ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta dopravní

Bakalářská práce

Využití evolučních technik při hledání parametrů dopravních systémů

Autor

Valerii Gopak

Vedoucí práce

doc. Ing. Vít Fábera Ph.D.

2015

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
d ě k a n
Konviktská 20, 110 00 Praha 1

K614..... Ústav aplikované informatiky v dopravě

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Valerii Gopak

Kód studijního programu a studijní obor studenta:

B 3710 – AUT – Automatizace a informatika

Název tématu (česky): **Využití evolučních technik při hledání parametrů dopravních systémů**

Název tématu (anglicky): Using Evolutionary Techniques in Search of Parameters of Transportation systems

Zásady pro vypracování

Při zpracování bakalářské práce se řiďte osnovou uvedenou v následujících bodech:

- prostudujte problematiku genetických algoritmů, genetického programování a gramatické evoluce
- ověřte nasazení vybraných evolučních technik pro hledání parametrů diferenciálních rovnic popisující dopravní aplikace
- porovnejte vhodnost jednotlivých technik

Rozsah grafických prací: podle pokynů vedoucího práce

Rozsah průvodní zprávy: minimálně 35 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)

Seznam odborné literatury:
Mařík a kol.: Umělá inteligence 3,4, Academia, 2003

Zelinka I. a kol.: Evoluční výpočetní techniky, BEN, 2009

Brockwell P., Davis R. Time Series: Theory and Methods, Springer Verlag, 1991

Vedoucí bakalářské práce: **doc. Ing. Vít Fábera, Ph.D.**

Datum zadání bakalářské práce: **20. října 2014**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání bakalářské práce: **24. srpna 2015**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

L. S.

doc. Dr. Ing. Tomáš Brandejský
vedoucí
Ústavu aplikované informatiky v dopravě

prof. Dr. Ing. Miroslav Svítek
děkan fakulty

Potvrzuji převzetí zadání bakalářské práce.

Valerii Gopak
jméno a podpis studenta

V Praze dne ..... 20. října 2014

Poděkování

Rád bych touto cestou vyjádřil poděkování doc. Ing. Vítu Fáberovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Prohlášení

„Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.“

„Nemám závažný důvod proti užívání tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).“

V Praze dne ..... podpis.....

Autor: Gopak Valerii

Název práce: Využití evolučních technik při hledání parametrů dopravních systémů

Vysoká škola: Dopravní fakulta ČVUT v Praze

Ústav: K614 Ústav aplikované informatiky v dopravě

Datum:

Klíčová slova

Evoluční techniky, genetické programování, aproximace signálu, regrese

Abstrakt

Cílem této bakalářské práce je modifikace již existujícího algoritmu genetického programování. Výsledkem práce je upravený počítačový program s rozšířenou funkcionalitou oproti původnímu algoritmu. Při testování programu byly použity reálné signály elektroencefalogramu. Na základě provedených experimentů jsou posouzeny teoretické možnosti aproximace dané křivky diferenciální rovnicí pomocí genetického programování.

Author: Gopak Valerii

Title: Using Evolutionary Techniques in Search of Parameters of Transportation systems

College: Faculty of transportation Sciences, CTU in Prague

Department: K614 Department of Applied Informatics in Transport

Date:

Keywords

Evolutionary Techniques, Genetic programming, Signal Approximation, Regression

Abstract

The goal of my bachelor thesis is to modify an existing genetic programming algorithm. The result of the thesis is an adjusted application with an extended functionality of the original algorithm. The real electroencephalogram signals are used during application testing. On the basis of the experiments results, a theoretical possibility to generate a differential equation of the curve is assessed.

Obsah

Úvod 8

1. Evoluční výpočetní techniky 9

1.1 Genetické algoritmy 11

1.1.1 Selekce 11

1.1.2 Rekombinační operátory 12

1.1.3 Příklad GA 13

1.2 Gramatická evoluce 16

1.2.1 Backus-Naurova forma 16

1.2.2 Rekombinační operátory v GE 19

1.3 Genetické programování 20

1.3.1 Reprezentace jedinců 21

1.3.2 Rekombinační operátory v GP 22

2. Numerické metody řešení diferenciálních rovnic 23

2.1 Eulerova metoda 23

2.2 Metoda Runge-Kutta 24

3. Cíle práce 25

3.1 Problematika 25

3.2 Modifikace algoritmu GP 26

3.3 Řešení 27

3.3.1 Rozšíření třídy GPTree 27

3.3.2 Omezení hloubky křížení 28

3.3.3 Binární čtení dat 28

3.3.4 Práce s daty EEG 29

4. Experimenty 30

4.1 Testování na jednoduchých rovnicích 31

4.2 Testování s využitím trigonometrických funkcí 32

4.3 Testování na rovnici Mathieu 33
4.4 Testování algoritmu s využitím dat EEG 34
4.4.1 Popis signálu EEG 34
4.4.2 Testování 36
5. Závěr 37
6. Seznam použité literatury 39
7. Přílohy 40
7.1 Vzorkování diferenciální rovnice 40
7.2 Zobrazení grafů vstupní a nejlepší generované rovnice 40

Seznam zkratek

V textu byly použity následující zkratky a na příslušném místě použití byly také vysvětleny.

ET – evoluční techniky

EVT – evoluční výpočetní techniky

GA – genetické algoritmy

SGA – standardní genetické algoritmy

GE – gramatická evoluce

BNF – Backus-Naurova forma

GP – genetické programování

EEG - elektroencefalogram

Úvod

První, kdo přišel s myšlenkou evoluce založené na přirozeném výběru v polovině 19. století, byl britský přírodovědec Charles Darwin. Principy evoluce popsané britským vědcem slouží jako základ evolučních technik. Jedním z prvních, kdo se inspiroval těmito přírodními jevy a snažil se je uplatnit v technické praxi, byla skupina studentů univerzity v Berlíně. Budoucí inženýři, kteří se také zabývali dopravou, chtěli aplikovat jeden z principů ET při návrhu konstrukce převodovky. Myšlenka spočívala v náhodné kombinaci dvou již existujících konstrukcí pro návrh nové převodovky, s lepšími technickými vlastnostmi.[1] Po rozvoji výpočetní techniky našly poměrně široké uplatnění Darwinovy myšlenky v informatice. Cílem této bakalářské práce je seznámení s různými druhy EVT, detailnější přehled genetického programování a jeho aplikace při řešení konkrétní úlohy.

Práce navazuje na diplomovou práci [5], zabývá se stejnou problematikou – aproximaci EEG signálů pomoci genetického programování. Jejím základem je algoritmus vzniklý v rámci diplomové práce, který byl implementován v programovacím jazyce C++. Vstupem pro něj je soubor obsahující navzorkovanou křivku a počáteční podmínky, výstupem je diferenciální rovnice popisující křivku.

Jedním z cílů práce je úprava tohoto algoritmu, přidání funkcí pro zvětšení jeho účinnosti. Rozšíření funkcionality mělo by umožnit práce s reálným EEG signálem, na kterém bude program testován. Na základě výsledků experimentů bude posouzeno, zda je aproximace diferenciální rovnici pomocí genetického programování vhodnou technikou pro řešení podobných úloh.

[1] Zdrojem historické poznámky je kniha MAŘÍK, V., O. ŠTĚPÁNKOVÁ, J. LAŽANSKÝ a kol.: *Umělá inteligence (III)*. Praha, Academia, 2001.

1. Evoluční výpočetní techniky

V současné době se pro řízení a predikci chování dopravy čím dále častěji používá umělá inteligence. Vzhledem k tomu, že jde o hodně obsáhlé téma, které přesahuje rámce bakalářské práce, je jejím cílem seznámit se pouze s jedním z účinných nástrojů umělé inteligence, s evolučními výpočetními technikami (EVT).

Přestože v dnešní době lidstvo udělalo velký pokrok v oblasti výpočetních výkonů, ne všechny úlohy můžeme vyřešit během nám vyhovující doby. Například vyřešení úlohy obchodního cestujícího s rozsáhlou sítí míst, které musí cestující navštívit za nejkratší dobu s nejmenším počtem ujetých kilometrů. Řešení této úlohy jednou z exaktních metod[2] by mohlo trvat několik týdnu i let, i kdyby data zpracovával takzvaný superpočítač. Daná úloha patří mezi úplné NP problémy, což jsou nedeterministicky polynomiální problémy, které nelze vyřešit v polynomiálním čase. To znamená, že pro vyřešení tohoto problému nemůžeme prakticky prohledávat celou množinu možných řešení. Jiný přístup mají heuristiky, které řeší problém tak, že se ve většině případů na základě logických úvah omezuje prostor prohledávání. Mezi heuristické algoritmy patří i EVT.

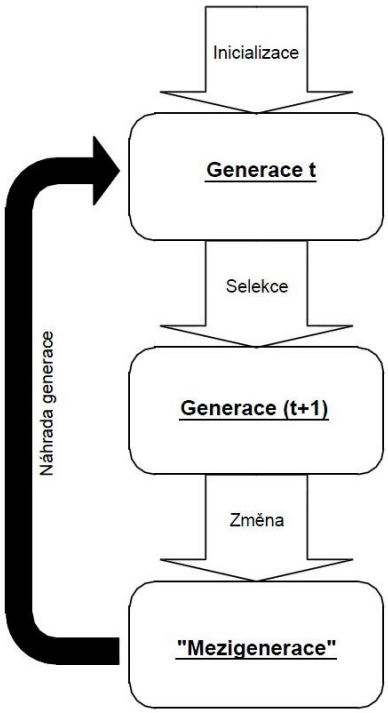
EVT řadíme ke stochastickým algoritmům, jejichž vlastností je práce s tzv. jedinci a jejich postupná evoluce. Pod pojmem jedinec se nejčastěji skrývá zakódované potenciální řešení konkrétní úlohy. Nepracuje se pouze s jedním jedincem, ale s celou populací jedinců vyjádřenou stejnou datovou strukturou. Algoritmus začíná téměř vždycky sestavením (vygenerováním) počáteční generace jedinců. Daný proces většinou se provádí náhodně s určitými omezeními. Dále v cyklu následuje ohodnocování kvality všech jedinců generace. Kvalitu jedince měříme výpočtem tzv. fitness (kriteriální) funkce (zároveň můžeme vyhledat nejlepšího jedince a statisticky vyhodnotit danou generaci). Poté probíhá selekce jedinců, což znamená výběr jedinců do nové generace obvykle na základě jejich kvalitativní (fitness) hodnoty. Navíc neurčujeme, zda bude konkrétní jedinec vybrán do další generace, ale pouze pravděpodobnost jeho přežíti. Bude-li jedinec ohodnocen jako kvalitnější, znamená to, že má větší pravděpodobnost postupu (výběru) do následující generace. Tím teoreticky dosáhneme postupného zlepšení průměru ohodnocení generace. Následující fáze – změna, zahrnuje v sobě různé rekombinační operátory použité v závislosti na reprezentaci jedinců. Dva základní typy však se používají nezávisle na reprezentaci, jsou to mutace a křížení.

Mutace je unární operace pracující pouze s jedním jedincem, která podle příslušných pravidel mění jeho část.

[2] Exaktními nazýváme metody hrubé síly, které pro nalezení řešení zkoumají všechny permutace

Křížení tvoří jednoho či dva jedince (potomky) z nejméně dvou jedinců, takzvaných rodičů. Běžně generujeme potomky ze dvou rodičů a operace je tím pádem binární.

Postup algoritmu EVT je zobrazen níže.



Obr. 1.1: Schéma algoritmu EVT

Po změně generace se objevuje pojem „mezigenerace“ – je to generace, ve které proběhl proces selekce a je obohacená o jedinci vygenerované rekombinačními operátory. Tudíž obsahuje jak „rodiče“ tak i „potomky“. Za tím následuje proces náhrady generace, je to část algoritmu, jejíž součásti je vývojová strategie. Pod tímto pojmem rozumíme výběr jedinců ze stávající množiny, kteří budou v další iteraci tvořit novou populaci. Zpravidla je velikost populace po dobu běhu algoritmu konstantní.

Nejrozšířenějšími typy vývojové strategie jsou: generační a postupné. Podstatou generačního typu je úplná náhrada jedné populace populací následující. Při typu postupném mění se pouze část populace, která ve výsledku má v sobě jak rodiče, tak i potomky.

Všechny algoritmy EVT ukončují svou iterativní činnost na základě zastavovacího pravidla. Obecně podmínkou zastavení algoritmu je buď výsledek, který splní naše požadavky nebo maximální počet generací, respektive čas, během kterého musí být úloha vyřešena. Například máme za úkol najít nejkratší cestu mezi vrcholy *A* a *B*, zastavovací pravidlo můžeme nadefinovat, buď jako počet kilometrů, se kterým budeme spokojeni, nebo maximální čas běhu algoritmu, který jsme ochotni čekat na výsledek.

1.1 Genetické algoritmy

Významnou odlišností GA od ostatních EVT je reprezentace jedinců ve tvaru řetězců konečné délky, které se nazývají chromozomy. Je to posloupnost symbolů, ve které se každá pozice podle terminologii jmenuje alela. Element takového řetězce se nazývá gen. Genetickou informaci kódující chromozomy nazýváme genotyp a skutečná informace zakódovaná v konkrétním chromozomu je fenotyp.

Chromozomy fixní délky obsahující geny tvořené dvojkovou soustavou jsou základem SGA (standardní genetický algoritmus). Chromozom představuje zakódované řešení nějaké úlohy. Kvůli dvojkové soustavě použité v SGA pro reprezentaci jedinců musí algoritmus obsahovat dvě funkce navíc, a to jsou funkce kódování a dekódování. Podle složitosti řešení a hodnot, kterých může chromozom nabývat, musíme zvážit, zda je dvojkové kódování vhodné. Při náročném řešení se silně omezeným intervalem nabývajících hodnot chromozomu může funkce kódování stát zbytečným zdrojem chyb a zpomalit běh algoritmu, proto se volí kódování genů celými, resp. reálnými čísly.

GA se přidržují vývojového schéma zobrazeného výše, proto po vygenerování populací následuje její ohodnocení, neboli přiřazení kvality (fitness). Funkce hodnoticí jednotlivé jedince je zobrazením množiny jedinců do množiny reálných čísel.

1.1.1 Selekce

Účel selekce je dávat přednost jedincům s vyšší kvalitou. Popíšeme si několik nejrozšířenějších metod selekce.

Ruletová selekce a její definitivní rys spočívá v pravděpodobnosti přežití jedince, která je přímo úměrná jeho kvalitě. Osud jedince závisí na náhodném pokusu, ale čím je jedinec kvalitnější, tím je větší pravděpodobnost, že se dostane do další generace. Ukázalo se však, že toto schéma má řadu problémů, mezi něž patří velké vzorkovací chyby a nutnost mít relativně velkou populace pro správnou funkci [2].

Turnajová selekce je v dnešní době nejpoužívanější selekční strategie. Pro naplnění nové generace provede se turnaj tolikrát, kolik je rozměr populace. Podle předem určeného parametru *n* ze staré generace se vezme *n* chromozomů a nejlepší z nich postoupí do generace nové. Nastavením *n* určujeme selekční tlak – čím větší *n*, tím větší selekční tlak. V případě zvolení velkého *n*, například když *n* se rovná velikosti populace, by se nová generace skládala z nejlepších jedinců a jednalo by se o předčasnou konvergenci.

Pořadová selekce (ranking selection) se vyznačuje omezením předčasné konvergence a odstraněním problému se vzorkováním u malých populací. Na rozdíl od

ruletové selekce pravděpodobnost přežití jedince souvisí pouze s jeho umístěním v posloupnosti, ve které jsou chromozomy seřazeny podle kvality. Příkladem může být posloupnost skládající se ze třech jedinců ohodnocených *f(A)=12*, *f(B)=4* a *f(C)=59*. Pořadová selekce je uspořádá do posloupnosti: *B, A, C* (předpokládáme maximalizaci) a nad nimi proběhne ruletová selekce. Pravděpodobnosti přežití dostaneme následující [3]: *p(B)*= $\frac{1}{6} \approx 17\%$, *p(A)*= $\frac{2}{6} \approx 33\%$ a *p(C)*= $\frac{3}{6} \approx 50\%$, zatímco u ruletové selekce dostaneme *p(B)* = $\frac{4}{75} \approx 5\%$, *p(A)*= $\frac{12}{75} \approx 16\%$, *p(C)* = $\frac{59}{75} \approx 79\%$.

1.1.2 Rekombinační operátory

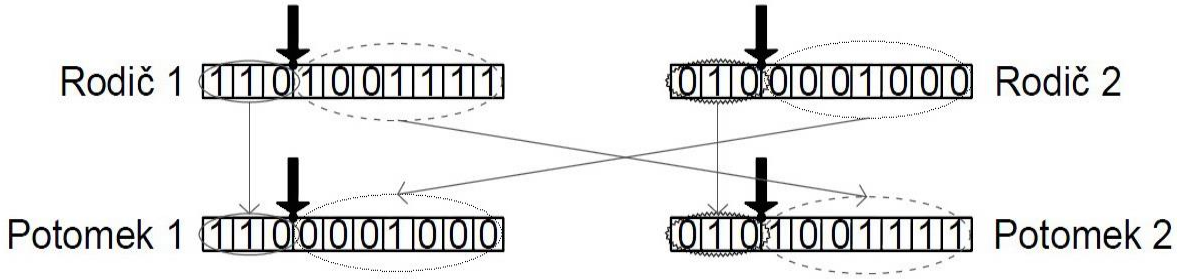
Selekce je důležitý parametr GA, má velký vliv na kvalitu výsledného řešení, ale jeho podstatou je pouze kopírováni některých jedinců do nové generace. K vytváření změn v původně vygenerované populaci slouží změnové neboli rekombinační operátory. K základním rekombinačním operátorům GA patří křížení a mutace. Na rozdíl od selekce jedinci nemusejí být ovlivněni změnovými operátory. Každý jedinec bude zasažen s určitou pravděpodobností. U křížení se nejčastěji jedná o pravděpodobnost v intervalu *0,6* až *1*, pravděpodobnost mutace je vždy výrazně menší než pravděpodobnost křížení, běžně *0,05* až *0,15*. Obecné schéma křížení je zachyceno na obrázku *1.2*.

Obr. 1.2: Křížení v GA

V SGA křížení kombinuje části pouze dvou rodičů, z nichž vytváří dva potomky. Pro určení jedinců, které podléhají křížení, provede se pokus nad každým jedincem generace a s předem nastavenou pravděpodobností bude zařazen do množiny, nad kterou se provede křížení. Proces křížení proběhne tak, že se z této množiny vybírá pár buď náhodně, nebo se definuje jako dva po sobě jdoucí jedinci. V případě lichého počtu se zkrátí množina o jeden prvek. Další možnou variantou implementace křížení je výběr jedinců do mezigenerace, která se mutuje a následovně nahrazuje starou generaci.

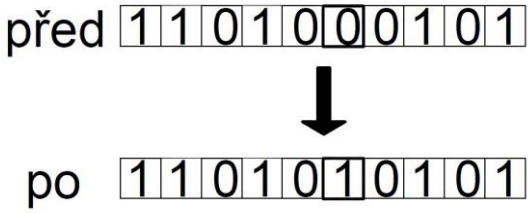
Implementace v SGA, kde jsou jedinci reprezentované bitovou posloupností, se provádí pomocí jednobodového křížení. Bod křížení se zvolí náhodně v jakékoli části posloupnosti na intervalu *1* až *n-1*, kde n je délka posloupnosti (jedince, chromozomu). Po zvolení bodu bude první potomek obsahovat bity ležící vlevo od bodu křížení prvního rodiče, bity vpravo budou převzaty od druhého rodiče. Druhý potomek naopak ve své levé části má

bity zkopírované od druhého rodiče a v pravé od prvního. Na následujícím obrázku je znázorněn tento postup.



Obr. 1.3: Příklad křížení v SGA

Druhým rekombinačním operátorem je mutace. Tento operátor pracuje s jedním jedincem a provádí změnu v jedné alele. Bude-li chromozom zařazen do množiny, nad kterou se provede mutace, náhodně se v něm vybere alela a změní se příslušný gen na opačný 0 na 1, respektive 1 na 0.

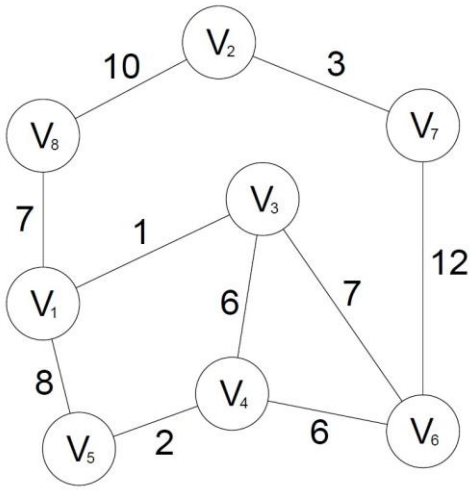


Obr. 1.4: Chromozom před a po mutaci

Používá se také varianta průchodu celého chromozomu každého jedince, kde s určitou pravděpodobnosti je procházený gen invertován, v tomto případě musí být pravděpodobnost mutace výrazně nižší než v předchozím případě (doporučená hodnota $p < 0{,}1$).

1.1.3 Příklad GA

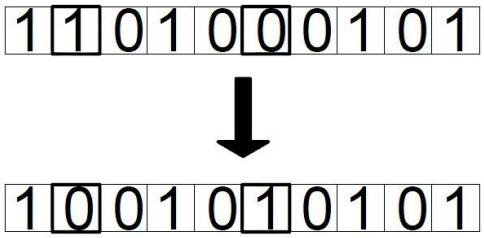
Představme si neorientovaný graf s ohodnocenými hranami (obrázek *1.5*). Úkolem je rozdělení vrcholů do dvou množin tak, aby součet ohodnocení hran spojujících této dvě množiny byl minimální.



Obr. 1.5: Zadání úlohy

Vstupem programu je textový soubor obsahující matici sousednosti, která se načítá do dvourozměrného pole. Prvním krokem GA je generování počáteční generace, náhodně naplníme dvojrozměrné pole nulami a jedničkami, kde počet sloupců odpovídá rozměrem vstupní matici, a počet řádků je roven předem definované velikosti generace. Mezi parametry GA také patří pravděpodobnost křížení, mutace a maximální počet iterací, který je v daném algoritmu zastavovací podmínkou vyjadřující počet vytvořených generací. Vytvořená generace se vyhodnotí pomocí fitness funkce a najde se nejlepší jedinec, na základě těchto údajů vypíše se statistika. Hodnota vrácená funkcí fitness v našem případě je součet ohodnocení hran spojující množiny vrcholů.

Po vyhodnocení nulté generace přichází cyklus vyvíjení s výše zmíněnou zastavovací podmínkou. Cyklus začíná generováním pseudonáhodného čísla z intervalu 0 až 1. V případě, že číslo je menší, než konstanta pravděpodobnosti křížení, je tento rekombinační operátor aplikován. Počet pokusů o křížení dvou jedinců je stejný jako velikost generace. Pro konkrétní úlohu je jednobodové křížení navíc ošetřeno kontrolou počtu nul a jedniček v potomcích. Jestliže počet nul nebo jedniček překračuje polovinu velikosti, jedinec se vyhodnotí jako chybný a přepíše se na jednoho z rodičů. Algoritmus pokračuje vygenerováním pseudonáhodného čísla, které rozhodne, proběhne-li mutace. Proces mutování probíhá následovně: náhodně se vyberou dvě alely a vzájemně si vyměnění svůj obsah, jak je ukázáno na obrázku *1.6*.



Obr. 1.6: Schema mutace v daném příkladě

Mutované jedince a potomky křížení tvoří mezigeneraci, která je dále vyhodnocená. Poté proběhne turnajová selekce s parametrem dva, to znamená, že turnaj proběhne tolikrát, kolik je velikost generace a vybírá se mezi dvěma jedinci. Po úplné náhradě staré generace se nová vyhodnotí a vypíše se statistika. Tím končí vyvíjecí cyklus, po poslední iteraci se nejdříve vypíše nejlepší jedinec ve dvojkové podobě a pak dekódovaný. Princip dekodéru je jednoduchý, jestli na alele *n* je zapsané číslo *0* to znamená, že vrchol s indexem *n* leží v první množině, v opačném případě vrchol patří ke druhé množině. Program, vstupem kterého byl výše uvedený graf a nastavené parametry:

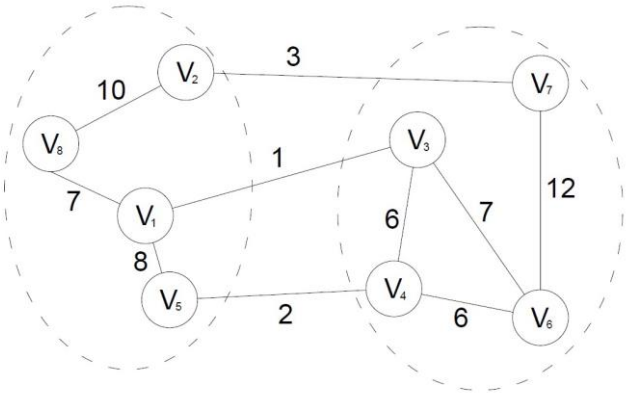
- Velikost generace = 6
- Počet iteraci = 20
- Pravděpodobnost křížení = 0.8
- Pravděpodobnost mutace = 0.4

Vypsal následující informace:

```
Zadejte matici sousednosti: matice.txt
Nejlepsi jedinec 0. generace je: 01110100
0. generace:     28
1. generace:     23
2. generace:     23
3. generace:     20
4. generace:     20
5. generace:     20
6. generace:     20
7. generace:     20
8. generace:     6
9. generace:     6
10. generace:    6
11. generace:    6
12. generace:    6
13. generace:    6
14. generace:    6
15. generace:    6
16. generace:    6
17. generace:    6
18. generace:    6
19. generace:    6
20. generace:    6
Nejlepsi jedinec je: 00110110
K prvni podmnozine patri vrcholy: 1 2 5 8
K druhe podmnozine patri vrcholy: 3 4 6 7
```

Obr. 1.7: Výpis programu

Ve výpisu vidíme, že i když generace nebyla vůbec rozsáhlá, již v 8. generaci bylo nalezeno nejlepší řešení. Pro přehlednost uvádím výsledek v podobě grafu.



Obr. 1.8: Výsledek v grafické podobě

1.2 Gramatická evoluce

GE je jedna z nejmodernějších technik EVT kombinující v sobě některé rysy GA a GP (ve výkladu je rozebrané později). Hlavní odlišnosti od GA je výsledek poskytovaný GE – vygenerovaný program nebo výraz.

Pro popis mechanismu GE je třeba zavést pojem jazyka a gramatiky. Jazyk *L* nad abecedou *T* je libovolná podmnožina množiny všech řetězců nad abecedou *T*, kde abeceda *T* je konečná množina symbolů. Příkladem může být abeceda *T = {a, :, 1}* a jazyk *L = {a:1, 1a, a:, 1:}*. Jazyky využívané GE jsou popsané Backus-Naurovou formou. BNF je metajazyk sloužící ke generování jazyků bezkontextovou gramatikou. Gramatika je definovaná jako množina *G = {N, T, P, S}*, kde

N – množina neterminálních symbolů
T – množina terminálních symbolů
P – množina pravidel
S – startovací symbol $S \in N$

1.2.1 Backus-Naurova forma

Pro názorné zobrazení je uveden příklad reprezentace gramatiky s použitou BNF. Účelem je generování rovnice obsahující pouze konstanty a operace sčítání, odčítání.

N = {výraz, operace}
T = {+, -, konst}
S = výraz

Jedna z možných variant množiny pravidel P může vypadat takto:

(A) < výraz >::= < výraz ><operace> (0)

| konst<operace> <operace> (1)

| konst<operace> (2)

(B) <operace>::= +konst (0)

| -konst (1)

Výše popsaná gramatika je vlastně dekodérem pro chromozom. V GE se používá binární reprezentace chromozomu, který je ještě dále rozdělen do podřetězců tzv. kodonů, kódující celá čísla. Algoritmus rozhoduje, jaké pravidlo bude použito právě na základě přečteného kodonu, tento proces se jmenuje rozvinutí neterminálu. Obecně hodnota kodonu může přesahovat počet pravidel příslušného neterminálu, proto stanovení pravidla používá se vztah:

pravidlo = hodnota_kodonu MOD počet_pravidel_daného_neterminálu

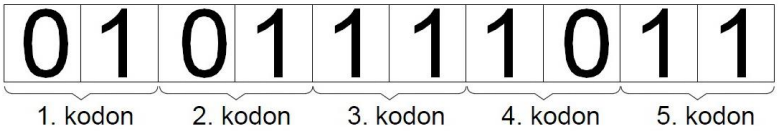
Při dekódování chromozomu čtení probíhá zleva doprava po jednotlivých kodonech. Za normálního běhu programu mohou nastat tři případy:

1. Je přečten celý chromozom a všechny neterminály byly přepsané na terminály. Dekódování proběhlo úspěšně, čte se následující chromozom.

2. Nezbyly žádné neterminály, ale byla přečtena pouze část chromozomu. V tomto případě zbytek chromozomu bude ignorován a program pokračuje dál.

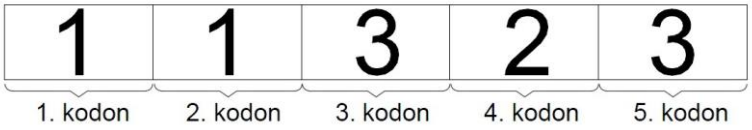
3. Byly přečteny všechny kodony chromozomu, ale zbyly nerozvinuté neterminály. Většinou řeší se to tak, že pro přepsání zbylých neterminálů budou kodony opětovně přečteny od začátku, a aby nedošlo k zacyklení, počet průchodu chromozomem je přesně stanoven.

Proces dekódování chromozomu měl by být jasný následujícího popisu.



Obr. 1.9: Chromozom rozdělený na kodony

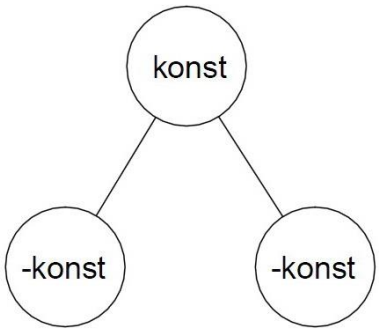
Po převodu kodonů z dvojkové do desítkové soustavy dostáváme



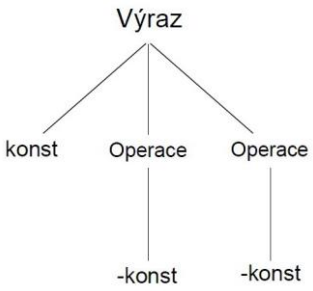
Obr. 1.10: Chromozom v desítkové soustavě

Dekódování vždy začíná rozvinutím startovacího symbolu, v našem případě je to <*výraz*>. Pravidlo, podle kterého neterminál přepíšeme, se určí prvním kodonem a výše uvedeným vztahem: *1 MOD 3 = 1*, tím pádem dostáváme rovnici *konst*<*operace*><*operace*>. Stejný princip použijeme pro rozvinutí posledních dvou neterminálů, v obou případech bude použito první pravidlo, zbytek chromozomu je ignorován. Výsledná rovnice vypadá takto: *konst - konst - konst*.

Pro přehlednější zobrazení výsledků existuje také grafická podoba řešení představená stromy dvou typů buď ve formě klasického stromu nebo stromu derivačního, kde je zobrazené postupné přepisování neterminálů.



Obr. 1.11: Dekódovaný chromozom ve formě

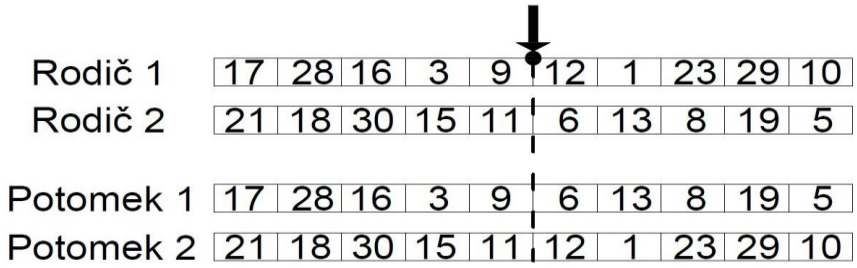


Obr. 1.12: Dekódovaný chromozom ve formě derivačního stromu

Na jednoduchém příkladu byl ukázán princip interpretace výsledků z binární podoby do skutečné rovnice.

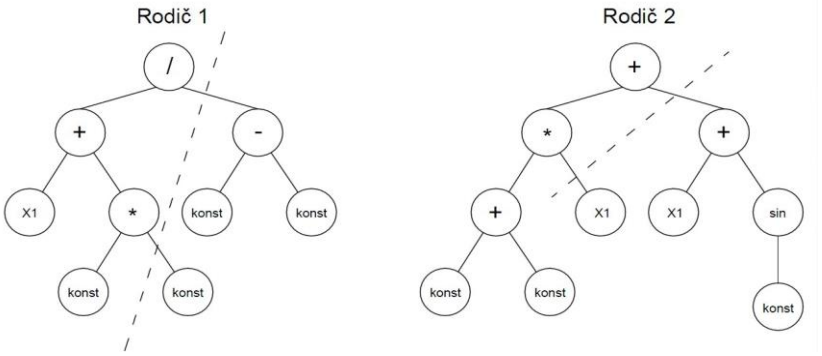
1.2.2 Rekombinační operátory v GE

Základní principy rekombinačních operátorů jsou stejné, jako u SGA, však implementace v GE je poněkud jiná. Stálé se jedna o jednobodové křížení, ale v GE se bod křížení vybírá na úrovní kodonů a ne genů, jak je to v SGA. Danou skutečnost si ukážeme na příkladu. Bohužel z důvodu triviality předchozí úlohy se nedá použít výše definovaná gramatika, ale princip procesu by měl být zřejmý.

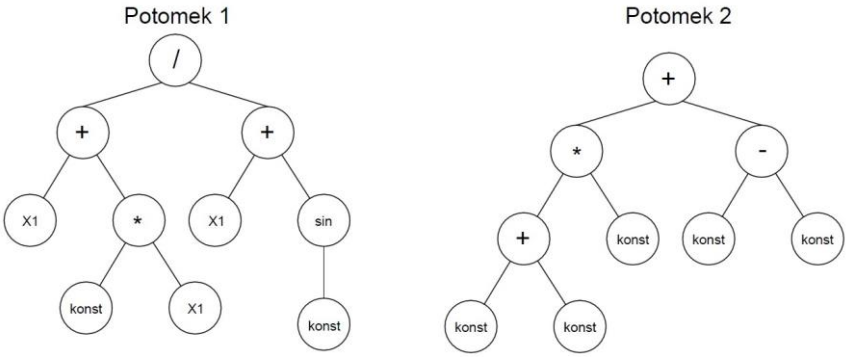


Obr. 1.13: Křížení jedinců převedených do desítkové soustavy v GE

Dále následuje ukázka křížení stejných jedinců ve stromové podobě. Tvoření stromů je podrobněji rozebrané v kapitole zabývající GP.

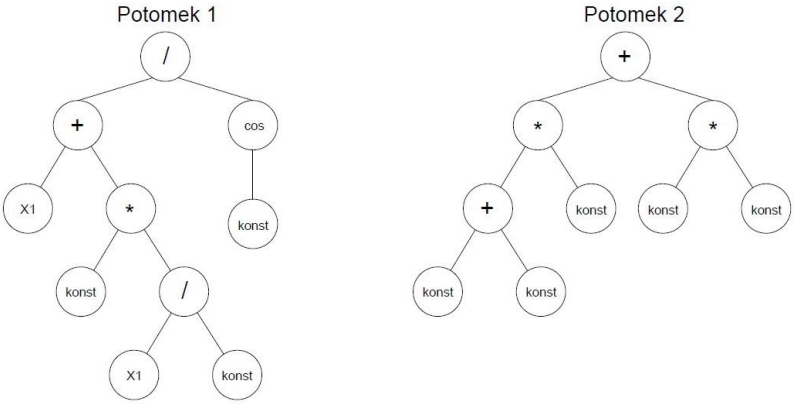


Obr. 1.14(a): Křížení stromů - předkové



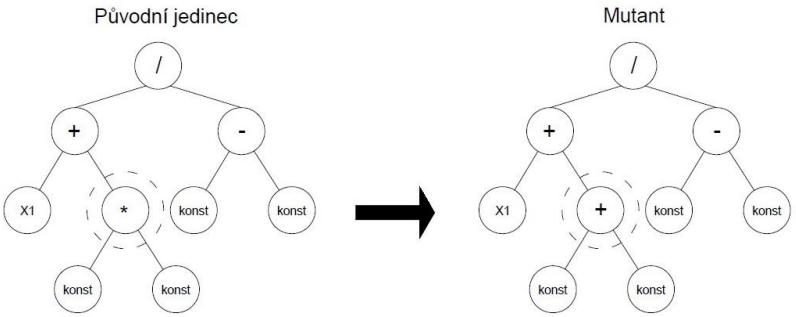
Obr. 1.14(b): Křížení stromů - potomci

Však výše uvedené schéma je pouze jeden z možných případů dekódování potomků, který se podobá křížení v GP, na následujícím obrázku si ukážeme další možný případ.



Obr. 1.15: Křížení stromů potomci další možný případ

Tolik ke křížení. Dále si popíšeme další základní rekombinační operátor – mutace. Tento operátor ve své podstatě je stejný jako v SGA, proces mutace probíhá nad chromozomy vyjádřené ve dvojkově soustavě, což ovlivňuje převod do desítkové soustavy a následovně i dekódování. Jeden z možných případů mutace je na následujícím obrázku.



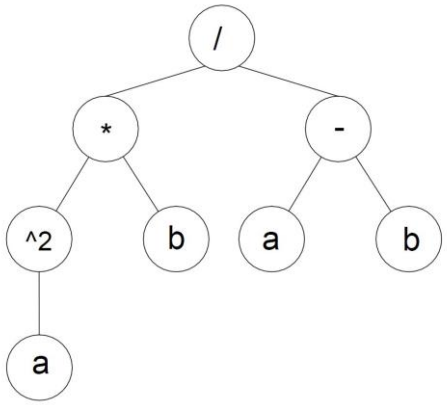
Obr. 1.16: Mutace v GE

1.3 Genetické programování

Genetické programování je moderní EVT napodobující GA, které byly základem pro vytvoření této techniky. Podstatným rozdílem mezi GA a GP je oblast použití. GA jsou určené k řešení optimalizačních úloh vyhledáváním proměnných v rozsáhlém systému, a GP se zabývá buď generováním programů (společný rys s GE), nebo rozhodováním za určitých podmínek. Reprezentace jedinců je také výraznou odlišností těchto technik. Na rozdíl od GA a GE, ve kterých chromozomy mají podobu lineárních řetězců, GP nejčastěji pracuje se stromovými strukturami.

1.3.1 Reprezentace jedinců

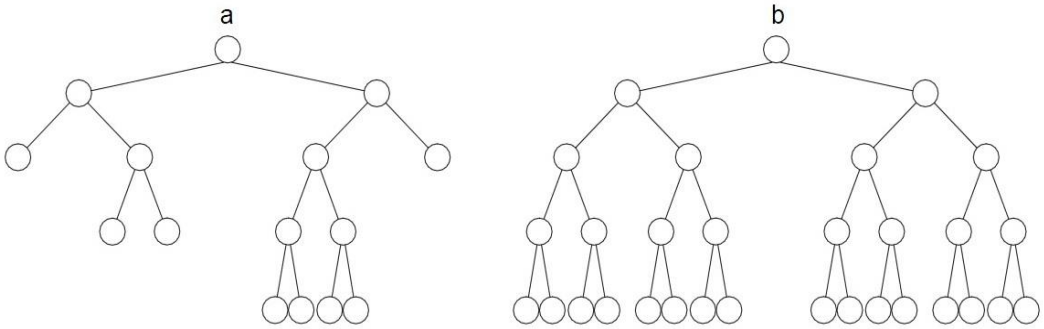
Jak již bylo zmíněno, chromozomy v GP jsou tvořené stromy. Definice stromu zní: strom je souvislý graf, který neobsahuje jako podgraf kružnici. Uzly stromu jsou ohodnoceny terminálními a neterminálními symboly. Terminální symboly se vyskytují pouze listových vrcholech, neterminální symboly se mohou objevit pouze v nelistových uzlech. Vzhledem k tomu, že neterminální symboly představují funkce s různým počtem argumentů, opět se zavádí pojem arity. Ukážeme si příklad: nechť máme množinu neterminálních symbolů F = {/, *, -, ^2}, kde všechny funkce kromě poslední mají aritu dva, a funkce druhé mocniny má aritu jedna. Terminální symboly T = {a, b}, jsou použity pro ilustraci tvoření stromu.



Obr. 1.17: Výraz $(a^2 * b)/(a - b)$ *ve stromové podobě*

Daný způsob reprezentace umožňuje implementovat proces vygenerování počáteční populace řízeným. Vhodný zásah do generování zvýší efektivitu evoluce a zkrátí výpočetní čas. Jestliže se pokusíme náhodně vygenerovat rozsáhlou populaci jedinců, pravděpodobnostní předpoklad je, že většina z nich má buď příliš velkou hloubku[3] nebo je triviální[4]. Právě proto se hloubka generovaných stromů omezuje definováním intervalu možných hodnot. Obecně existují dvě strategií generování: růstová a úplné generování stromu. Růstové generování zohledňuje pouze maximální hloubku, a tím pádem vytváří stromy, jejichž větve mají různou hloubku. Při úplném generování musí být hloubka všech větví stejná.

[3] Počet vrcholů nejdelší cesty mezi kořenem a listem grafu.
[4] Skládá se z pouze jednoho terminálu



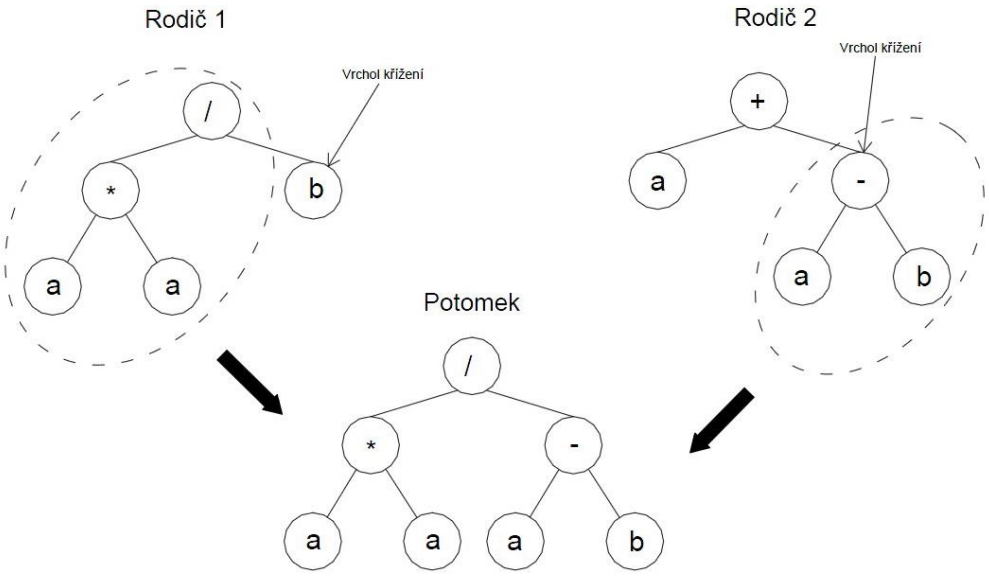
Obr. 1. 18: Strategie generování (a) růstová, (b) úplná

Stromy zobrazené na obrázku *1.14* mají hloubku čtyři. Implementace těchto strategií se liší již po vygenerování kořene stromu. Za úplné strategie vybíráme z množiny neterminálů, pokud neklesneme na hloubku *MAX – 1*, zbytek se naplňuje z množiny terminálů. Generace růstová vybírá náhodně z obou množin během celého procesu generování jedince.

Nejenom určením minimální a maximální hloubky můžeme zkvalitnit počáteční populaci, ale i přednastavením procentuálního počtu jedinců určité hloubky. Při aplikaci GP na tvorbu výrazu většinou nejsme schopni predikovat délku, respektive hloubku řešení. Proto je z logického hlediska výhodnější mít stromy všech hloubek ve stejném počtu. V takovém případě, při definované maximální hloubce pět, každá hloubka (kromě hloubky jedna) by měla *25%* procentní zastoupení v inicializační populaci. Pro ještě větší rozmanitost populace a zároveň větší pravděpodobnost úspěchu, jedna polovina každého z těchto segmentů je vygenerovaná růstovým generováním a druhá úplným.

1.3.2 Rekombinační operátory v GP

Procesy rekombinace GP a GE jsou velmi podobné na první pohled, avšak existují mezi nimi principiální rozdíly, které si dále ukážeme. Nejprve se náhodně vybere vrchol křížení v prvním rodiči potom podle toho, byl-li vybrán terminál respektive neterminál, vybírá se vrchol stejného typu ve druhém rodiči. Přičemž arita neterminálů obou rodičů musí být stejná. Křížením dvou rodičů vznikne jeden potomek, a to tak, že z prvního rodiče budou odstraněny všechny vrcholy směřující od vybraného směrem k listům. Na toto místo budou zkopírované vrcholy druhého rodiče od zvoleného směrem k listům. Následující obrázek zachycuje klasickou variantu křížení v GP.



Obr. 1.19: Křížení v GP

Proces mutace probíhá následovně: náhodně se vybere vrchol stromu a vymění se na náhodně zvolený ze seznamu všech možných, přičemž terminály se mění na terminály a neterminály na neterminály. Grafický postup zde neuvádím, protože je stejný jako v GE.

2. Numerické metody řešení diferenciálních rovnic

Diferenciální rovnice jsou matematickým nástrojem pro popis veškerých procesů nebo celých systémů. Ruční zpracování těchto rovnic je celkem náročná úloha, a při práci s komplexními procesy se stává téměř nemožnou. Na pomoc přicházejí počítače, na kterých probíhají všechny výpočty. Vzhledem ke specifiku strojových výpočtů se nepoužívají analytické metody, ale numerické metody. Numerický způsob řešení se liší tím, že pouze aproximuje řešení funkce v bodech po určitém časovém kroku, tj. dochází k diskretizaci funkce. Při numerickém řešení obyčejné diferenciální rovnice výpočet začíná v bodě x_0, kde musí být zadaná počáteční podmínka, která je faktický funkční hodnotou v tomto bodě. Dále začíná samotné hledání řešení v ostatních bodech. V závislosti na tom kolik bodů se používá pro výpočet následující hodnoty, dělíme numerické metody na jednokrokové a vícekrokové.

2.1 Eulerova metoda

Byla první jednokrokovou metodou použitou pro řešení diferenciálních rovnic, která vychází z definice derivace:

$$\dot{y}(x_n) = \lim_{\Delta x \to 0} \frac{y(x_{n+1}) - y(x_n)}{\Delta x}$$

Při numerickém řešení je tento vzorec upraven na:

$$\dot{y}(x_n) = \frac{y(x_{n+1}) - y(x_n)}{h} = f(x_n, y_n)$$

Kde *h* je vzdálenost mezi body x_n a x_{n+1}, tzv. krok. Po vyjádření $y(x_{n+1})$ dostaneme vzorec, který vlastně je Eulerovou metodou:

$$y(x_{n+1}) = y(x_n) + h \times f(x_n, y_n)$$

Vzhledem k jednoduchosti této metody je její předností rychlost výpočtu, ale metoda je schopna najít pouze přibližné řešení. Nízká přesnost je způsobena tím, že při průchodu z bodu x_n do x_{n+1} je hodnota derivace konstantní, která se ve skutečnosti mění. Zlepšení hledaného řešení lze dosáhnout modifikací metody nebo použíti přesnější, vícekrokové metody.

2.2 Metoda Runge-Kutta

Dává poněkud kvalitnější výsledky, než Eulerova metoda, za cenu větší výpočetní náročnosti. Principem je použíti několika pomocných bodů mezi sousedními uzly. Rozlišujeme více metod Runge-Kutta, které se liší počtem bodů využitém pro výpočet jednoho kroku. Obecně lze s-stupňové metody zapsat ve tvaru:

$$y(x_{n+1}) = y(x_n) + h \times (b_1 \times y_1 + b_2 \times y_2 + \ldots + b_s \times y_s),$$

kde koeficienty k_i, $i = 1, 2, \ldots, s$, jsou určeny předpisem

$$k_1 = f(x_n, y_n),$$

$$k_2 = f(x_n + h \times c_2, y_n + h \times a_{21} \times k_1),$$

$$k_3 = f\big(x_n + h \times c_3, y_n + h \times (a_{31} \times k_1 + a_{32} \times k_2)\big),$$

$$\vdots$$

$$k_s = f\left(x_n + h \times c_s, y_n + \sum_{i=1}^{s-1} a_{si} k_i\right)$$

Nejpoužívanější z nich je metoda Runge-Kutta 4. řádu, která se vyskytuje v programu, jako nástroj pro výpočet hodnoty fitness funkce. Volá se z externí knihovny GNU Scientific Library[5].

[5] Knihovna pro aplikovanou numerickou matematiku, vytvořená v rámci projektu GNU

3. Cíle práce

3.1 Problematika

Problematika, kterou se zabývá daná práce, je aproximace, resp. regrese, signálů EEG. Konkrétněji, snaha vygenerovat diferenciální rovnici, která popisuje průběh EEG signálu pomoci EVT. Problematika byla již částečně řešena v bakalářské a diplomové práci Ing. Z. Barneta [4]. Metoda popsaná v bakalářské práci vychází z gramatické evoluce. Výsledkem práce byl program, kde vstupem je textový soubor s navzorkovanou křivkou s daným konstantním časovým krokem. Úkolem algoritmu je při zadaném řádu najít diferenciální rovnici právě pomocí GE. Aplikace nebyla testována na reálných signálech EEG, ale na průbězích, které byly vygenerovány jako numerické řešení předem vybraných rovnic; předmětem testu bylo porovnání, zda GE je schopna nalézt diferenciální rovnici shodnou s tou, která byla využita jako „generátor" testovacího průběhu (viz dále). Ukázalo se, že pro jednoduché rovnice je řešení nalezeno během krátké doby. Při práci se složitějšími rovnicemi (rovnice typu Mathieu) se rovnici najít nepodařilo. Dalším rysem byl velký počet po sobě jdoucích generací beze změn objevujících se v okamžiku, kdy algoritmus už pomocí rekombinačních operátorů nedokáže získat lepší řešení. Na základě těchto výsledků bylo posouzeno, že se vhodnější technikou jeví GP. Proto následující práce [5] byla věnovávaná hledání diferenciální rovnici, ale již pomocí GP.

Experimenty provedené během řešení diplomové práce ukázaly celkově lepší konvergenci metody GP. Algoritmus byl opět testován na stejných vygenerovaných průbězích. Potvrdila se snadnost nalezení jednoduchých řešení diferenciálních rovnic. Řešení komplikovanější úlohy (typu Mathieu) bylo tentokrát skutečně v jednom běhu nalezeno, ale byla nezbytná rozsáhlá populace a velký počet iterací. Právě počet iteraci nejčastěji přímo určuje kvalitu výsledku, samozřejmě podmínkou je dostatečně velká populace. Po delší době běhu algoritmu se výsledek čím dál víc přibližuje správnému řešení.

Rovnice použité pro experimenty lze rozdělit do třech úrovní složitosti. Analytický zápis první sady:

$$y''(t) = 0 \times y(t)$$

$$y''(t) = 2 \times y(t)$$

$$y''(t) = t \times y(t)$$

Obě metody zvládly najit správné řešení i s malým počtem generaci. Však při použíti GE řešení bylo nalezeno již v první generaci, a v GP jedinci se postupně rozvíjeli, pokud

algoritmus neukončil úspěšně svou činnost. Dále byl pokus o generování diferenciálních rovnic s trigonometrickými koeficienty, jejich analytický zápis:

$$y''(t) = \sin(t) \times y(t)$$

$$y''(t) = \cos(t) \times y(t)$$

Stejně jako v předchozím případě se oběma metodami podařilo najít správné řešení. Pouze při experimentech s GP bylo rozhodnuto nastavit menší populaci a zvětšit počet iteraci, což podle očekávání vedlo ke snížení efektivity algoritmu, a správný výsledek byl nalezen pouze v malém procentu experimentů. Testování pokračovalo na komplikovanější rovnici typu Mathieu, z toho důvodu, že svým průběhem napodobuje křivku EEG. Její analytický zápis:

$$y''(t) = -[16 + 16 \times \cos(2t)] \times y(t)$$

Danou rovnici pomoci GE v nejlepším experimentu podařilo se pouze aproximovat, a to jenom v prvních dvou periodách, další průběh nalezené rovnici je velice odlišný od hledaného. Potvrdil se předpoklad perspektivy GP, při pomoci které jednou bylo nalezeno přesné řešení. Jedna se samozřejmě o náhodu, ale i ostatní řešení nalezené v jiných experimentech dost podobaly správnému řešení. Podmínkou, ale zase je dostatečně velká populace zvětšení počet iteraci.

3.2 Modifikace algoritmu GP

Výsledkem diplomové práce je algoritmus umožňující práce se složitějšími průběhy, ale zatím se nehodí pro aproximaci signálů EEG. Důvodem jsou některá omezení.

1) V současné podobě je program implementující metodu GP schopen generovat pouze lineární diferenciální rovnici vyššího (ale předem stanoveného) řádu ve tvaru:

   $y^{(n)} + c_{n-1}(t)y^{(n-1)} + \ldots + c_1(t)y' + c_0(t)y = f(t)$, kde

   n – je řád lineární diferenciální rovnice

   koeficienty $c_0(t), \ldots, c_{n-1}(t)$ – jsou funkce obecně závislé pouze na t

   funkce $f(t)$ – pravá strana, přičemž může to být i konstantní nulová funkce

Jedním z rozšíření algoritmu je přidání možnosti generování funkcí c_i ve tvaru $c_i\big(t, y(i), y'(i), \ldots, y^{(n)}(i)\big)$, zvětšující rozmanitost poskytovaných řešení, předpokládaným důsledkem je přiblížení ke správnému řešení.

2) Hlavními nástroji vývoje jedinců ve všech druzích EVT jsou rekombinační operátory, jelikož přinášejí změny do stávajících jedinců. Ne vždy jsou to změny zvyšující kvalitu jedinců. Jedním z tak zvaných vedlejších efektů křížení je nadbytečná hloubka

potomku, což v kontextu řešené úlohy znamená, zbytečně rozsáhlé řešení. Návrhem je omezení hloubky křížených jedinců. Poznámka: při generování vzorových posloupností jsou data ukládána do textového souboru a dochází přirozeně k zaokrouhlovacím chybám při převodu z interní binární reprezentace typu double do textové podoby. To mělo za následek nenulovou hodnotu fitness funkce (suma kvadrátů odchylek) i v případě, že byla nalezena rovnice shodná s „generátorem“. Chyba se eliminuje možností binárního formátu vstupních dat. Předpokladem je, že po uskutečnění výše uvedených úprav aproximace poskytovaná algoritmem se víc přiblíží ke správnému řešení. Má tedy smysl vyzkoušet funkčnost GP na reálném signálu EEG.

Cíle této práce jsou tedy následující:

- rozšíření stávajícího algoritmu GP o možnost generování funkcí c_i ve tvaru $c_i\left(t, y(i), y'(i), \dots, y^{(n)}(i)\right)$
- omezení hloubky generovaných potomků při křížení
- aplikace metody na vzorové signály EEG
- možnost čtení vstupních dat v binární podobě

3.3 Řešení

3.3.1 Rozšíření třídy GPTree

První aktualizací stávajícího algoritmu bylo rozšíření možností generování jedince implementované úpravou definované v diplomové práci [5] třídy GPTree. Klíčovou myšlenkou rozšíření třídy je větší počet nabízených řešení a jako důsledek větší výkon celé metody. Dřívější verze algoritmu generovala pouze konstanty diferenciální rovnice předem daného řádu, v rámci dané práce byla přidaná možnost generování další derivací funkcí a konstant náhodně. Změny ve třídě GPTree začínají přidáním atributu typu *int*[6] *degree*, který vyjadřuje řád diferenciální rovnice. Následovně byl změněn konstruktor[7], který jako argument má výše popsanou proměnnou. Dále byly upraveny metody *run()* a *evaluate()*, které vyhodnocují strom, respektive podstrom, v čase t s úvahou dalších derivací.

Jedinci stále ukládají v poli *buffer*, byl pouze rozšířen rozsah hodnot $\langle 0; num\ const\rangle \cup \langle 10\,000; 10\,006\rangle \cup \langle 10\,006; 10\,006 + degree\rangle$, kde $degree$ je nejvyšší řád diferenciální rovnice. Kódování v rozsahu *0* až *10 006* zůstává stejné, následující hodnoty

[6] Integer – jednoduchý celočíselný datový typ
[7] Metoda třídy její úkolem je vytvoření a inicializace objektu

potom mají význam: *10 007* je první derivace, *10 008* druhá derivace, *10 009* třetí derivace atd.

3.3.2 Omezení hloubky křížení

V době provádění experimentů nad algoritmem byl objeven podstatný nedostatek algoritmu – nadbytečná hloubka výsledných stromů. Zdroj způsobující překročení maximální povolené hloubky se nacházel na etapě rekombinace jedinců během jejich křížení. Pro odstranění nadbytečného růstu v této fázi byly naprogramovaný další metody umožňující kontrolu hloubky stromu. Doplněné metody třídy GPTree:

- *traverse()*
- *travers_with_depth()*
- *find_depth()*
- *find_vertex()*

Přetížená metoda *traverse()* byla obohacena o jeden výstupní parametr, do kterého (jako vedlejší efekt – side effect – funkce) je ukládána hloubka traverzovaného stromu, respektive podstromu. Přetížená metoda *traverse()* volá rekurzivní metodu *travers_with_depth()*, která na rozdíl od původní metody *traverse()* při procházení jednotlivých vrcholů poznamenává hloubku, ve které se nachází.

Metoda *find_depth()* prostřednictvím volání *find_vertex()* určuje hloubku uzlu, který je argumentem funkce. Princip hledání je shodný s metodou *travers_with_depth().*

Výše popsané metody umožnily implementovat přetíženou metodu *crossover()*, která zabraňuje generování potomků s větší než povolenou hloubkou. Křížení v ní probíhá následujícím způsobem: nejdřív se náhodně po celé délce vybere bod (vrchol) křížení v prvním rodiči, stejně se vybere bod ve druhém rodiči a poznamená se jeho hloubka, potom je vypočtená hloubka od kořene do bodu křížení v prvním rodiči, a uložená do proměnné *depth_of_cross*. Dále není-li splněná zastavovací podmínka – součet hloubky křížení druhého rodiče a *depth_of_cross,* musí být menší nebo rovnat se maximální stanovené hloubce, snižuje se hloubka křížení ve druhém rodiči. Pokud zastavovací pravidlo vyhodnocené jako pravdivé probíhá sestavení jedince, identické původní verzi křížení.

3.3.3 Binární čtení dat

Načítání dat v algoritmu navrženém v rámci diplomové práce probíhalo z textového souboru. Textový formát souboru se používal z důvodu přehlednosti poskytovaných dat. Ale při pokusech s testovacími úlohami byla návratová hodnota fitness funkce, která měla být teoreticky nulová (nalezená rovnice byla shodná s tou, pomocí které byla generována

vzorová data), byla ve skutečnosti nenulová a to z jednoduchého důvodu – zaokrouhlovací chybě. Řešením je ukládání dat v binární podobě a implementace funkce umožňující čtení přímo z binárního souboru. Tímto krokem se vyhneme zaokrouhlovacím chybám při generování textového souboru z vnitřní reprezentace typu doubl .

Původně prvním krokem algoritmu bylo načítání dat z textového souboru, v současnosti je toto čtení vstupních dat implementováno v samostatné funkci *read_text()*. Výchozím je stále čtení dat textového souboru, ale v případě spouštění programu s přepínačem „-b“ na konci se volá nově implementovaná funkce *read_binary()* a čtení probíhá z binárního souboru.

3.3.4 Práce s daty EEG

Výstupní textový soubor elektroencefalografu obsahuje dvacet jedna kanálů dat EEG. Každý kanál nese v sobě informaci z konkrétní elektrody, nacházející se na určité části lebky. Soubor se skládá ze třiceti třech sloupců a počet řádků je závislý na tom, jak dlouho bylo prováděno měření. Dvacet jedna prvních sloupců jsou právě kanály EEG, zbytek je služební informace. Pro větší přehlednost jsou v prvním řádku názvy jednotlivých kanálů, tak například řádek začínající názvem Fp1 znamená, že všechna data v prvním sloupci popisují kanál Fp1. Další řádky obsahují naměřené hodnoty signálu EEG v mikrovoltech na jednotlivých kanálech, přičemž frekvence vzorkování byla 256 vzorků za sekundu.

Zřejmé je, že algoritmus GP není schopen pracovat s takovým vstupním souborem. Pro vytvoření vhodného souboru byl napsán pomocný program *Read_data_EEG.exe*. Daný program nejenom načítá vybraný kanál do nového souboru, ale i přidává chybějící časovou složku. Tím pádem *Read_data_EEG.exe* vytváří soubor se dvěma sloupci, kde první je čas a druhy odpovídající hodnota kanálu v mikrovoltech. Spouštění programu probíhá z příkazového řádku, například:

Read_data_EEG.exe R1.180002.txt Fp2 , kde

R1.180002.txt – soubor EEG

Fp2 – vybraný kanál

Výsledkem programu v daném případě bude textový soubor Fp2.txt vhodný pro čtení algoritmem GP.

4. Experimenty

Po provedených úpravách má smysl vyzkoušet efektivitu algoritmu na diferenciálních rovnicích různé složitosti. Daná práce se zabývá stejnou problematikou, jako i diplomová práce [5], proto je vhodné porovnat výsledky obou algoritmu při práci s náročnějšími rovnicemi. Vyzkoušet vliv velikosti generace, počtu iterací, pravděpodobnosti použití rekombinačních operátorů a jiných genetických parametrů na výsledné řešení. Porovnání vstupního a generovaného signálu probíhá pomocí metody nejmenších čtverců (hlavní kritérium pro fitness funkci) a korelace. Během experimentů v rámci diplomové práce se však ukázalo, že korelace není vhodná v daném případě k porovnání signálů, proto budou brány zřetel pouze výsledky metody nejmenších čtverců.

Složitost testovaných rovnic se postupně zvyšovala. Nejdřív byly použity nejjednodušší rovnice shodné s diplomovou prací, jedná se o funkce: $y''(t) = 0 \times y(t)$, $y''(t) = 2 \times y(t)$, $y''(t) = t \times y(t)$. Dále pro vyzkoušení práce s trigonometrickými funkcemi byla použita rovnice $y''(t) = \sin(t) - 10 \times y(t)$. Pro zhodnocení činnosti modifikované verze algoritmu a porovnání s jeho původní verzí byla použita rovnice Mathieu, obecný analytický zápis, které lze zapsat: $y''(t) = -(a + 16 \cdot q \cdot \cos(2 \cdot t)) \times y(t)$, kde *a, q* náleží prostoru reálných čísel. Pro možnost porovnání výsledků konstanty použité při testování jsou stejné, jako v diplomové práci, analytický zápis je tedy: $y''(t) = -(16 + 16 \cdot \cos(2 \cdot t)) \times y(t)$.

Hlavním cílem je vyzkoušet algoritmus na reálných datech EEG. Výstupem elektroencefalografu je soubor obsahující všechny kanály[8] z různých částí lidského skalpu. Obtížnost generování diferenciální rovnice pro signál EEG je poněkud vyšší než u předcházejících funkci, proto lze očekávat pouze aproximaci signálu.

Spouštění samotného programu *diff_rce_gp.exe* probíhá s následujícími argumenty: *diff_rce_gp.exe vst_soubor rad hloubka prava_str Pc Pm N I metoda log_soubor nacteni,* kde:

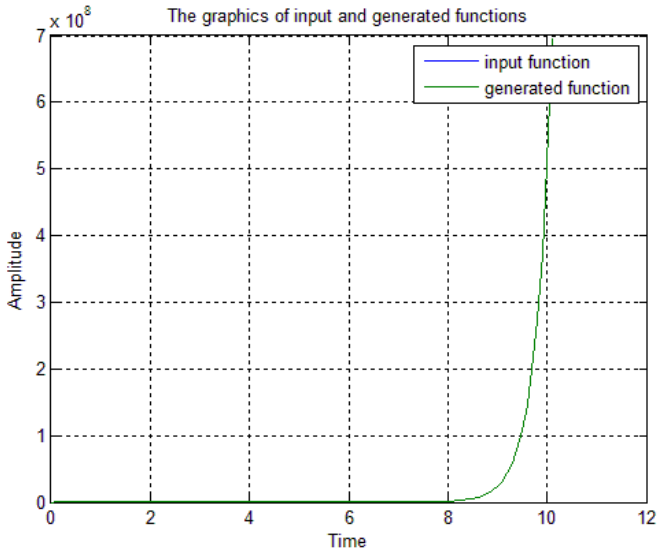
- vst_soubor – název vstupního souboru
- rad – řád generované diferenciální rovnice
- hloubka – maximální hloubka stromu
- prava_str – povolení generování pravé strany může nabývat pouze hodnot *ano* a *ne*

[8] Data z jednoho snímače

- Pc – pravděpodobnost křížení může nabývat hodnot *0* až *1*
- Pm – pravděpodobnost mutace může nabývat hodnot *0* až *1*
- N – velikost generace
- I – počet iterací
- metoda – výběr metody pro vypočet hodnoty fitness může nabývat hodnot *CTVERCE, KORELACE* a *VAZSOUCET*
- log_soubor – nepovinný přepínač *-l* zapíná zápis do logovacího souboru, za tím musí následovat jeho název
- nacteni – nepovinný přepínač -b, v případě jeho přítomnosti načítá se vstupní soubor v binárním modu, jinak v textovém

4.1 Testování na jednoduchých rovnicích

Pro ověření funkčnosti algoritmu byly použity následující rovnice: $y''(t) = 0 \times y(t)$, $y''(t) = 2 \times y(t)$, $y''(t) = t \times y(t)$. Změny neovlivnily schopnosti algoritmu a řešení bylo nalezeno téměř vždycky už v první generaci. Objevují se problémy při příliš malé populaci, což není problém algoritmu, ale evolučních technik obecně. Dalším charakteristickým rysem je generování komplikovanějších rovnic, jejichž snadnou úpravou přijdeme na správné řešení, například: *0.000000*y_1(t)+t*y_0(t).* Graf vstupní a generované rovnice v drtivé většině případů vypadá takto:



Obr. 4.1: Graf zobrazující zpracování rovnice $y''(t) = t \times y(t)$

Zda se, že na obrázku pouze generovaná funkce, kvůli tomu, že je buď stejná jako vstupní nebo rozdíl je příliš malý, aby byl vidět v daném měřítku.

4.2 Testování s využitím trigonometrických funkcí

Dále byly testovány schopnosti algoritmu při práci s trigonometrickými funkcemi, konkrétně byla použitá rovnice $y''(t) = \sin(t) - 10 \times y(t)$. Na rozdíl od předchozí úlohy nebyla nalezená rovnice shodná se zadanou, ale ve většině řešeni se jí velmi blíží. Testování probíhalo s následujícími explicitními parametry:

Řad rovnice: *2*

Hloubka stromu: *4*

Generovat pravou stranu: *ne*

Pravděpodobnost křížení: *Pc = 0.8*

Pravděpodobnost mutace: *Pm = 0.1 - 0.5*

Velikost generace: *N = 100 až 10 000 jedinců*

Počet iteraci: *I = 10 až 100*

A implicitními parametry:

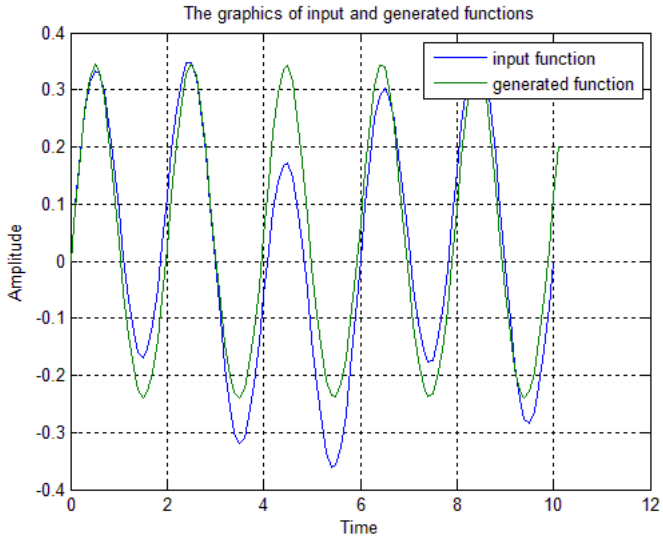
GPTree::num_const = 100;

GPTree::gener_const(0,5);

GPTree::P_liter = 0.2 až 0.5;

Během testování dané úlohy se ukázalo, že generace obsahující pouze sto jedinců je příliš malá pro nalezení správného řešení. Výsledky experimentů při velikosti generace 1 000 a 10 000 jsou skoro stejné, tj. pro generování rovnice dané složitosti optimální velikost je 1 000 jedinců. Rozdíl vývoje jedinců při 10 a 100 iterací není podstatný, proto většina testů byla uskutečněná s menším počtem iterací. Při změně pravděpodobnosti mutace a generování literálu[9] závislost nalezená nebyla. Jedno z nejlepších řešení v grafické podobě:

[9] Proměnná nebo konstanta, potom pravděpodobnost generování funkce je 1 – P_liter



Obr. 4.2: Graf zobrazující zpracování rovnice

$$y''(t) = \sin(t) - 10 \times y(t)$$

Na grafu vidíme, že už nesplývají křivky jako v předchozím případě, ale generovaná rovnice se hodně podobá vstupní funkci. Dá se říct, že výsledek má správnou periodu, ale v některých místech neodpovídá amplituda.

4.3 Testování na rovnici Mathieu

Pro přiblížení k reálným průběhům EEG byl program otestovaný na funkci typu Mathieu, analytický zápis: $y''(t) = -(16 + 16 \cdot \cos(2 \cdot t)) \times y(t)$. Stejně jako u předchozí úlohy neočekává se, že bude nalezené řešení přesně odpovídající zadané rovnici, ale jeho aproximace. Podle předchozích zkušenosti testování probíhalo s následujícími explicitní parametry:

Řád rovnice:	*2*
Hloubka stromu:	*4*
Generovat pravou stranu:	*ne*
Pravděpodobnost křížení:	*Pc = 0.8*
Pravděpodobnost mutace:	*Pm = 0.1 - 0.8*
Velikost generace:	*N = 1 000 až 10 000 jedinců*
Počet iteraci:	*I = 10 až 100*

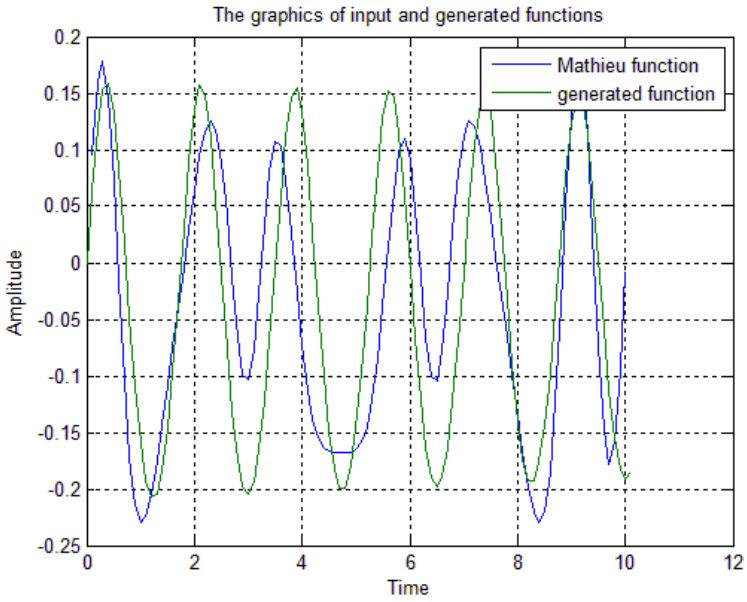
A implicitními parametry:

GPTree::num_const = 100;

GPTree::gener_const(0,5);

GPTree::P_liter = 0.2 až 0.5;

Provedené experimenty ukázaly, že pro řešení podobných úloh vyplatí se nastavení větší pravděpodobnosti mutace (0.8) a velikosti generace (10 000). Naopak nižší pravděpodobnost generování literálu (0.2) se jeví jako lepší. Stejně, jako v předchozím případě, počet iterací 10 je dostatečným pro nalezení lepšího řešení. Grafické zobrazení většiny výsledků vypadá takto:



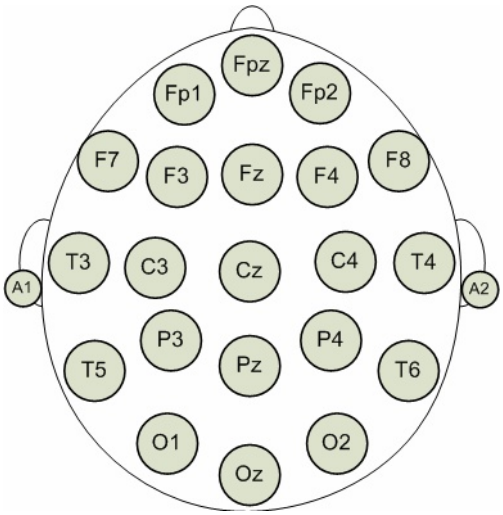
Obr. 4.3: Graf zobrazující zpracování rovnice Mathieu

Na grafu vidíme, že vygenerovaná funkce se velmi podoba funkci Mathieu, součet nejmenších čtverců v nejlepším případě byl pouze 0.8263. Na grafu také pozorujeme stejný nedostatek jako v předchozím případě – výsledná křivka se napodobuje sinusoidě, což trochu neodpovídá správnému průběhu.

4.4 Testování algoritmu s využitím dat EEG

4.4.1 Popis signálu EEG

EEG je průběh mozkové aktivity, který se projevuje rozdílem potenciálu mezi určitými místy na povrchu skalpu. Signál EEG vzniká synchronizací impulsů většího množství neuronů se nacházejících v cortexu. Rozmístění jednotlivých elektrod v nejpoužívanějším systému 10/20 znázorňuje následující obrázek:



Obr. 4.4: Rozmístění elektrod [10]

Rozlišujeme dva druhy měření bipolární a unipolární. V případě bipolárního napěti se měří mezi libovolnými dvěma elektrodami. Unipolární je měření rozdílu elektrického potenciálu mezi referenční a jakoukoli jinou elektrodou. Jako referenční mohou být použité elektrody A1 a A2.

Dále rozlišujeme několik druhů aktivit, které určují frekvenci vlny a ve většině případů amplitudu. V následující tabulce budou popsány charakteristiky jednotlivých vln:

Aktivita	Frekvence, [Hz]	Popis	Amplituda, [mV]
Gama	30 – 120	Největší aktivita mozku, zpracování mimořádných událostí	–
Beta	14 – 30	Vysoká aktivita, zpracování informace z okolního světa	5 – 10
Alfa	7 – 14	Klidový stav, relaxace, objevuje se v bdělém při neaktivním zpracování informace	20 – 200
Théta	4 – 7	Objevuje se při meditaci, je hlavní aktivitou ve spánku	10
Delta	< 4	Nejčastěji se vyskytuje v hlubokém spánku	20 – 200

Tabulka 1: Základní aktivity mozku

4.4.2 Testování

Hlavním cílem nejenom dané práce, ale i celého projektu fakulty dopravní zabývajícího evolučními technikami, je aproximace signálu EEG. Během studia této problematiky bylo zjištěno, že GP se jeví jako vhodnější technika pro její řešení. Tak vznikl a následně byl modifikován algoritmus, umožňující generování diferenciální rovnice zadané křivky. Následnými experimenty se ověří, zda algoritmus v současné podobě zvládne vyřešit úlohu, pro kterou byl vytvořen. Pro experiment byly použity reálné EEG údaje, respektive jejích malá část. Konkrétně se použil deseti vteřinový úsek (v čase 0:5:44 – 0:5:54) mozkové aktivity člověka, během kterého byla zafixovaná maximální delta vepředu, proto byl testován kanál Fp1 (použita data byla separovaná ze souboru R1.180001.txt). Tak krátký úsek byl využit z důvodů výpočetní náročnosti kladené na počítač při práci s takovým množstvím dat (frekvence vzorkování – 256 vzorků za sekundu). Explicitní parametry použité během experimentu:

Řád rovnice:	*2*
Hloubka stromu:	*4*
Generovat pravou stranu:	*ne*
Pravděpodobnost křížení:	*Pc = 0.8*
Pravděpodobnost mutace:	*Pm = 0.8*
Velikost generace:	*N = 1 000 až 10 000 jedinců*
Počet iteraci:	*I = 10*

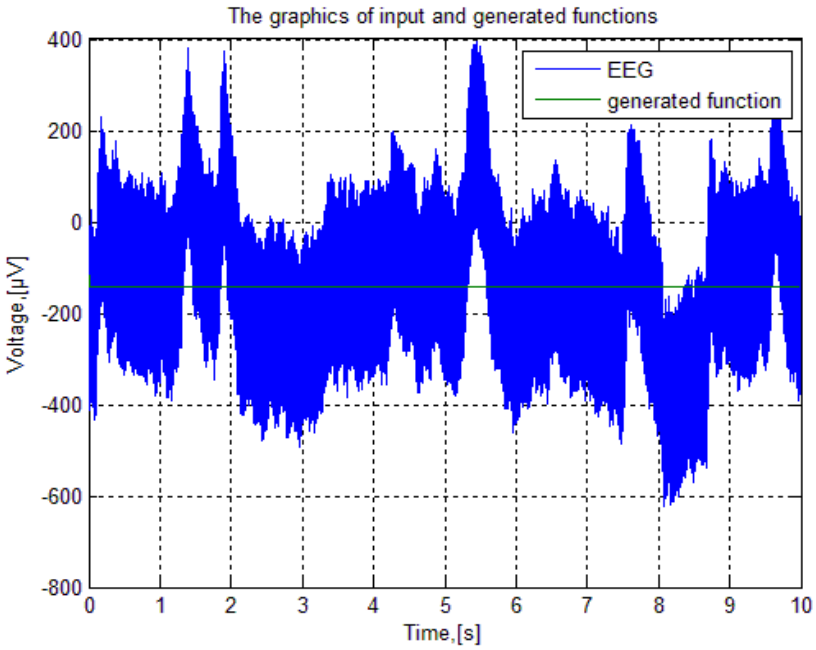
A implicitními parametry:

GPTree::num_const = 100;

GPTree::gener_const(0,5);

GPTree::P_liter = 0.2 až 0.5;

Pro pochopení výsledků dosažených během experimentu, uvádím typické řešení v grafické podobě:



Obr. 4.5: Graf zobrazující zpracování EEG dat

Bohužel nejlepším i jediným řešením byla vodorovná přímka. Veškeré změny parametrů nezlepšili situace, proto musím konstatovat, že na současné etapě není algoritmus schopen pracovat s křivkami EEG.

5. Závěr

Byly splněny všechny cíle práce: seznámení s evolučními technikami, napsání pomocného programu, modifikace existujícího algoritmu GP a jeho testování na reálných EEG datech. Po provedeném testování na rovnici Mathieu, lze usoudit, že změny provedené v již existujícím algoritmu byly úspěšné, protože výsledná rovnice se víc napodobuje správnému průběhu, než v diplomové práci [5]. Lze všimnout pozitivní dynamiku celého projektu, kde v rámci bakalářské [4], diplomové [5] a následovně i této práce se víc a víc přibližujeme správnému řešení rovnice Mathieu, což v budoucnu povede k aproximaci EEG signálu.

Má smysl v budoucnu podrobněji prostudovat problematiku porovnání dvou signálu, protože metoda nejmenších čtverců ne vždy se jeví jako nejlepší. Dále EEG signál má omezený frekvenční rozsah, například alfa aktivita generována v bdělém stavu má frekvenci od 8 do 13 Hz. Tuto vlastnost lze využit při generování signálu, což značně omezí množinu rovnic, které mohou popsat takový průběh. Dalším nápadem je ovlivnění počáteční generace a konkrétně úprava jedinců do podoby rovnice Mathieu. Předpokladem je, že v případě upravené počáteční generace, generování správného řešení bude jednodušší.

Funkčnost algoritmu bych ohodnotil jako velmi dobrou při odhadu jednoduchých diferenciálních rovnic. Táto skutečnost svědčí o tom, že evoluční techniky mohou být použité při řešení podobných úloh, ale zpracování komplexních rovnic vyžaduje zdokonalení současné verze algoritmu.

6. Seznam použité literatury

[1] MAŘÍK, V., O. ŠTĚPÁNKOVÁ, J. LAŽANSKÝ a kol.: *Umělá inteligence (III)*. Praha, Academia, 2001. ISBN 80-200-0472-6.

[2] MAŘÍK, V., O. ŠTĚPÁNKOVÁ a J. LAŽANSKÝ a kol.: *Umělá inteligence (IV)*. Praha, Academia, 2003. ISBN 80-200-1044-0.

[3] POŠÍK, P.: *Genetické algoritmy*. In: [online]. [cit. 2015-04-22]. Dostupné z: http://labe.felk.cvut.cz/~posik/pga/theory/ga-theory.htm

[4] BARNET Z.: *Aproximace signálu s využitím evolučních technik*, bakalářská práce, ČVUT FD, 2013

[5] BARNET Z.: *Aproximace s využitím genetického programování*, diplomová práce, ČVUT FD, 2015

[6] POLI R., LANGDON W. et al.: *A Field Guide to Genetic Programming*.

[7] ZELINKA I. a kol.: *Evoluční výpočetní techniky*, Praha, BEN 2009, ISBN 978-80-7300-218-3

[8] NAVARA, Mirko a Aleš, Němeček. *Numerické metody.* Praha: Vydavatelství ČVUT, 200, ISBN 80-01-02689-2.

[9] DONT, Miroslav. *Numerické metody – cvičení.* Praha: ČVUT, 1990. ISBN 80-01-00400-7

[10] SCHINDLING E.: *Electroencephalography*. In: [online]. [cit. 2015-07-30]. Dostupné z: https://github.com/evsc/eegOSCworkshop/tree/master/presentation

7. Přílohy

V přílohách jsou uvedeny skripty pro převádění diferenciálních rovnic do podoby navzorkované křivky. Opravený hlavní program a algoritmus pro separování kanálu se nachází v komprimovaném souboru, který je k dispozici v elektronické podobě. Přílohy taky obsahují textový soubor s EEG daty na kterých byl algoritmus zkoušen.

7.1 Vzorkování diferenciální rovnice

Skript a funkce napsané v Matlabu generuje dva vektory obsahující čas a funkční hodnotu rovnice. Definování funkce, v daném případě jde o popis rovnice Mathieu.

```
function dy = mathieu(t,y)
dy = zeros(2,1);
dy(1) = y(2);
dy(2) = -(16+16*cos(2*t))*y(1);
```

Zdrojový kód skriptu, který naplňuje vektory času (T) a funkční hodnoty (Y)

```
clear all;
close all;
t = 0:0.1:100; % vektor času
p = 0:1;       % počáteční podmínky
[T,Y] = ode45(@mathieu,t,p);
```

7.2 Zobrazení grafů vstupní a nejlepší generované rovnice

Následující skript napsány v Matlabu načítá data ze dvou souborů (musejí se nacházet ve stejné složce) obsahující původní a nejlepší vygenerovanou funkce. Dále zobrazuje průběhy dvou funkcí na společném grafu.

```
close all;
clear all;

fileID = fopen('mathieu.txt','r');      % načtení dat z
                                          původního souboru
if fileID == -1                         % v případě
                                          neotevření souboru

    error('File is not opened');
end
formatSpec = '%f %f';
Data = textscan(fileID,formatSpec);     % načtení hodnot do
                                          matice Data

fclose(fileID);

Time = Data{1,1};                       % separováni vektoru
                                          casu
Amplitude = Data{1,2};                  % separováni vektoru
```

```
                                                funkčních hodnot
Time(2) = [];                                % smazaní počátečních
                                                podmínek ve vektoru
                                                času
Amplitude(2) = [];                           % smazaní počátečních
                                                podmínek ve vektoru
                                                funkčních hodnot

fileID = fopen('last_gen_curve.txt','r');% načtení nejlepší
                                                vygenerované křivky
if fileID == -1
    error('File is not opened');
end
formatSpec = '%f %f';
Data = textscan(fileID,formatSpec);
fclose(fileID);

X = Data{1,1};                               % vektor času
Y = Data{1,2};                               % vektor funkčních ¨
                                                hodnot

plot(Time,Amplitude,X,Y)                     % zobrazeni obou
                                                grafu
% popis grafu
title('The graphics of input and generated functions');
xlabel('Time');
ylabel('Ampltude');
legend('Mathieu function', 'generated function');
grid
```