehw.fit.vutbr.cz/evoapproxlib/

- (ID_OUT$_1$, ID_OUT_2, ...,ID_OUT_{15}, ID_OUT$_{16}$)
  - ID_OUT$_i$ - ID hradla, jehož výstup je daným výstupem obvodu

6.2 Charakteristika obvodů

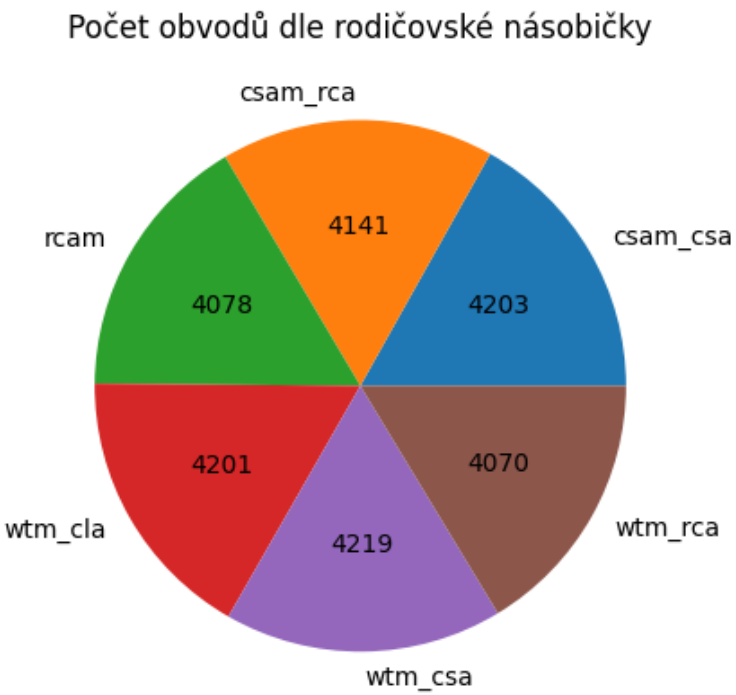
Další informace o obvodech jsou v souboru cgp-approx14ep.json. [17]

6.3 Rodiče

Obvody nacházející se v EvoApproxLib byly evolučně vytvořeny z 6 rodičovských obvodů - 8b přesných násobiček, které se liší zejména ve velikosti a zpoždění, viz tabulka 6.1 a obrázek 6.1. Názvy rodičovských obvodů referují na architekturu daného návrhu. Protože násobičky jsou typicky složené ze sčítaček, tak druhá trojce písmen referuje na specifické sčítačky, zatímco ta první na architekturu násobiček. Architektury násobiček jsou rcam - Ripple-Carry Array, csam - multiple Carry-Save Array a wtm - Wallace Tree architectures. Architektury sčítaček jsou rca - Ripple Carry Adder, csa - Carry Select Adder a cla - Carry Look-Ahead Adder. Počet obvodů pocházejících z daného rodiče je zobrazen na grafu 6.1.

Tabulka 6.1: Počet jednotlivých typů hradel, dle rodičovského obvodu.

Jméno	Počet hradel	and	or	xor	nand	nor	xnor
wtm_cla	418	224	83	111	0	0	0
csam_csa	347	180	54	108	0	0	0
csam_rca	320	168	48	104	0	0	0
wtm_csa	396	202	61	122	0	0	0
rcam	320	168	48	104	0	0	0
wtm_rca	332	174	47	111	0	0	0



Obrázek 6.1: **Počet obvodů** dle rodiče.

Kapitola 7

Cíle analýzy a implementace

Cílem této práce je najít znaky analyzovaných aproximativních obvodů, které by mohly napovídat o příbuznosti rodičovských obvodů, nebo jejich vlastnostem jako je spotřeba, přesnost, nebo plocha. Identifikace typických znaků by mohla v budoucnu umožnit vylepšení metod aproximace.

Pro analýzu dat jsem si zvolil použiti programovacího jazyka Python, konkrétně ve verzi 3.8. Hlavními důvody pro volbu Pythonu byla nabídka knihoven vhodných pro práci a analýzu dat, ale taky výchozí funkce jazyka, které například práci s textem, nebo objekty jazyka dělají velmi jednoduchou a rychlou.

7.1 Využité knihovny

7.1.1 Pandas

Knihovna implementující objekt DataFrame[1]. DataFrame uchovává data v podobě tabulky, nad kterou umožňuje mnoho operací, které pracují nad celou tabulkou, jednotlivými řádky, či sloupci. Jedná se o různé aritmetické operace, třídění, řazení a vizualizaci.

7.1.2 Sickit-learn

Sickit-learn je knihovna nabízející nástroje potřebné pro strojové učení a pro předzpracování dat[2]. Veškeré metody strojového učení popsané v kapitole 3 jsem použil za pomocí této knihovny. Protože Sickit-learn neumí pracovat nad Pandas DataFramem použil jsem i objekt array z knihovny Numpy.

7.1.3 Matplotlib a Seaborn

Matplotlib je obsáhla knihovna pro vytváření grafů a vizualizací v jazyce Python[3]. Seaborn take slouží pro vytváření grafů, ale oproti matplotlibu nabízí vysokoúrovňový přístup a umí pracovat s DataFramem, z knihovny Pandas.

[1] https://pandas.pydata.org/docs/user_guide/index.html#user-guide
[2] https://scikit-learn.org/stable/user_guide.html
[3] https://seaborn.pydata.org/tutorial.html https://matplotlib.org/stable/users/index

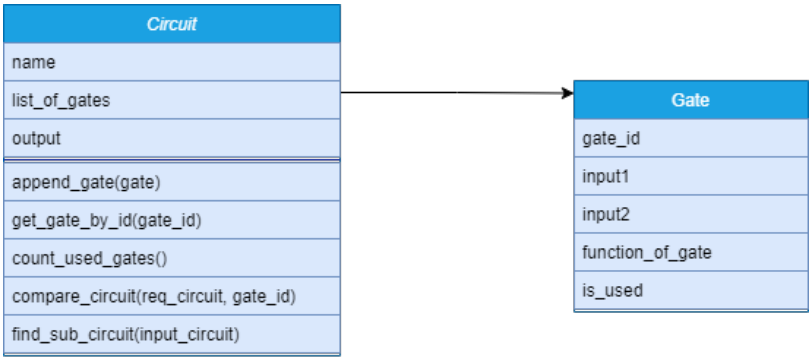
7.1.4 Paretoarchive

Knihovna pro výpočet Pareto linie, která funguje s Pandas DataFramem [4].

7.2 Struktura programu

Program je strukturovaný do dvou souborů. V prvním souboru, bp-script.py probíhá zpracování dat ze zdrojových a jejich následná analýza jako je počítání jednotlivých typů hradel, podobvodů, velikost souboru chromozomu a jeho velikost po kompresi. K tomu jsem navrhl třídy Circuit a Gate, které další analyzování chromozomu obvodů usnadňují

Všechny informace jsou uloženy v DataFramu (objekt z knihovny Pandas). Kvůli časové náročnosti analýzy obvodů, není žádané jí provádět při každém spuštění. Proto objekt dataframu serializuji a ukládám do souboru dataframe.pkl pomocí knihovny Pickle. Díky tomu můžu získaná data kdykoliv otevřít v jiném skriptu a pokračovat, kde jsem skončil.



Obrázek 7.1: **UML digram** tříd.

7.3 Vyhledávání podobvodů

Vyhledávání podobvodů zajišťuje metoda třídy Circuit – `find_sub_circuit()`, ta dále využívá metodu `compare_circuit()`. Metoda find_sub_circuit se stará o zpracování vstupu a počítání výskytů, samotné vyhledání probíhá rekurzivně v metodě compare_circuit.

7.3.1 Vstup

Nejdříve je potřeba nějak specifikovat podobvody, které budou hledány. V BP je zvolen přístup, kdy jsou tyto podobvody popsány gramatikou. Formát vstupního textového řetězce jsem zvolil tak, aby bylo možné snadno implementovat rekurzivní algoritmus. To mi umožňuje volnost ve velikosti a uspořádání podobvodů, které budu hledat. Tento zápis ale příliš neodpovídá struktuře programu, proto se způsobu implementace budu v následují části podrobněji věnovat. Vstup lze popsat následující gramatikou:

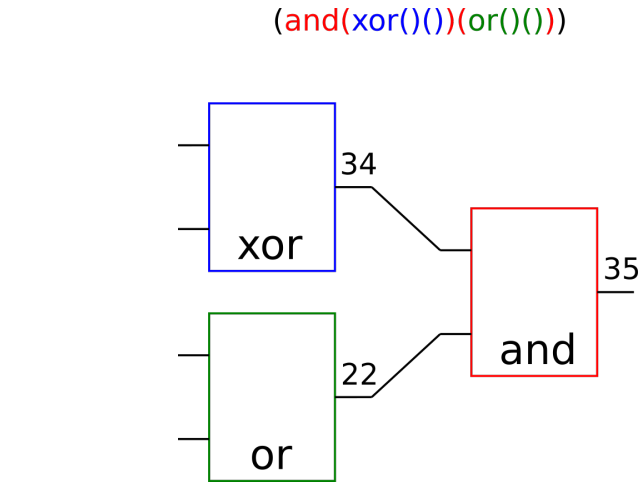
$G = (\{S, A\}, \{(,), xor, nor, and, or, nand, inv, wire, nxor, \epsilon\}, P, S)$,
kde P:
$S \rightarrow (A)$

[4] https://github.com/ehw-fit/py-paretoarchive

$A \rightarrow B(A)(A)|\epsilon$
$B \rightarrow xor|nor|and|or|nand|inv|wire|nxor$

Kde S je počáteční symbol, $\{S, A\}$ je množina neterminálních symbolů. $\{(,), xor, nor, and, or, nand, inv, wire, nxor, \epsilon\}$ je abeceda terminálních symbolů. P je množina přepisovacích pravidel.

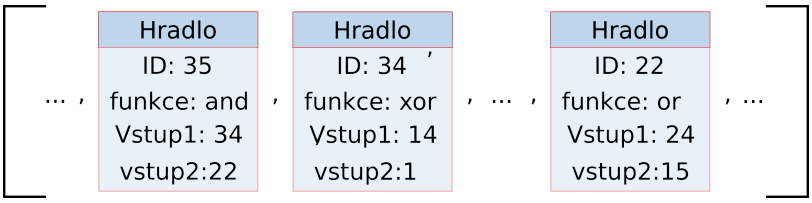
Takže například vstup *"(and(xor()())(or()()))"* bude odpovídat obvodu, tak jak je popsáno na obrázku 7.2.



Obrázek 7.2: **Obvod vstupu** Vstupní řetězec a obvod, který popisuje. Jednotlivé části obvodu a řetězce jsou barevně odlišeny tak, aby bylo zřejmé, která část řetězce popisuje, kterou část obvodu

7.3.2 Algoritmus pro vyhledání podobvodu

Pro snazší pochopení jak algoritmus vyhledávání podobvodu funguje, na obrázku 7.3 ilustrují podobu uložení obvodu v mém kódu. Jedná se o stejný obvod jako v obrázku 7.2.



Obrázek 7.3: **Seznam hradel** - ilustrace podoby atributu list_of_gates, třídy Circuit. V tomto seznamu, jsou uložena všechna hradla obvodu. Na obrázku jsou ukázány jen tři tak, aby odpovídala obvodu z obrázku 7.2.

Vyhledávání podobvodů, stejně jako počítaní použitých hradel, probíhá od konce, od výstupů. Hradla referuji zpětně na svoje vstupy pomocí id hradel.

Funkce *najit_podobvod* prvně zpracuje vstup. Poté iteruje všemi použitými hradly a pro každé zavolá funkci *porovnat_obvod*. Tato funkce porovnává hledaný podobvod s podob-

vodem, který začíná daným hradlem. Podobvody jsou porovnávány rekurzivně a nezávisle na tom, jestli jsou vstupy hradla prohozeny.

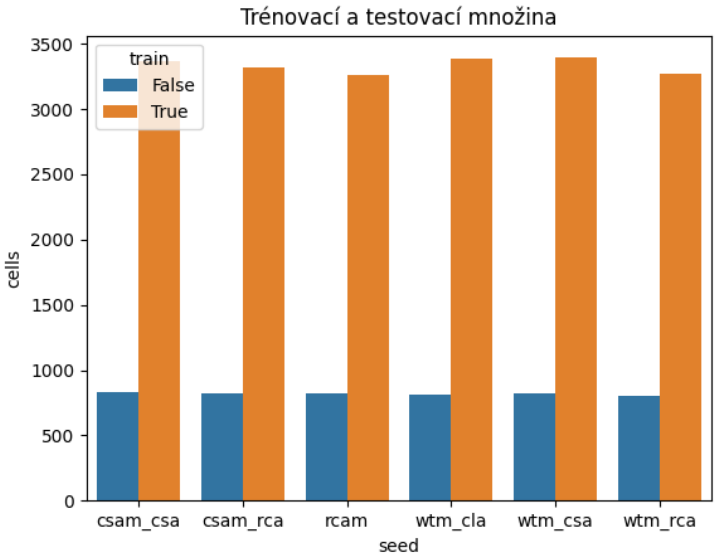
```
   Input: Řetězec popisující podobvod
   Output: Počet výskytů podobvodu
1  Function najit_podobvod(hledany_obvod):
2      zpracovaný_obvod = parse(hledany_obvod)
3      počet_výskytů = 0
4      for hradlo ∈ self.list_hradel do
5          if není použitý v obvodu then
6              continue
7          if porovnat_obvod(hradlo, hledan_obvod) then
8              počet_výskytů = počet_výskytů + 1
9      return počet_výskytů
```

```
1  Function porovnat_obvod(pozadovany_obvod, id_hradla):
2      if pozadovany_obvod je prazdny then
3          return True
4      else if id hradla odkazuje na vstup obvodu then
5          return False
6      hradlo = načti hradlo s id_hradla
7      if hradlo.funkce == pozadovany_obvod[0] then
8          return (porovnat_obvod(hradlo.vstup1, pozadovany_obvod[1]) and
            porovnat_obvod(hradlo.vstup2, pozadovany_obvod[2]))
9          or
10         (porovnat_obvod(hradlo.vstup1, pozadovany_obvod[2]) and
            porovnat_obvod(hradlo.vstup2, pozadovany_obvod[1]))
11     else
12         return False
```

Všechny podobvody, které jsem hledal jsou v příloze A.

7.4 Trénovací a testovací množina

Násobičky jsem rozdělil na trénovací čast, která obshaje 20 000 násobiček a testovací část s 4 912 násobičkama.Počet obvodů, dle rodiče, přibližně odpovídá, viz graf 7.4.



Obrázek 7.4: **Rozdělení na trénovací a testovací množinu**

Kapitola 8

Výsledky experimentů

Mým cílem bylo klasifikovat obvody do třídy určené rodičovským obvodem na základě příznaků vycházejících z jejich chromozomu, respektive získat charakteristiky, které pomůžou daný obvod zařadit. Mezi příznaky, které jsem získal, jsou hlavně počty jednotlivých hradel a 16 specifických podobvodů, viz graf 8.2, jejichž podoba je v příloze A a jejich počty jsou vizualizovány na krabicovém grafu 8.1 v závislosti na rodiči. Průměrné počty hradel dle rodičovského obvodu jsou na grafu 8.2 Využil jsem k tomu tři různé klasifikátory - Random forest, Support vector machine a multilayer perceptron.

- RandomForest, který využívá 100 rozhodovacích stromů o maximální hloubce 10.
- Support-vector-machine využívající kernel rbf - radial basis function, počet dimenzí dat, nad kterými SVM pracuje, byl snížen pomocí PCA.
- Multi-layer perceptron (dále jen MLP) s jednou skrytou vrstvou o velikosti 100. Aktivační vrstva využívá funkci relu a solver adam, viz podrobněji v dokumentaci [1].

8.1 Podobvody

V tabulkách 8.1, 8.2 a 8.3 je počet výskytu daných podobvodů v rodičovských násobičkách.

Tabulka 8.1: Část 1 - počet jednotlivých typů podobvodů, dle rodičovského obvodu.

seed	used_cells	cgp_cells	podobvod1	podobvod2	podobvod3	podobvod4
wtm_cla	418	490	12	20	0	43
csam_csa	347	347	0	13	0	13
csam_rca	320	320	0	13	0	13
wtm_csa	396	418	12	20	0	20
rcam	320	320	0	8	0	8
wtm_rca	332	333	12	20	0	20

8.2 Klasifikace

Vhledem k tomu, že klasifikuji do 6 tříd, tak nejhorší možná úspěšnost klasifikace by neměla být menší než 1/6. Úspěšnost bude zaokrouhlená na 4 desetinná místa. Hodnota značící

[1] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Tabulka 8.2: Část 2 - počet jednotlivých typů podobvodů, dle rodičovského obvodu.

podobvod5	podobvod6	podobvod7	podobvod8	podobvod9	podobvod10
63	16	0	0	0	0
53	0	0	0	1	2
48	0	0	0	0	0
59	0	0	0	1	6
48	0	0	0	0	0
47	0	0	0	0	0

Tabulka 8.3: Část 3 - počet jednotlivých typů podobvodů, dle rodičovského obvodu.

podobvod11	podobvod12	podobvod13	podobvod14	podobvod15	podobvod16
0	26	0	4	10	6
0	44	0	3	0	2
0	41	0	4	0	1
0	36	0	12	0	2
0	41	0	0	0	0
0	27	0	11	0	0

úspěšnost je průměrná úspěšnost, která se počítá dle vztahu 8.1, kde y seznam správných tříd vzorků a $\hat{y}$ je seznam odpovídacích hodnot předpovězených klasifikátorem.

$$\texttt{přesnost}(y, \hat{y}) = \frac{1}{n_{\text{vzorků}}} \sum_{i=0}^{n_{\text{vzorků}}-1} 1(\hat{y}_i = y_i) \tag{8.1}$$

8.2.1 Klasifikace dle počtu hradel

Průměrný počet hradel daného typu, dle rodičovského obvodu je zobrazen na grafu 8.2. V tabulce 8.4 jsou výsledky klasifikace.

Tabulka 8.4: Úspěšnost klasifikace dle počtu hradel určitého typu

Klasifikátor	úspěšnost
RandomForest	0.7477
SVM	0.7361
MLP	0.7575

8.2.2 Klasifikace dle počtu jednotlivých podobvodů

Podobvody, na jejichž základě probíhala klasifikace, jsou v příloze A. Výsledky klasifikace, dle těchto podobvodů je v tabulce 8.5

8.2.3 Klasifikace dle počtu jednotlivých podobvodů a počtu hradel

Úspěšnost klasifikace na základě počtu hradel a podobvodů v tabulce 8.6

Tabulka 8.5: Úspěšnost klasifikace dle výskytu podobvodů

Klasifikátor	úspěšnost
RandomForest	0,7477
SVM	0,7361
MLP	0,7575

Tabulka 8.6: Úspěšnost klasifikace dle počtu hradel určitého typu a výskytu podobvodů.

Klasifikátor	úspěšnost
RandomForest	0,7679
SVM	0,7390
MLP	0,7675

8.2.4 Klasifikace dle informací známých z knihovny

Kromě údajů z knihovny tam je navíc velikost souboru obsahující chromozom a jeho velikost po kompresi (zip). Celý seznam využitých pro klasifikaci - cells, mae, pdk45_area, pdk45_cells, pdk45_delay, pdk45_pwr, seed, wce, wcre%, file_size, file_compressed, used_cells. Celkem tedy klasifikátor využívá 11 parametrů.

Tabulka 8.7: Úspěšnost klasifikace dle parametrů dostupných z knihovny obvodů.

Klasifikátor	úspěšnost
RandomForest	0,9297
SVM	0,8713
MLP	0,9311

8.2.5 Klasifikace dle všech charakteristik dohromady

Kombinuje všechny parametry z předchozí sekce v kombinaci s podobvody a počtem jednotlivých hradel. Celkem má tedy klasifikátor 35 vstupů.

Tabulka 8.8: Úspěšnost klasifikace, dle všech předchozích parametrů dohromady.

Klasifikátor	úspěšnost
RandomForest	0,9651
SVM	0,8912
MLP	0,9529

8.2.6 Úspěšnost klasifikace dle podobvodů - rozděleno dle rodiče

V grafu 8.9 je ukázána úspěšnost klasifikace pro obvody z jednotlivých tříd.

8.2.7 Porovnaní klasifikátoru dle času tréninku

Porovnání času tréninku klasifikátorů na 20 000 obvodech je v tabulce 8.10. Čas je měřen pomocí příkazu *time* na operačním systému Ubuntu 20, použitá hodnota je *cpusystemtime*.

Tabulka 8.9: Úspěšnost klasifikace dle výskytů podobvodů dle každé třídy

Klasifikátor	rcam	wtm_cla	csam_csa	wtm_csa	wtm_rca	csam_rca
RandomForest	0,8693	0,6936	0,8431	0,6504	0,6504	0,7760
MLP	0,9768	0,7794	0,7233	0,6359	0,6554	0,7735

Tabulka 8.10: Čas tréninku klasifikátoru.

Klasifikátor	čas [sekund]
RandomForest	0,274
SVM	0,396
MLP	85.3678

8.2.8 Závěr klasifikace

Na začátku shrnu poznatky k jednotlivým klasifikátorům, až poté se dostanu k samotným datům. Support-vector-machine klasifikátor poskytuje konzistentně horší výsledky než zbylé klasifikátory a je pomalejší na natrénování než RandomForest. MLP a RandomForest poskytují často velmi podobné výsledky, ale RandomForest byl opět výrazně rychlejší na natrénování. MLP pravděpodobně má větší potenciál na zlepšení po optimalizaci parametrů (počet a velikost skrytých vrstev například), použil jsem výchozí nastavení v knihovně Sickit-learn.

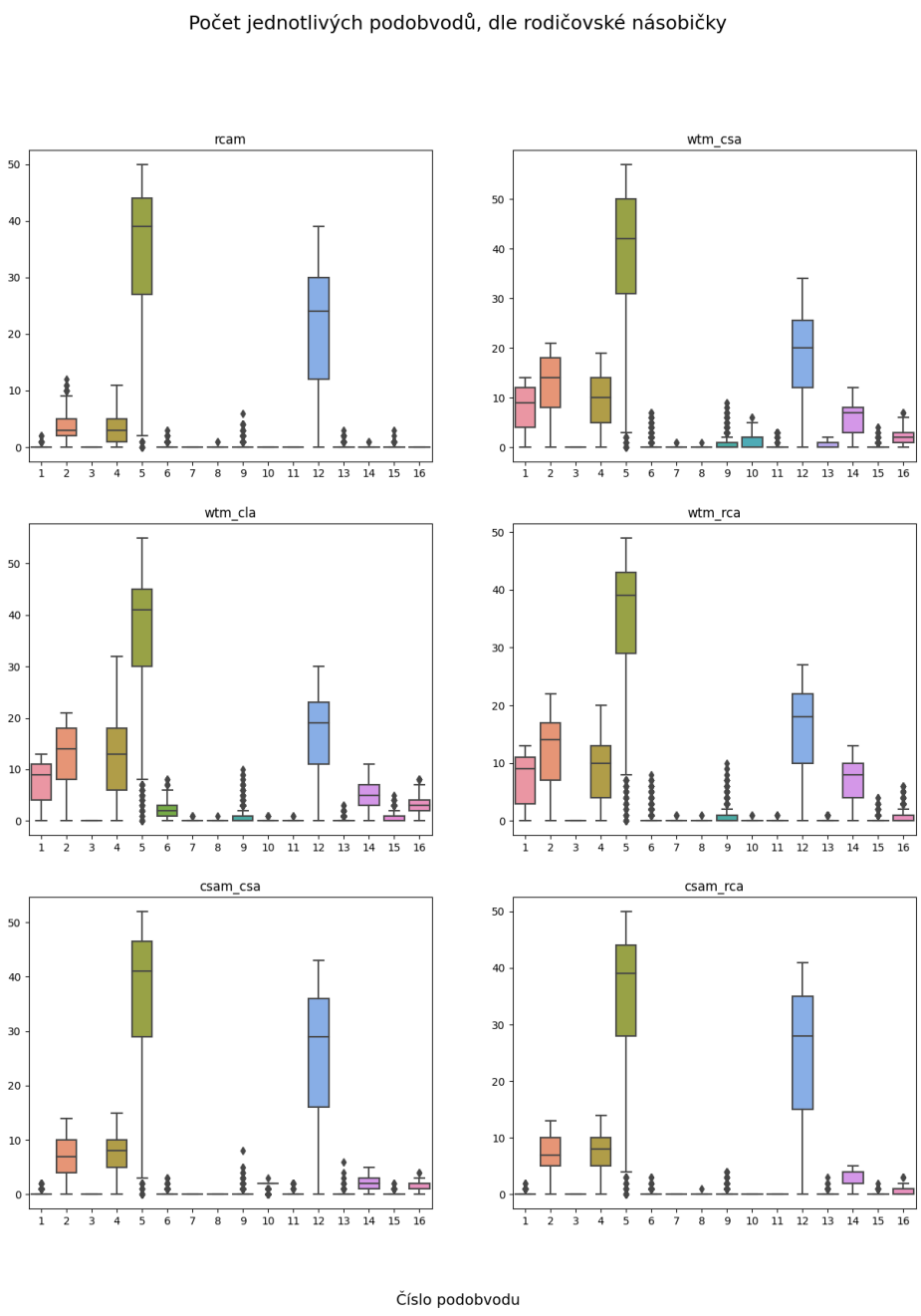
Nejlepší úspěšnost mělo využití charakteristik již obsažených v knihovně k jednotlivým obvodům. Přisuzuji to především rozdílům mezi architekturou rodičovských obvodu, které byly optimalizovány pro odlišné parametry (jako jsou zpoždění,spotřeba, nebo plocha).

Zajímavější z pohledu mé práce je klasifikace na základě podobvodů a počtu hradel. Klasifikace na základě podobvodů byla značně úspěšná. Podle dat v sekci 8.2.6 je vidět, že oba klasifikátory lépe klasifikují třídy rcam a csam_csa. Výsledky pro zbytek tříd by pravděpodobně bylo možné zlepšit volbou jiných, či více parametrů. Hledaní podobvodů, speciálně složitějších (jako jsou podobvody 10-16) je výpočetně náročné (na mém PC s core i5-4460 zpracování všech obvodů trvalo 83 minut) a nelze předem určit, který jak moc pomůže a proto jsem zůstal u aktuálních 16. Po nahlédnuti do grafu 8.1 lze spekulovat, jaké podobvody mohly být pro klasifikaci nejdůležitější, například podobvod č. 7 se vyskytuje jen v třech třídách a ve třídě csam_rca se vůbec nevyskytuji podobvody č. 7, 10, 11.

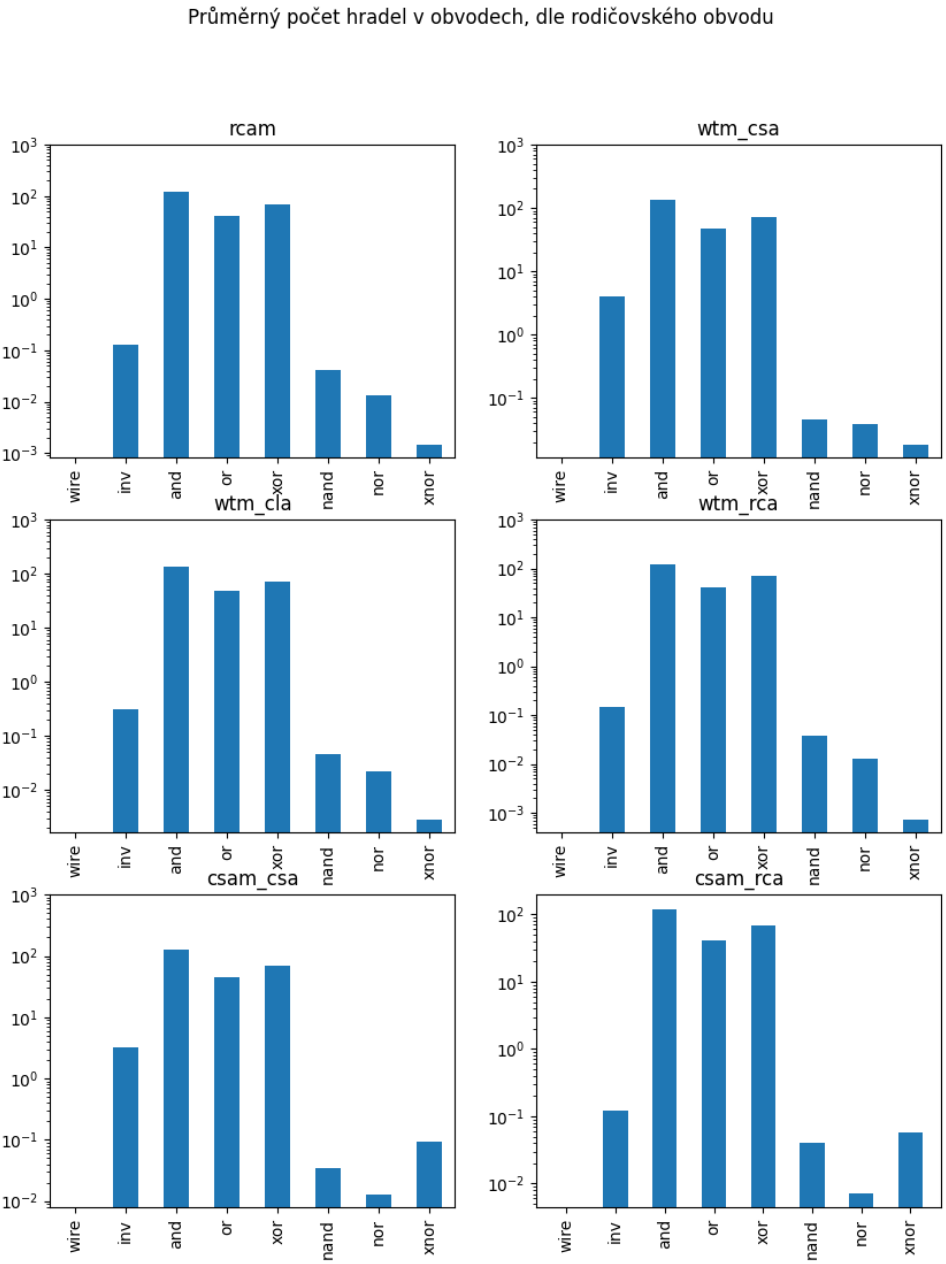
8.2.9 Další poznatky

- V knihovně nejsou dva stejné obvody, které by měli stejného rodiče.
- Počet jednotlivých hradel má malý (ne však zanedbatelný) vliv na klasifikaci obvodu.
- Klasifikace rodiče do třídy dle rodiče měla 100% úspěšnost pro klasifikátory RF a MLP, 1 z 6 byl špatně klasifikovaný v případě SVM.
- Podobvod číslo 5 byl nejčastější z jednoduchých, číslo 12 nejčastější z těch složitějších. Podoba podobvodů je ukázána na obrázku 8.3.
- Zároveň výskyt těchto podobvodů koreluje s chybovostí a plochou obvodu (větší obvody mají více podobvodů a jsou přesnější) viz příloha D.

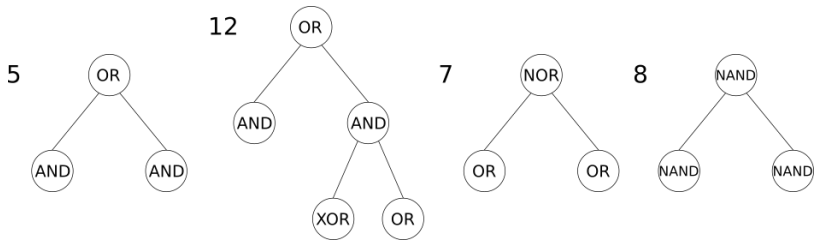
- Podobvody 7 a 8 se nevyskytují ani v jednom z rodičovských obvodů, ale objevily se v několika obvodech z knihovny. Podoba podobvodů je ukázána na obrázku 8.3.



Obrázek 8.1: **Počet podobvodů** rozděleno dle rodičovského obvodu.



Obrázek 8.2: **Průměrný počet hradel** v závislosti na rodičovském obvodu.



Obrázek 8.3: **Vybrané podobvody.**

Kapitola 9

Závěr

V této práci byly nastudovány a popsány základní informace o využití aproximace v obvodech, přehled metod strojového učení pro klasifikaci dat a obecné informace o evolučních algoritmech se zaměřením na kartézské genetické programování. Cílem práce bylo analyzovat evolučně navržené obvody a najít ukazatele příbuznosti. Cíl byl práce byl splněn, zvolené ukazatele příbuznosti se ukázaly jako dostatečné pro úspěšnou klasifikaci obvodů do tříd dle rodičovského obvodu.

Jedním ukazatelem příbuznosti byl počet použitých hradel a počet použitých hradel, dle jejich funkce. Dalším bylo hledání 16 podobvodů, které se ukázaly jako silné ukazatele příbuznosti.Podařilo se klasifikovat obvody dle podobvodů s úspěšnosti až 75%. Když se k podobvodům přidaly ještě počty jednotlivých hradel, úspěšnost stoupla až na 77%. Nejvyšší úspěšnosti dosáhl klasifikátor využívající údaje vycházející ze simulování obvodů a běžných metrik obvodů poskytnutých v knihovně a to 93%.

Práce mi umožnila se nepřímo a přesto docela podrobně seznámit s evolučními algoritmy a jejich možnostmi. Také zkušenost s knihovnami Pandas a Sickit-learn pro práci s daty a jejich analýzu jistě využiji v budoucnu.

Na práci je možné navázat několika způsoby. Hledat více podobvodů, nebo objevit nějaký specifičtější podobvod by mohlo zlepšit výsledky. Vícevrstvý perceptron měl nejlepší výsledky, ale trval dlouho natrénovat, experimentování s parametry klasifikátoru by mohlo proces urychlit i zkvalitnit. Další možností by bylo vyzkoušet výsledky z této práce na jiných obvodech.

Literatura

[1] Agrawal, A., Choi, J., Gopalakrishnan, K., Gupta, S., Nair, R. et al. Approximate computing: Challenges and opportunities. In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, s. 1–8. DOI: 10.1109/ICRC.2016.7738674.

[2] Bentley, P. An Introduction to Evolutionary Design by Computers. *Evolutionary Design by Computers*. Leden 1999.

[3] Bento, C. *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis* [online]. 2021 [cit. 2022-5-05]. Dostupné z: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.

[4] Gandhi, R. *Support Vector Machine — Introduction to Machine Learning Algorithms* [online]. 2018 [cit. 2022-5-05]. Dostupné z: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[5] Jaadi, Z. *A Step-by-Step Explanation of Principal Component Analysis (PCA)* [online]. 2021 [cit. 2022-5-05]. Dostupné z: https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[6] Jarosz, Q. *Neuron*. 2009. DOI: 10.1109/ACCESS.2019.2958605. Dostupné z: https://cs.wikipedia.org/wiki/Neuron#/media/Soubor:Neuron_Hand-tuned.svg.

[7] Khadra, M. A. B. An introduction to approximate computing. *ArXiv*. 2017, abs/1711.06115.

[8] Kulkarni, P., Gupta, P. a Ercegovac, M. Trading Accuracy for Power in a Multiplier Architecture. *J. Low Power Electronics*. Prosinec 2011, sv. 7, s. 490–501. DOI: 10.1166/jolpe.2011.1157.

[9] Miller, J. F. *Cartesian genetic programming*. New York: Springer Berlin Heidelberg, 2011. Natural computing series. ISBN 978-3-642-17309-7.

[10] Mrazek, V., Hrbacek, R., Vasicek, Z. a Sekanina, L. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 2017, s. 258–261. DOI: 10.23919/DATE.2017.7926993.

[11] Mrazek, V., Vasicek, Z. a Sekanina, L. Design of Quality-Configurable Approximate Multipliers Suitable for Dynamic Environment. In:. Srpen 2018, s. 264–271. DOI: 10.1109/AHS.2018.8541479.

[12] Pant, A. *Introduction to Machine Learning for Beginners* [online]. 2019. Dostupné z: https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08",cited={2022-5-05}.

[13] Sekanina, L. Introduction to approximate computing: Embedded tutorial. In: *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2016, s. 1–6. DOI: 10.1109/DDECS.2016.7482460.

[14] Shlens, J. A Tutorial on Principal Component Analysis. *ArXiv*. 2014, abs/1404.1100.

[15] Sodhi, P., Awasthi, N. a Sharma, V. Introduction to Machine Learning and Its Basic Application in Python. *Proceedings of 10th International Conference on Digital Strategies for Organizational Success*. 2019. Dostupné z: https://ssrn.com/abstract=3323796.

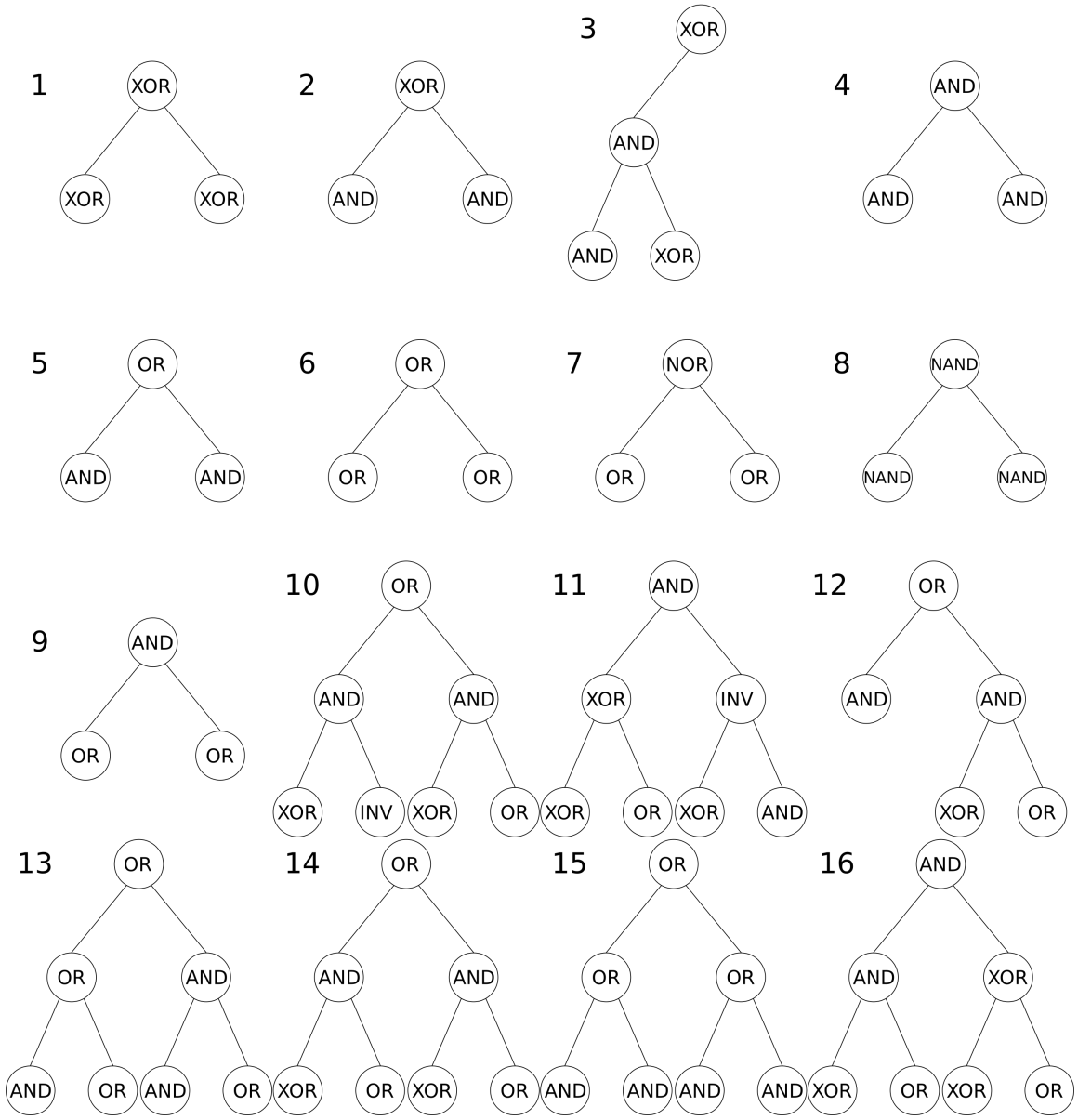
[16] Sruthi, E. R. *Understanding Random Forest* [online]. 2021 [cit. 2022-5-05]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/.

[17] Vasicek, Z. Formal Methods for Exact Analysis of Approximate Circuits. *IEEE Access*. 2019, sv. 7, s. 177309–177331. DOI: 10.1109/ACCESS.2019.2958605.

[18] Yiu, T. *Understanding Random Forest* [online]. 2019 [cit. 2022-5-05]. Dostupné z: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

Příloha A

Podobvody


Hledané pod-obvody:
1
XOR
XOR
XOR
2
XOR
AND
AND
3
XOR
AND
AND
XOR
4
AND
AND
AND
5
OR
AND
AND
6
OR
OR
OR
7
NOR
OR
OR
8
NAND
NAND
NAND
9
AND
OR
OR
10
OR
AND
AND
XOR
INV
XOR
OR
11
AND
XOR
INV
XOR
OR
XOR
AND
12
OR
AND
AND
XOR
OR
13
OR
OR
AND
AND
OR
AND
OR
14
OR
AND
AND
XOR
OR
XOR
OR
15
OR
OR
OR
AND
AND
AND
AND
16
AND
AND
XOR
XOR
OR
XOR
OR

Obrázek A.1:

Příloha B

Obsah přiloženého paměťového média

Obsah:

- bp_script.py - skript implementující zpracování obvodů,
- graphs.py - skript pro vyhodnocení zpracovaných dat o obvodech,
- readme.txt - popis, jak výše zmíněné skripty spustit,
- dataframe.pkl - soubor vygenerovaný skriptem bp_script.py obsahující dataframe df s obvody,
- dataframe2.pkl - soubor vygenerovaný skriptem bp_script.py obsahující dataframe df2 s rodicovskymi obvody,
- xkrejc68_latex - složka se zdrojovými soubory latexu,
- xkrejc68.pdf - pdf technické zprávy.

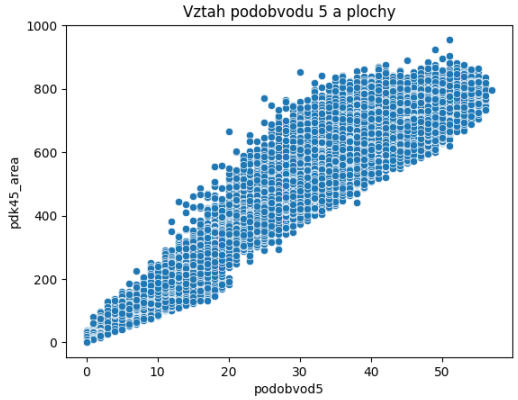
Příloha C

Spuštění

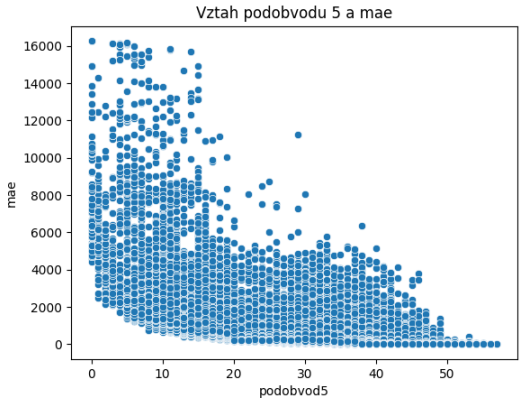
- bp_script.py ve výchozím stavu vyžaduje, aby soubor cgp-approx14ep.json a všechny *.chr soubory byly ve stejném adresáři, nebo změnit cestu k souborům ve skriptu
- skript vytvoří soubor dataframe.pkl, který se poté načte skriptem graphs.py. Doporučuji první dva kroky přeskočit, soubor dataframe.pkl je přiložený.
- graphs.py je třeba spustit s -7 pro vygenerování grafů, nebo s -1 pro trénování, testování a vygenerování dat o klasifikaci
- Pro práci s daty je možné skript graphs.py spustit v interaktivním módu, DataFrame se všemi daty je v proměnné df.

Příloha D

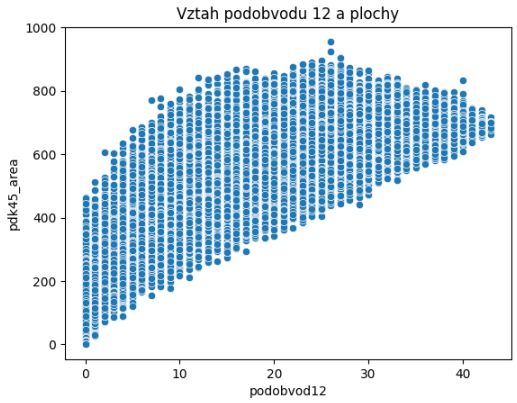
Závislost vlastností obvodů na podobvodech



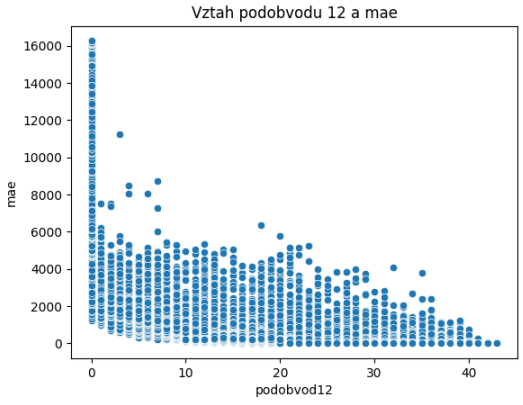
Obrázek D.1: Závislost počtu podobvodů 5 a plochy obvodu.



Obrázek D.2: Závislost počtu podobvodů 5 a mae - mean absolute error.



Obrázek D.3: Závislost počtu podobvodů 12 a plochy obvodu



Obrázek D.4: Závislost počtu podobvodů 12 a mae - mean absolute error.