

Symbolické integrování a derivování pomocí umělé inteligence

Symbolic integration and differentiation
using artificial intelligence

Bc. Petr Hruboš

Diplomová práce
2011



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petr HRUBOŠ**
Osobní číslo: **A09503**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Počítačové a komunikační systémy**

Téma práce: **Symbolické integrování a derivování pomocí umělé inteligence**

Zásady pro vypracování:

1. Seznamte se s problematikou evoluční syntézy struktur.
2. Seznamte se s nástrojem Analytické programování.
3. Naprogramujte jednotlivé příklady pro symbolické integrování a derivování.
4. Zpracujte pro výukové účely jednotlivé kroky derivování a integrování pomocí evolucí a klasické matematiky.
5. Zpracujte závěr.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ZELINKA, I., OPLATKOVÁ, Z., OŠMERA, P., ŠEDA, M., VČELAŘ, F. Evoluční výpočetní techniky – principy a aplikace. BEN – technická literatura, Praha, 2008, ISBN 80-7300-218-3.
2. ZELINKA, I., OPLATKOVÁ, Z., NOLLE, L., Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms–Comparative Study, International Journal of Simulation Systems, Science and Technology, Volume 6, Number 9, August 2005, pages 44 – 56, ISSN: 1473-8031, online <http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm>, ISSN: 1473-804x.
3. OPLATKOVÁ, Z.: Metaevolution – Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert–Publishing, 2009, ISBN 978-8383-1808-0.
4. KOZA J. R.: Genetic Programming: on the programming of computers by means of natural selection, The Bradford Book, MIT Press, UK, 1992, ISBN 0-262-11170-5.
5. ONEILL M., CONOR R.: Grammatical Evolution, Kluwer Academic Publishers, 2003, ISBN 1-4020-7444-1.
6. BANZHAF W. (ed.): Genetic Programming and Evolvable Machines, Vol. 7, Nr. 1, March, 2006, Springer, ISSN: 1389-2576.

Vedoucí diplomové práce:

Ing. Zuzana Oplatková, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

25. února 2011

Termín odevzdání diplomové práce:

13. června 2011

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Karel Vlček, CSc.
ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá možností využití evoluční syntézy struktur, což je jedna z metod umělé inteligence, v oblasti matematického integrování a derivování. V teoretické části diplomové práce jsou obecně popsány evoluční výpočetní techniky a vybrané evoluční algoritmy. Větší prostor je zde věnován algoritmům DE a SOMA, které jsou využity v praktické části diplomové práce. Dále je v teoretické části popsána evoluční syntéza symbolických struktur a proces symbolické integrace a derivace. V praktické části diplomové práce je použit algoritmus analytické programování jako nástroj symbolické regrese pro nalezení integrálů nebo derivací neznámých křivek definovaných určitou množinou dostatečného počtu bodů. Hlavní součástí práce je vizualizovaný interaktivní program, který bude využit při výuce v oblasti umělé inteligence nebo matematiky.

Klíčová slova: integrace, derivace, umělá inteligence, evoluční algoritmy, SOMA, diferenciální evoluce, analytické programování, optimalizace, evoluční syntéza struktur, mathematica.

ABSTRACT

This thesis deals with the possibility of using evolutionary synthesis structures, which is one of the methods of artificial intelligence, in the field of the mathematical integration and differentiation. In the theoretical part of the thesis, evolutionary computations and selected evolutionary algorithms are generally described. More space is devoted to DE and SOMA algorithms, which are used in the practical part of the thesis. Next, the theoretical description of the evolutionary synthesis of symbolic structures and the process of symbolic integration and differentiation are described. In the practical part of the thesis, analytic programming algorithm is used as a programming tool for symbolic regression to find integrals or derivatives of an unknown curves defined by a certain set of sufficient number of points. The main part of the thesis is a visualized interactive program that will be used for teaching in the field of artificial intelligence or mathematics.

Keywords: integration, differentiation, artificial intelligence, evolutionary algorithms, SOMA, differential evolution, analytical programming, optimization, evolutionary synthesis structures, mathematica.

Na tomto místě bych rád poděkoval Ing. Zuzaně Oplatkové, Ph.D. za cenné rady, připomínky a metodické vedení práce. Poděkování patří také mé rodině za jejich morální podporu a pochopení.

*„Není to ten nejsilnější, kdo přežije,
ani ten nejinteligentnější, ale ten,
kdo se dokáže nejlépe přizpůsobit.“*

(Charles Robert DARWIN)

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 EVOLUČNÍ VÝPOČETNÍ TECHNIKY – EVT	11
1.1 Hlavní myšlenka EVT	11
1.2 Základní postup evolučních algoritmů	12
1.3 Historie	15
1.4 Oblasti použitelnosti evolučních algoritmů.....	16
1.5 Společné rysy	16
1.6 Optimalizační a evoluční techniky	19
1.6.1 Krátký přehled vybraných algoritmů	19
1.6.2 SOMA: SamoOrganizující se Migrační Algoritmus.....	24
1.6.2.1 Parametry a terminologie.....	25
1.6.2.2 Princip SOMA	26
1.6.2.3 Kroky běhu SOMA.....	28
1.6.2.4 Strategie SOMA.....	30
1.6.3 DE: Diferenciální Evoluce	31
1.6.3.1 Parametry a terminologie.....	31
1.6.3.2 Princip a kroky DE	32
1.6.4 Metaevoluce	34
2 EVOLUČNÍ SYNTÉZA SYMBOLICKÝCH STRUKTUR	35
2.1 Symbolická regrese.....	35
2.2 Analytické programování	35
2.2.1 Hlavní myšlenka.....	35
2.2.2 Programová syntéza	37
2.2.3 Posílené hledání	39
2.2.4 Bezpečnostní procedury	39
2.2.5 Podobnosti a rozdíly mezi již existujícími metodami	40
2.2.6 Funkce nelineárního prokládání	40
3 SYMBOLICKÁ INTEGRACE	42
4 SYMBOLICKÁ DERIVACE	45
II PRAKTICKÁ ČÁST	46
5 NÁVRH POSTUPU ŘEŠENÍ	47
5.1 Algoritmy symbolické integrace a derivace	47
5.2 Množina GFS	47
5.3 Testování DE	48
5.3.1 Proces symbolické integrace pomocí DE.....	49
5.3.2 Proces symbolické derivace pomocí DE.....	50
5.4 Testování SOMA	51
5.4.1 Proces symbolické integrace pomocí SOMA	52
5.4.2 Proces symbolické derivace pomocí SOMA.....	53

5.5	INTEGRACE A DERIVACE KLASICKÝMI METODAMI.....	54
5.6	VIZUALIZACE PRO VÝUKOVÉ ÚČELY	55
5.7	OVLÁDÁNÍ APLIKACE	58
ZÁVĚR	59
SEZNAM CITOVANÉ LITERATURY	61
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	63
SEZNAM OBRÁZKŮ	64
SEZNAM TABULEK	65
SEZNAM PŘÍLOH	66

ÚVOD

Umělou inteligenci lze zařadit mezi obory patřící do oblasti informatiky. Umělá inteligence se zabývá tvorbou *strojů*, které vykazují určitý typ inteligentního chování. Definice pojmu *inteligentní chování* je samozřejmě stále předmětem diskuze, nejčastěji se jako etalon inteligentního chování užívá lidské myšlení. Poměrně velké množství optimalizačních či jiných metod, využívajících umělou inteligenci, vychází ze základních principů Darwinovy a Mendelovy teorie evoluce. Stejně tak je tomu i u evolučních výpočetních technik, jejichž hlavní ideou je předání rodičovského genomu novým potomkům a následné uvolnění životního prostoru nové generaci (1).

Algoritmy DE a SOMA, se kterými pracuji, jsou toho příkladem. Pomocí tzv. evoluční syntézy, což je v podstatě proces, obecně nazývaný jako symbolická regrese, a vhodným spojením evolučního algoritmu a nástroje nazvaného analytické programování, lze efektivním způsobem syntetizovat libovolnou funkci v symbolickém, v tomto případě matematickém, tvaru. V této diplomové práci se snažím srozumitelným a jednoduchým způsobem syntetizovat funkce, které jsou integrálem nebo derivací předem neznámé křivky, reprezentované množinou dostatečného počtu bodů ve dvourozměrném prostoru. K tomuto účelu mi poslouží symbolická regrese a nástroj analytické programování spolu s evolučními algoritmy SOMA a DE.

Hlavní cíle praktické části jsou dva. V první řadě je to nastavení a spojení všech tří používaných procesů v jeden celek tak, aby byl výsledný algoritmus schopen syntetizovat funkce integrálů nebo derivací neznámé křivky s uspokojivou, pokud možno co nejmenší, chybou. V druhé řadě se zabývám tvorbou vizualizovaného interaktivního prostředí, které bude v budoucnu využíváno při výuce umělé inteligence nebo matematiky. Veškeré zdrojové kódy jsou přímo závislé na softwaru *Mathematica* a bez jeho přítomnosti není možné jejich použití.

I. TEORETICKÁ ČÁST

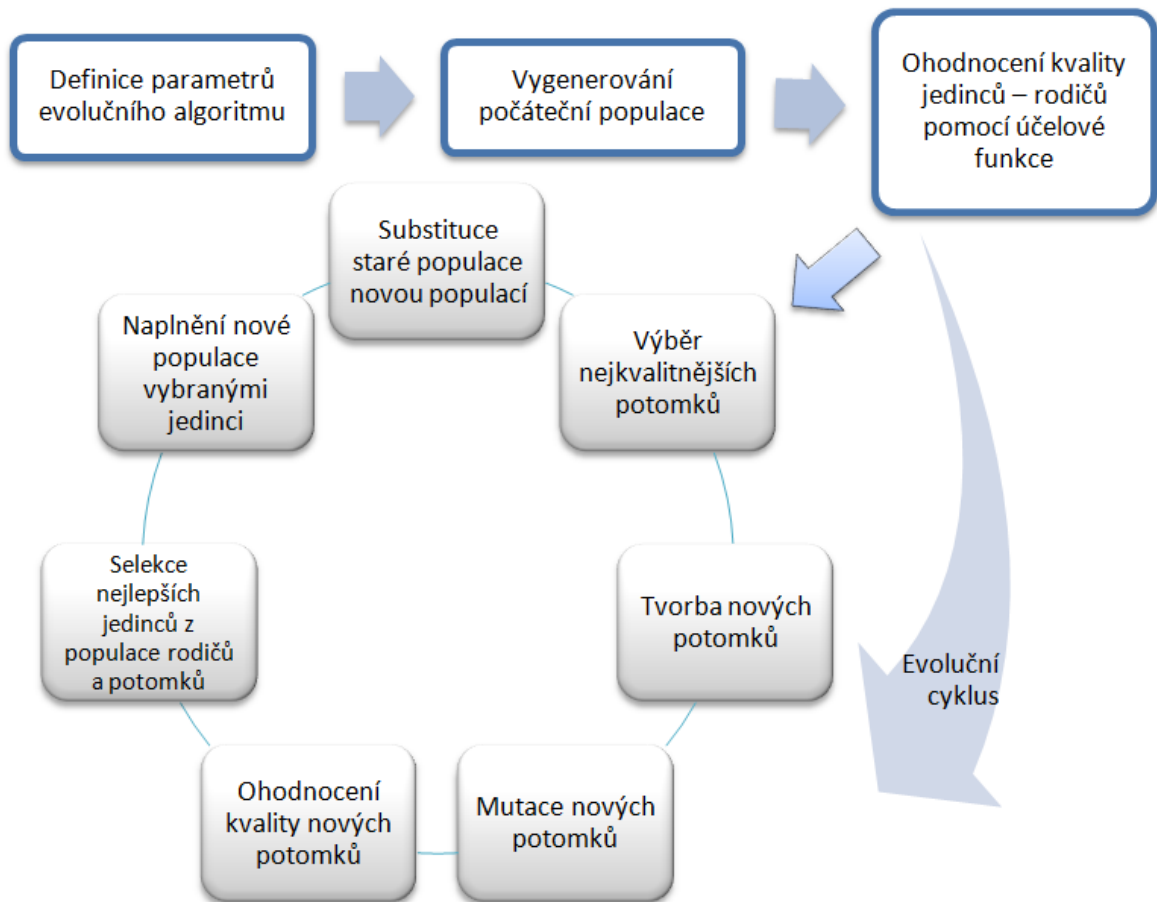
1 EVOLUČNÍ VÝPOČETNÍ TECHNIKY – EVT

1.1 Hlavní myšlenka EVT

Evoluční výpočetní techniky, dále jen *EVT*, jsou numerické algoritmy, které vycházejí ze základních principů Darwinovy a Mendelovy teorie evoluce, jejíž hlavní ideou je předávání rodičovského genomu novým potomkům a následné uvolnění životního prostoru potomkům. Ovšem, ani Darwin, příp. Mendel nebyli první, kteří se touto teorií zabývali. Již ve starověku se vyskytli myslitelé, kteří přišli se stejnou myšlenkou jako Darwin a Mendel. Výrazným myslitelem, který již před Darwinem propagoval myšlenku evoluce, byl Anaximandros z Milétu, občan maloasijského iónského města Milétu. Anaximandrovy filosofické názory jsou shrnuty v jeho nedochovaném filosofickém spise, který nese název „O přírodě“. Toto pojmenování pochází až z pozdějších dob. Původním principem světa, příčinou všeho bytí, je podle Anaximandra tzv. neomezeno (řecky *apeiron*), které se pak dále člení na studené a teplé a suché a vlhké - v podstatě si lze tento princip představit ve smyslu neomezené a nediferenciované vlhkosti, z níž pak postupně vznikají další přírodní látky i jednotlivé druhy živočichů. (1)

Svojí myšlenkou, že Země, kterou si představuje jako volně se vznášející v prostoru, byla nejprve kapalném stavu a teprve později, při svém vysoušení dala vznik živočichům, kteří nejprve žijí ve vodě a později přesídlují na zem, anticipuje zčásti Anaximandros moderní vývojovou teorii. (1)

Myšlenka EVT je založena na existenci tzv. evolučních algoritmů (*evolutionary algorithms*, EA), které v podstatě tvoří větší část EVT. Mimo evoluční algoritmy existují ještě další rozšíření, jako jsou genetické programování, evoluční hardware apod. Lze tedy konstatovat, že základem EVT jsou evoluční algoritmy. Podle klasické Darwinovy a Mendelovy teorie evoluce je uznáváno evoluční dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku různým mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají "cyklicky" po tzv. generacích (*generations*), čímž uvolňují místo novým lepším potomkům, kteří se stávají novými rodiči. Schéma principu evoluce je znázorněno na Obrázek 1.1 Obecný cyklus evolučního algoritmu dle , str 27. (Není zobrazen poslední krok algoritmu, což je ukončení evoluce po n generacích a výběr nejlepšího jedince = výsledné nejlepší řešení) (1)



Obrázek 1.1 Obecný cyklus evolučního algoritmu dle (1), str 27

1.2 Základní postup evolučních algoritmů

Definice parametrů evoluce

Každý algoritmus musí mít definovány parametry, které řídí běh algoritmu, anebo jej regulérně ukončí, jsou-li naplněna předem stanovená kritéria ukončení (např. počet cyklů - generací). Součástí tohoto procesu je i stanovení účelové funkce (*cost function*), případně tzv. vhodnosti (*fitness*), což je upravená návratová hodnota účelové funkce (obvykle normalizovaná hodnota účelové funkce). (1)

Účelovou funkcí se rozumí obvykle matematický model problému, jehož *minimalizace* či *maximalizace* (obecně tedy *extremalizace*) vede k řešení problému. Tato funkce s případnými omezujícími podmínkami je jakýmsi "ekvivalentem životního prostředí", v němž se vyhodnocuje kvalita aktuálních jedinců. (1)

Vytvoření počáteční populace

Počáteční populace je obecně matice $N \times M$ kde N je počet parametrů jedince a M je počet jedinců v populaci. Podle počtu optimalizovaných argumentů účelové funkce a uživatelských kritérií je vygenerována prvotní populace tzv. jedinců (*individuals*). Jedincem se rozumí vektor čísel, který má tolik složek, kolik je optimalizovaných parametrů účelové funkce. Tyto složky jsou nastaveny nahodile a každý jedinec tak představuje jedno možné konkrétní řešení problému. Takovou množinu jedinců nazýváme populace (*population*).

Analýza jedinců

Všichni jedinci se ohodnotí přes definovanou účelovou funkci a každému z nich se přiřadí :

- a) přímá hodnota navrácená účelovou funkcí
- b) vhodnost

Výběr nejlepších jedinců

Nastává výběr rodičů podle jejich kvality (vhodnost, hodnota účelové funkce), případně i podle dalších kritérií – tzv. selekce (*selection*)

Křížení jedinců

Křížením rodičů se tvoří potomci. Proces křížení je u každého algoritmu odlišný. U klasických genetických algoritmů jsou přehozeny části rodičů, u diferenciální evoluce je křížení jistou vektorovou operací apod.

Mutování potomků

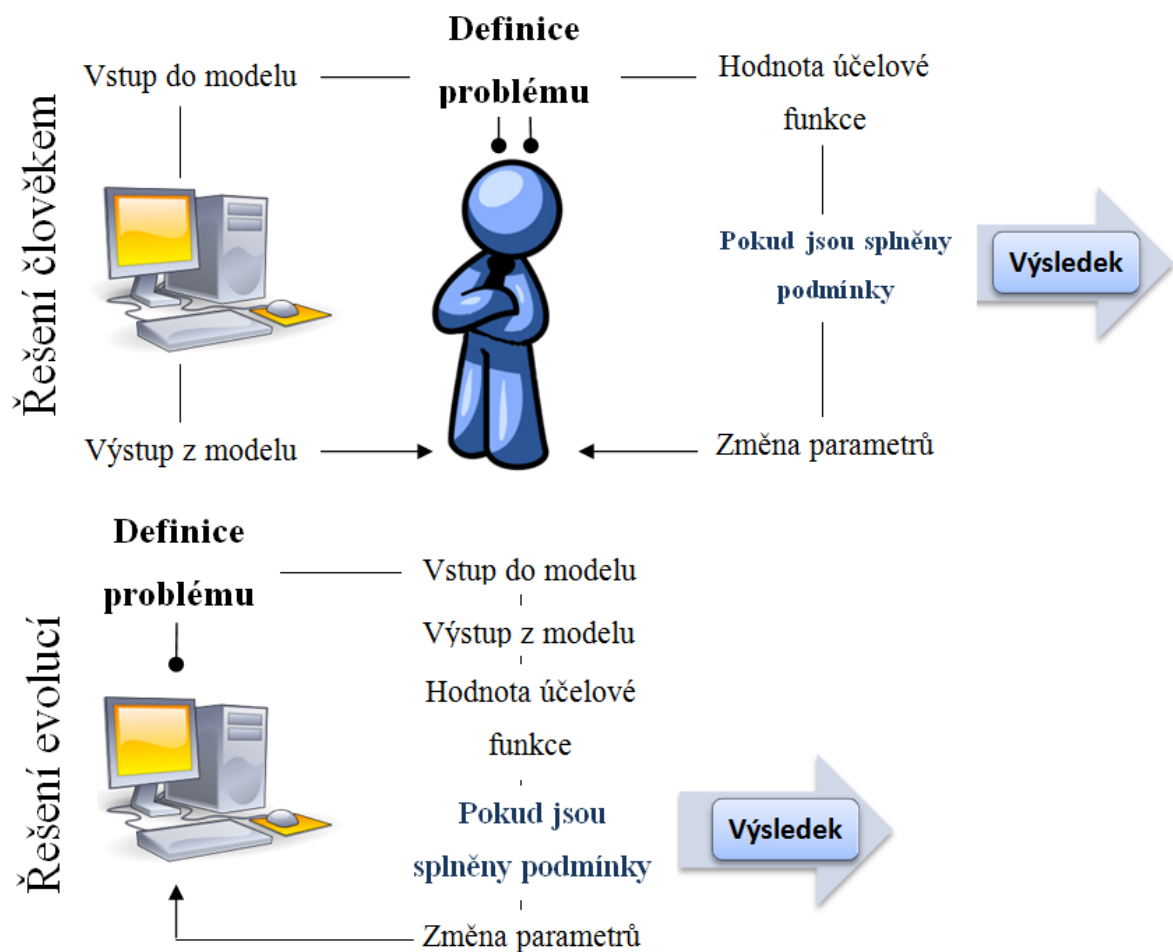
Každý potomek je zmutován (*mutation*). Jinými slovy, nový jedinec je pozměněn pomocí vhodného náhodného procesu. Tento krok je ekvivalentem biologické mutace genů jedince.

Cyklické opakování

Každý nový jedinec se opět ohodnotí, viz výše a vyberou se nejlepší z nich a vytvoří tak novou populaci. Stará populace je zapomenuta (zlikvidována, vymazána, umírá) a na její místo je zvolena populace nová. Celý cyklus se poté opakuje do té doby, než dosáhneme cíle s uspokojivým výsledkem, anebo naplníme zadaný počet kroků evoluce.

Princip evolučního algoritmu naznačený výše je obecný a v konkrétních případech se může více či méně lišit. Existují i výjimky, které se neřídí těmito kroky. V takovém případě se o příslušných algoritmech nemluví jako o algoritmech evolučních, ale jako o algoritmech, které patří k EVT.

Evoluční algoritmy nejsou populární jen proto, že jsou moderní a odlišné od algoritmů klasických, ale hlavně pro fakt, že v případě vhodného aplikování jsou schopny nahradit člověka. To je zobrazeno na Obrázek 1.1 Obecný cyklus evolučního algoritmu dle , str 27. Jsou v něm zobrazeny dva způsoby řešení problému. První představuje kroky lidského řešitele a druhý postup při použití EVT. (1)



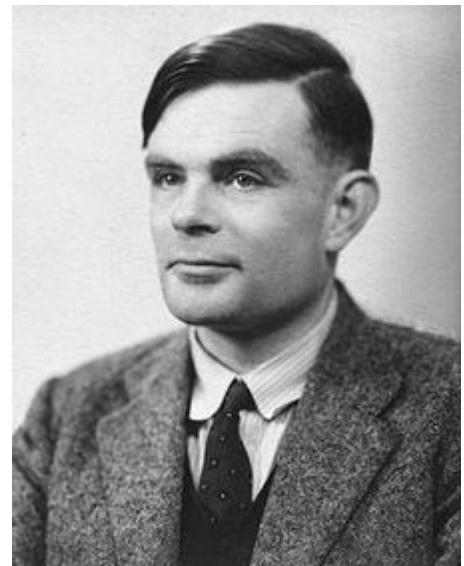
Obrázek 1.2 Porovnání řešení problémů pomocí EVT a člověka dle (1), str. 29

1.3 Historie

Začátek historie evolučních algoritmů se obvykle datuje do poloviny 70. let, kdy se poprvé objevily genetické algoritmy, případně do poloviny let 60., kdy byly poprvé s úspěchem použity tzv. evoluční strategie. Vezmeme-li však v úvahu všechna fakta, lze s jistotou tvrdit, že duchovními otci evolučních výpočetních technik jsou takové osobnosti, jako byl matematik **A. M. Turing** nebo **N. A. Barricelli** a jiní. Již v této době byly formulovány a definovány principy, které zcela jasně popisují principy evolučních algoritmů. To, že nebyly programově realizovány, bylo zřejmě dáno nedostatkem výkonné výpočetní techniky.

V eseji s názvem „Inteligentní stroje“ z roku 1948 Turing uvádí :

" ... if the untrained infant's mind is to become an intelligent one, it must acquire both discipline and initiative ... discipline is certainly not enough in itself to produce intelligence. That it is required in addition we call initiative ... our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines ... " (1)



Obrázek 1.3 Alan Mathison Turing dle (2)

Pokud se pokusíme volně parafrázovat :

„...inteligentní mysl musí zahrnovat dvě základní oblasti aktivit, a to disciplínu a iniciativu. Disciplína sama o sobě nestačí k existenci inteligence. To, co je ještě potřeba, nazýváme iniciativou a úkolem budoucnosti, je nalézt, jak ji lze přenést do strojů.“ (1)

1.4 Oblasti použitelnosti evolučních algoritmů

Dnes můžeme říci, že existuje opravdu velké množství algoritmů, které lze zařadit do kategorie evolučních algoritmů. Mezi typické příklady patří například *Ant Colony Optimization* (ACO), *Immunology System Method* (ISM), *Scatter Search* (SS) nebo také *Particle Swarm* (PS). Ne všechny výše jmenované algoritmy lze dle libosti používat a považovat je za univerzální evoluční algoritmy. Každý z nich, jak je už z názvu patrné, využívá jiných postupů, a tedy se hodí na řešení specifických problémů ve specifických oblastech. (1)

Velikost množiny problémů, které lze daným algoritmem řešit, může být libovolná pro každý algoritmus. Například genetické algoritmy lze použít na poměrně širokou třídu problémů, zatímco ACO algoritmus, pracující na principu simulace chování mravenců, je v podstatě předurčen k řešení kombinatorických problémů typu např. *Obchodní Cestující*. Zde dosahují ACO excelentních výkonů. (1)

Z těchto informací tedy vyplývá, že nestačí mít jen výborný algoritmus, ale je nutné znát, velice přesně, také třídu problémů, se kterými je daný algoritmus schopen pracovat. Je nutné stanovit tzv. *oblast použitelnosti* algoritmu. V oblasti evolučních algoritmů lze tuto oblast chápat jako třídu problémů, na níž bude dávat daný algoritmus uspokojivé výsledky.

Každý optimalizační problém si lze znázornit či představit, jako problém geometrický, u něhož je cílem najít *minimum* nebo *maximum* na N rozměrné ploše. (1)

1.5 Společné rysy

Evoluční algoritmy mají několik společných vlastností:

- Jednoduchost
 - Hlavní myšlenka je poměrně prostá
 - Naprogramování je obvykle velmi jednoduché
- Schopnost pracovat s diskrétní množinou čísel
 - Můžeme použít pouze vybrané množiny čísel
 - Lze kombinovat bez problému např. čísla typu *real* a *integer*

- Používání dekadických čísel
 - Není nutný převod *DEC* -> *BIN*, tj. nedochází ke vzniku zkreslení během tohoto převodu.¹
- Rychlost
 - Díky své jednoduchosti jsou mnohem rychlejší, než běžně používané metody
- Existence vícenásobného řešení
 - Pokud řešený problém vykazuje více globálních extrémů, lze očekávat, že budou nalezeny. K dispozici tedy bude více stejně kvalitních řešení.

Populace

Typickým rysem evolučních algoritmů je, že jsou založeny na práci s populací jedinců. Populace může být znázorněna jako matice $N \times M$ (), kde sloupce představují jednotlivé jedince. Každý jedinec představuje aktuální řešení daného problému. V podstatě je to množina argumentů účelové funkce, jejichž optimální číselná kombinace je postupně hledána, S každým jedincem je navíc spojena hodnota účelové funkce (někdy též vhodnost – *fitness*), která říká, jak vhodný je jedinec pro další vývoj populace. Tato hodnota se neúčastní samotného evolučního procesu. Nese pouze informaci a kvalitě každého jedince. (1)

Hlavní činností evolučních algoritmů je cyklické vytváření nových populací, tedy náhrada starých populací novými. To vše se děje pomocí přesně definovaných matematických pravidel. Použitá pravidla a reprezentace jedinců v populaci určuje, o jaký evoluční se jedná. Vzhledem k tomu, že se populace v čase vyvíjí (stará je nahrazena novou), říká se tomuto druhu algoritmů "evoluční". (1)

¹ Genetické algoritmy pracují běžně s binárním kódem. Během převodu z dekadického tvaru čísla na binární tvar vzniká zkreslení, které je dáno omezenou délkou binárního řetězce. U binárního zápisu mohou mutace také způsobit skokovou změnu čísla, což může nepříznivě ovlivnit průběh evoluce.

Tabulka 1.1 Populace jedinců s velikostí $N \times M$

	J1	J2	J3	J4	JM
VHODNOST		55.2	68.3	5.36	9.5			0.89
P1		2.55	549.3	-55.36	896.5			1.89
P2		0.25	66.2	2	-10			-2.2
P3		-66.5	55	4.5	15.8			-75.82
...								
PN		266	-11	23.33	533.56			-41.56

J_i – i -tý jedinec P_j – j -tý parametr jedince, Vhodnost – kvalita jedince produkovaná účelovou funkcí.

Abychom mohli vytvořit populaci, je zapotřebí tzv. populační vzor (*specimen*), viz (1). Podle tohoto vzoru se poté generuje kompletní počáteční populace. Tento vzorový jedinec se rovněž používá pro korekci parametrů jedinců, kteří překročí hranice prohledávaného prostoru. (1)

$$Specimen = \{ \{Real, \{Lo, Hi\} \}, \{Integer, \{Lo, Hi\} \}, \dots, \{Real, \{Lo, Hi\} \} \} \quad [1]$$

Ve vzoru jsou pro každý parametr konkrétního jedince z populace definovány tři konstanty, a to typ proměnné (tj. celočíselný, reálný, diskrétní, atd.) a hranice intervalu, v němž se může pohybovat hodnota parametru. Například $\{Integer, \{Lo, Hi\}\}$ definuje celočíselný parametr se spodní hranicí Lo a horní Hi . Volba hranic je velmi důležitý krok, protože při jejich nevhodném zvolení se může stát, že budou nalezena řešení, která nebude možné fyzikálně realizovat, nebo nebudou mít opodstatnění. (1)

Další neméně důležitý význam hranic souvisí se samotným evolučním procesem. Může se stát, že daný optimalizační problém bude reprezentován plochou, na níž budou nabývat lokální extrémů stále větších hodnot se vzrůstající vzdáleností od počátku. To způsobí, že evoluce bude nacházet stále nová řešení až do nekonečna. Samozřejmě pokud nebude specifikováno zastavení v závislosti na počtu evolučních kol (generací, migrací, žihání). Je to způsobeno tím, že evoluční proces směřuje do stále hlubších a vzdálenějších extrémů. (1)

Populace je na základě vzorového jedince vygenerována pomocí (2). $P_{(0)}$ představuje prvopočáteční populaci, $X_{i,j}$ je j-tý parametr i-tého jedince.

$$P_{i,j}^{(0)} = x_{i,j}^{(0)} = \text{Rand}[0,1] \cdot (x_{i,j}^{(Hi)} - x_{i,j}^{(Lo)}) + x_{i,j}^{(Lo)} \quad [2]$$

$$i = 1, \dots, M, j = 1, \dots, N$$

Tento vztah zajišťuje, že všechny parametry jedinců budou náhodně generovány uvnitř povolených hranic prostoru možných řešení. (1)

1.6 Optimalizační a evoluční techniky

1.6.1 Krátký přehled vybraných algoritmů

Slepý algoritmus

Tento enumerativní algoritmus je první snahou o nalezení optimálních hodnot na účelové funkci. Jeho cílem je spočítat hodnotu účelové funkce ve všech jejích bodech a následně vybrat maximální či minimální hodnotu. Takový postup bude zcela jistě časově velmi náročný a tedy neefektivní. Tento algoritmus je také nazýván jako „*Náhodná procházka*“. (1)

Horolezecký algoritmus

Základem tohoto algoritmu je myšlenka, že efektivnějšího výsledku se dosáhne v případě, když budeme zkoumat hodnoty účelové funkce ve směru největšího gradientu. Vždy se vygeneruje okolí, které bude prohledáváno, nalezneme se maximální či minimální hodnota účelové funkce a současně se zaznamená nejlepší nalezené řešení. Každé nalezené maximum (minimum) je pak použito jako střed pro generování nového okolí (tzn. i v případě, že nalezená hodnota je horší než aktuální nejlepší řešení). (1)

Slabinou horolezeckého algoritmu je problém *cyklických řešení*, což znamená, že nelze vyloučit situaci, kdy algoritmus po skončení uváže v lokálním extrému. Tomuto problému lze účinně předcházet pomocí různých modifikací horolezeckého algoritmu.

Simulované žíhání

Simulované žíhání patří mezi algoritmy, které dovolují přijímat i horší výsledky než jsou aktuálně nejlepší dosažené. To umožňuje vyvarovat se uváznutí v lokálním extrému (hlavně v počátcích běhu algoritmu). V konečné fázi se tento algoritmus chová stejně jako

horolezecký algoritmus. Simulované žíhání je inspirováno fyzikálním procesem *žíhání tuhého tělesa*. I tento algoritmus existuje v několika modifikacích. (1)

Genetické algoritmy

Genetické algoritmy (GA) jsou jedny z nejznámějších EVT. Poměrně často jsou zaměňovány s evolučními algoritmy, což není zcela správné, protože genetické algoritmy jsou v dnešní době třídou algoritmů, která vychází z klasických genetických algoritmů.

Jsou založeny na principech evoluce, používané v přírodě a popsané Charlesem Darwinem a J. G. Mendelem. Většina terminologie (populace, jedinec, generace) byla tedy převzata z biologie. Typickou vlastností genetických algoritmů je, že pracují s binárním kódováním parametrů jedinců. (1)

Paralelní genetické algoritmy

Paralelní genetické algoritmy (PGA) využívají paralelních výpočtů k získání globálních extrémů, které vedou mnohem rychleji k cíli než běžné genetické algoritmy. Stejně tak jako rychlost výpočtu je důležitá také kvalita řešení. Existuje zde několik nezávislých populací, které si mezi sebou vyměňují vybrané informace. Není zde tedy jedna dominantní populace, nýbrž více populací, ze kterých mohou vzejít kvalitnější řešení. (1)

Evoluční strategie

Evoluční strategie (ES) se liší od genetických algoritmů v několika ohledech (1):

- Reprezentace jedinců ES je v oboru reálných čísel (GA v binárních číslech)
- ES nepoužívá operátor křížení – dochází pouze k mutaci a selekci, viz 1.2

ES byly vyvinuty v několika variantách (1):

- Dvoučlenné ES
- Vícečlenné ES
- Rekombinativní ES
- Sebeadaptivní ES

Particle Swarm

Particle swarm neboli česky „*rojení částic*“ je optimalizační algoritmus, pracující podobně jako GA. Inspiraci hledá v sociálním chování živočišných společenstev jako je například hejno ptáků či hejno ryb. Algoritmus je iniciován populací náhodných řešení (jedinců) obdobně jako u GA a hledá se optimální řešení vytvářením nových, lepších generací.

Rozdíl mezi GA a PSO je v tom, že PSO nepoužívá žádné evoluční operátory jako je křížení či mutace. Potenciální řešení následují v řešeném prostoru trajektorie těch částic, které mají nejlepší vhodnost (1), viz 1.2

Scatter Search

Scatter search (SS) neboli česky „rozptýlené hledání“ pracuje na množině řešení a referenční množině. Tyto množiny poté kombinuje a vytváří tak nová řešení pomocí lineární kombinace. (1)

Ant Colony Optimization

Ant colony optimization (ACO) neboli česky „optimalizace mravenčí kolonií“ je algoritmus, jehož přístup by se dal označit jako hledání dobré cesty přes grafy. Inspirace pochází opět z biologie – chováním skutečných mravenců v koloniích při hledání a sbírání potravy. V přírodě se mravenci pohybují nahodile. Kdy najdou potravu, vrací se zpět do mraveniště, přičemž za sebou zanechávají stopu feromonu. Když pak další mravenci během svého náhodného pohybu na tuto stopu narazí, začnou se pohybovat přesně podle trajektorie feromonové stopy, dokud nenajdou potravu. Poté se celý proces opakuje. Mravenci postupně opouští mraveniště za účelem najít potravu, další mravenci se do mraveniště vrací a zanechávají stopu, která je čím dál tím víc intenzivnější. Za těchto podmínek se kratší cesta stává využívanější – intenzivnější a naopak cesta delší, kde feromon postupně vyprchá, je pro mravence méně atraktivní. (1)

Umělý imunitní systém

Umělý imunitní systém (UIS) je oblastí výzkumu, který vytváří spojení mezi inženýrstvím a imunologií. Vytváří se matematické modely imunity, které slouží jako základ pro optimalizační algoritmy. UIS bere inspiraci z oblastí imunitních systémů u obratlovců. Zjednodušeně vysvětleno – algoritmy prohledávají možnosti charakteristických vlastností imunity v procesu učení a paměti a využívají je při řešení daných problémů. (1)

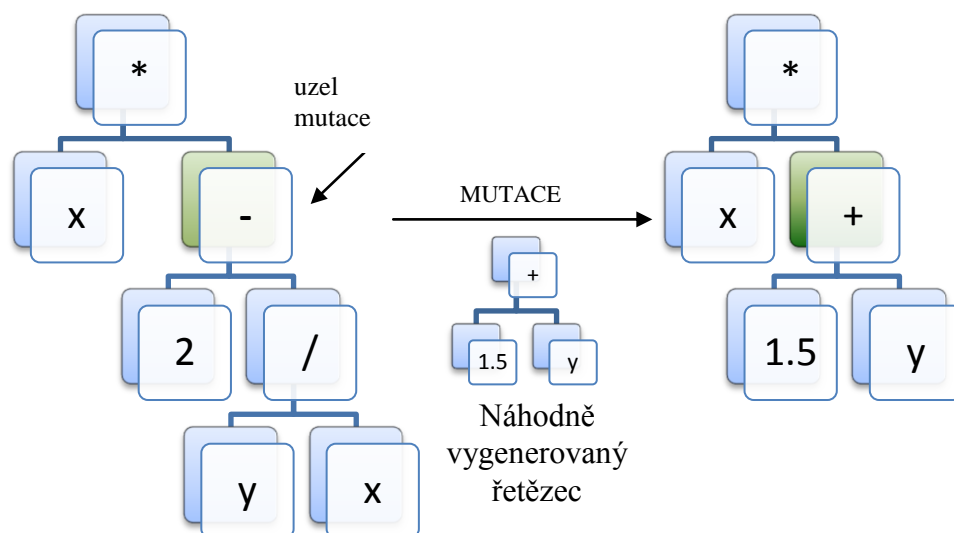
Genetické programování

Genetické programování (GP) vzniklo na konci 80. let minulého století. Jeho tvůrcem je John Koza. Navrhl modifikaci genetických algoritmů pro tvorbu tzv. programů (uživatelské programy, funkce, ...) v rámci symbolické regrese a pojmenoval jej genetické programování - *genetic programming*. Hlavní modifikací algoritmu bylo to, že nová populace nebyla *šlechtěna* klasickým způsobem, ale na symbolické úrovni. To znamená, že populace nebyla tvořena množinou čísel, nýbrž množinou symbolů, symbolických objektů, matematických funkcí atp. V podstatě jde o to, že ze základních objektů typu elementární funkce, konstanta, proměnná nebo operátor, je vytvořen vhodnější složitější tvar, který popisuje požadované chování daného problému. Například u aproximace funkcí by základními objekty mohly být matematické operátory, plus, minus, Cos, Sin, proměnná x , konstanty – např. číslo π , či jiná čísla. (1) (14)

Zjednodušeně řečeno – hledá se nejvhodnější výraz pro popis určitého problému v symbolickém tvaru.

Pro snadnější zobrazení se v kanonické verzi GP používá stromová struktura, viz Obrázek 1.4 Příklad stromové struktury s ukázkou možné mutace dle , str. 256.

Vrchol je nazýván kořenem a čtení této stromové struktury se provádí zleva zdola směrem ke kořenu.



Obrázek 1.4 Příklad stromové struktury s ukázkou možné mutace dle (1), str. 256

Gramatická evoluce

Gramatická evoluce (GE) je určitý typ genetického programování (GP), které je založeno na gramatice. Pomocí GE můžeme vytvářet *programy* v libovolném jazyce, pokud použijeme vhodnou gramatiku – tzv BNF gramatiku. Taková gramatika se skládá z terminálů, což jsou objekty, které se v daném jazyce mohou vyskytovat, a neterminálů. Neterminál může být nahrazen jedním nebo více terminály a neterminály. Neterminální symbol je takový symbol, který může být přepsán na jiný řetězec symbolů, zatímco terminální symbol již nemůže být přepsán. (1) (13)

1.6.2 SOMA: SamoOrganizující se Migrační Algoritmus

SOMA je algoritmus existující od roku 1999, jehož činnost je založena vektorových operacích. Vzhledem k tomu, že pracuje s populacemi podobně jako např. genetické algoritmy a výsledek po jednom evolučním cyklu (migračním kole) je totožný s genetickými algoritmy či diferenciální evolucí (je vytvořena nová populace), lze jej řadit např. mezi evoluční algoritmy navzdory faktu, že během jeho běhu nejsou z hlediska filozofie algoritmu vytvářeni noví potomci, jak je tomu u jiných klasických evolučních algoritmů. Pokud by se hledala biologická analogie, pak by se dal tento algoritmus spíše přirovnat k harémové tvorbě potomků ve stádu než ke klasickému výběru rodičů z populace. (1)

Mnohem přesnější je však řazení mezi algoritmy *memetické* či *hejnové*. Původní myšlenka, která vedla k jeho vytvoření, spočívá v napodobení chování skupiny inteligentních jedinců, kteří kooperují při řešení společného problému, jako je např. hledání zdroje potravy apod. SOMA od své základní verze doznal několik důležitějších změn, až do dnešní podoby, kdy se svou robustností ve smyslu nalezení globálního extrému vyrovná mnohým evolučním algoritmům. (1)

SOMA byl vyvinut na principech, které lze pozorovat v přírodě a kterými se v sociálně-biologickém prostředí řídí inteligentní jedinci, jenž kooperují na řešení společného úkolu. Na rozdíl od ostatních evolučních algoritmů v něm totiž neprobíhá tvorba nových řešení (jedinců, potomků) filozofií křížení rodičů, ale je založena na kooperativním prohledávání (migraci) prostoru možných řešení daného problému. Vzhledem k tomu, že vlastní jádro SOMA nekopíruje již zmíněné evoluční principy, ale řídí se principy vycházejícími ze spolupráce inteligentních jedinců migrujících v prostoru možných řešení tak jako jejich biologické protějšky po krajině, byl pro evoluční cyklus známý jako "generace" zvolen název *migrační kolo*. Příklady takového chování lze nalézt v reálném světě. Jsou to např. mravenci, včely, predátoři ve smečce hledající potravu apod. Vlastnost samo-organizace u SOMA algoritmu plyne z faktu, že se jedinci ovlivňují navzájem během hledání lepšího řešení, což mnohdy vede k tomu, že v prostoru možných řešení vznikají skupiny jedinců, které se rozpadají či spojují, putují přes prohledávaný prostor. Jinými slovy si skupina jedinců neboli populace sama organizuje vzájemný pohyb jedinců – proto tedy samo-organizace. (1)

1.6.2.1 Parametry a terminologie

Běh algoritmu SOMA, je ovlivňován speciální množinou parametrů, které se dělí na dva druhy parametrů a to na parametry *řídící* a *ukončovací*. Řídící parametry jsou ty, které mají vliv na kvalitu běhu algoritmu a ukončovací jsou ty, které za předem nadefinovaných podmínek běh algoritmu ukončují. Všechny tyto parametry musí být zvoleny uživatelem před začátkem samotného algoritmu. Parametry a jejich doporučené hodnoty pro SOMA algoritmus popisuje (1) (3)

Tabulka 1.2 Význam parametrů SOMA algoritmu, dle (1), str 226

Parametr	Doporučená hodnota	Popis
PathLength	1,1 až 5 a více	Řídící parametr
Step	0,11 až PathLength	Řídící parametr
PRT	0 až 1	Řídící parametr
D	ovlivněno daným problémem	Počet argumentů účelové fce
PopSize	10 až N	Ukončovací parametr (volí uživatel)
Migrace	10 až M	Ukončovací parametr (volí uživatel)
AcceptedError	libovolně zvolený	Ukončovací parametr - velikost chyby

PathLength – určuje, jak daleko se aktivní jedinec zastaví od vedoucího jedince.

PathLength = 1 => aktivní jedinec zastaví na pozici vedoucího jedince

PathLength = 2 => aktivní jedinec zastaví za vedoucím jedincem ve stejné vzdálenosti, ve které od něj startoval

PathLength < 1 => degenerace algoritmu (omezení na hledání lokálních extrémů)

PathLength = 3 => doporučená hodnota parametru pro řešení většiny problémů

Step – určuje zrnitost, s jakou bude mapována cesta aktivního jedince. Při řešení problému jednoduché unimodální účelové funkce je možné použít velkou hodnotu parametru pro urychlení chodu algoritmu. Pokud není účelová funkce známa ani přibližně, doporučuje se nastavit hodnotu na 0,11 – prostor možných řešení bude prohledáván podrobněji. Parametr Step nesmí být celočíselným násobkem parametru PathLength – došlo by ke snížení diverzibility populace a proces by tak mohl rychleji skončit v lokálním extrému.

PRT – tzv. pertubace. Podle tohoto parametru se tvoří pertubační vektor (PRTVector), který ovlivní to, zda se aktivní jedinec bude pohybovat přímo k vedoucímu jedinci či ne. Je to velmi důležitý parametr s obrovskou citlivostí. Optimální hodnota je kolem 0,1. V případě že $PRT = 1$ – dochází k čistě deterministickému prohledávání (stochastická složka je eliminována) – algoritmus je omezen pro hledání lokálních extrémů.

D – parametr, který vypovídá o počtu optimalizovaných proměnných. Závisí přímo na definici problému.

PopSize – Řídicí parametr, který určuje, kolik jedinců bude tvořit každou populaci. Populace může mít běžně 30 – 50 jedinců. Obvykle by hodnota neměla klesnout pod 10.

Migrate – Parametr, který je ekvivalentní k parametru Generace z jiných evolučních algoritmů. V podstatě určuje, kolikrát se populace obrodí – změní. Je to ukončovací parametr.

AcceptedError – Ukončovací parametr definovaný uživatelem. Určuje maximální možný rozdíl mezi nejhorším a nejlepším jedincem v populaci.

1.6.2.2 Princip SOMA

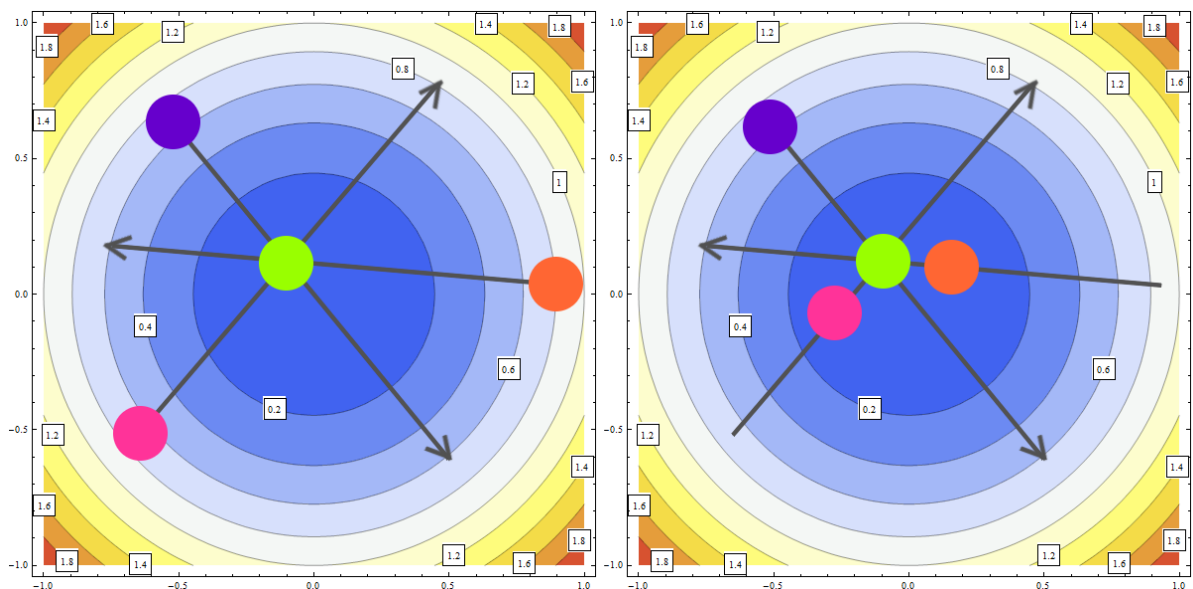
Vznik SOMA byl inspirován soutěživě-komparativním chováním inteligentních jedinců řešících společný problém. Chování tohoto typu lze objevit prakticky kdekoli na světě. Jako příklad lze použít chování smečky lovců vlků, včelího úlu, hejna holubů (J. Nash) apod. U těchto příkladů je společným úkolem např. hledání potravy, v rámci níž jedinci spolupracují, ale i, byť nevědomky, soutěží. Ve fázi spolupráce si navzájem jednotliví jedinci sdělují, jakou kvalitu hledaného momentálně našli a na základě toho se snaží přizpůsobovat své chování. Ve fázi soutěžení (předcházející fázi spolupráce) se každý jedinec snaží vyhrát nad ostatními - snaží se nalézt co nejlepší zdroj potravy apod. Po ukončení fáze soutěže nastane opět fáze spolupráce a jedinci si vymění informace o tom, který z nich má nejlepší zdroj potravy. Ostatní opustí své nalezené zdroje potravy a migrují (fáze soutěžení) směrem k jedinci s nejlepším zdrojem potravy a během této migrace se snaží nalézt ještě lepší zdroj. To se opakuje, dokud se všichni nesejdou u nejvydatnějšího zdroje potravy. Na tomto silně zjednodušeném principu funguje i algoritmus SOMA. (4)

Tabulka 1.3 Význam biologické terminologie v algoritmu SOMA dle (1), str. 226

BIOLOGICKÁ REALITA	IMPLEMENTACE V ALGORITMU
Členové smečky či společenství	Jedinci v populaci – parametr PopSize
Člen smečky s nejlepším zdrojem potravy	Nejlepší jedinec daného kola – migrace
Potrava	Hodnota účelové funkce
Oblast, ve které dané společenství žije	Hyper-plocha definovaná účelovou funkcí
Pohyb členů smečky v obydlené oblasti	Migrační kola SOMA algoritmu

SOMA pracuje v cyklech zvaných migrační kola. Ta hrají tutéž úlohu, jakou mají generace v případě genetických algoritmů. Rozdíl mezi migračními koly a generacemi je spíše filozofického charakteru. Během migračních kol nejsou tvořeni noví jedinci, pouze jsou přemísťováni do finální pozice pomocí sekvence pozic vypočítaných vzhledem k pozici Leadera - migrují prostorem možných řešení. (1)

Následující obrázek ukazuje, jak se jedinci pohybují k leaderovi. Zeleně je vyznačen leader, ostatní body nezornují migrující jedince.



Obrázek 1.5 Konvergence jedinců SOMA k leaderovi, dle (1), (vlastní zpracování)

1.6.2.3 Kroky běhu SOMA

1 Definice parametrů

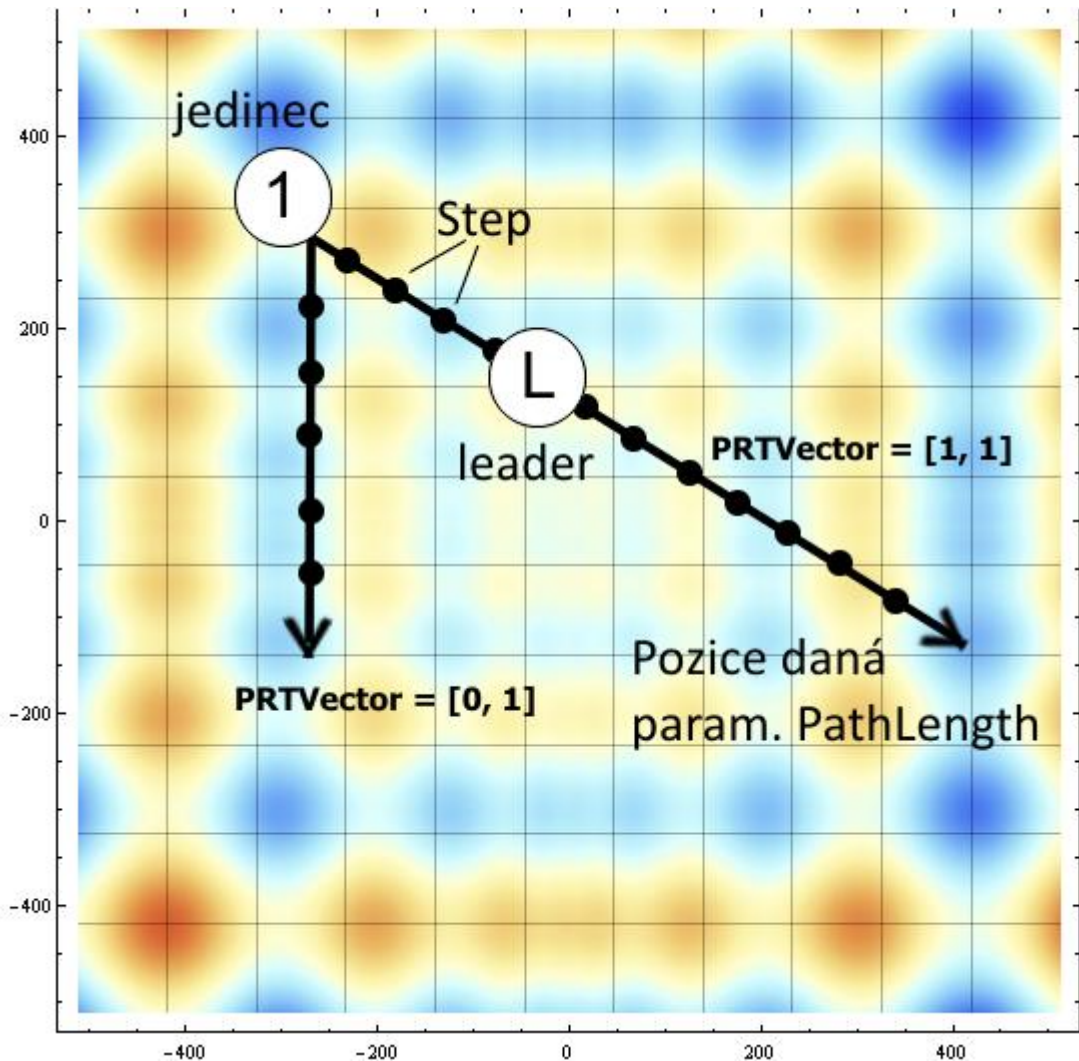
Nejdříve je nutné definovat veškeré potřebné parametry před samotným spuštěním algoritmu – řídicí a ukončovací parametry (*Specimen*, *Step*, *PathLength*, *AcceptedError*, *Popsize*, *PRT* a *Migrace* viz Tabulka 1.2 Význam parametrů SOMA algoritmu, dle [1], str 226). Dalším důležitým krokem, je nadefinovat účelovou funkci, kterou budeme optimalizovat. Tu si lze představit jako životní prostředí pro budoucí jedince z populace. (1)

2 Tvorba populace

V tomto kroku vytvoříme počáteční populaci – náhodně vygenerujeme jedince. S využitím parametru *Specimen* a generátoru čísel je pro každý parametr jedince generováno náhodné číslo (jeho hodnota je dána rozsahem *Specimenu*). (1)

3 Migrační kola – migrace

Každého jedince ohodnotíme účelovou funkcí a určíme mezi jedinci *leadera*. V tuto chvíli dochází k migraci ostatních jedinců směrem k leaderovi po zvolených krocích (parametr *Step*). Znovu dochází k ohodnocení jedinců účelovou funkcí a pokud se u některých jedinců změní hodnota k lepšímu – jedinec si ji zapamatuje. Po skončení migračního kola se všichni jedinci posunou na novou, nejlépe ohodnocenou pozici, vyjma *leadera*. Předtím, nežli daný jedinec započne svou cestu směrem k leaderovi, je vygenerován prázdný *PRTVector* o dimenzi $= D$ včetně vygenerované sekvence náhodných čísel, jejichž počet je roven D . Ty jsou porovnány s parametrem *PRT*. Jestliže je n -té vygenerované číslo větší nežli *PRT* parametr, pak je n -tý parametr *PRTVectoru* nastaven na 0 a v opačném případě na 1. Parametry jedince, které jsou takto nastaveny na 0 se nepřepočítávají, jsou zmrazeny - snižuje se počet stupňů volnosti pohybu jedince. Tento proces nahrazuje mutaci známou u jiných evolučních algoritmů. Díky tomu se rapidně zvyšuje robustnost SOMA algoritmu ve smyslu nalezení globálního extrému. (1) (4)



Obrázek 1.6 Funkce PRT vectoru, dle (4), str. 20 (vlastní zpracování)

4 Zjištění stavu ukončovacích parametrů

Nyní zkontrolujeme, zda je rozdíl mezi leaderem a nejhorším jedincem menší než *AcceptedError*. Stejně tak ověříme, zda došlo k naplnění počtu migračních kol – parametr *Migrations*. Pokud není splněna ani jedna podmínka, algoritmus se vrací do kroku 3.

5 Stop

Získáváme nejlepšího nalezeného jedince (řešení) po ukončení posledního migračního kola.

1.6.2.4 Strategie SOMA

V dnešní době existuje již několik různých variací algoritmu SOMA. Pro jejich obecné označení se používá výraz *strategie* – takové označení lépe poukazuje na to, že algoritmus využívá kooperace jedinců a geometrického přesouvání populace po hyper-ploše. Strategie lze tedy rozdělit takto:

1 AllToOne

„Všichni k jednomu“ – všichni jedinci migrují k leaderovi, vyjma jej samotného.

2 AllToAll

„Všichni ke všem“ - tato strategie neobsahuje žádného leadera. Všichni jedinci migrují ke všem a jediným rozdílem od předchozí *AllToOne* strategie je, že po dokončení migrace aktuálního jedince se daný jedinec vrátí na pozici, kde byl nalezen nejlepší extrém během jeho *PopSize - 1* migračních cest v jednom migračním kole. Tato strategie je náročná na výpočet, ale oto efektivnější je řešení problému.

3 AllToAllAdaptive

„Všichni ke všem adaptivně“ – strategie totožná s *AllToAll* s rozdílem, že aktuální migrující jedinec se k ostatním jedincům přesouvá po každé migraci, když je nalezeno lepší řešení (jedinec je lépe ohodnocen účelovou funkcí). Poté jedinec migruje ke zbylým jedincům.

4 AllToOneRand

„Všichni k jednomu náhodně“ – strategie, kde opět existuje leader a jedinci se snaží k němu přiblížit. Leader však není vybírán podle nejlepšího ohodnocení, ale náhodným výběrem. Zde je tedy možná určitá modifikace algoritmu, kdy leader bude vybírán daným algoritmem.

5 Clusters

„Svazky“ – Proces vytváření svazků v SOMA si lze jednoduše představit tak, že jedinci jsou reprezentováni nějakou sub-populací. V každé takové sub-populaci pak probíhá SOMA samostatně za účelem výběru nejlepšího jedince a následné migrace – z toho vyplývá, že svazky se mohou spojovat a rozpadat, čímž je opět podtržen efekt kooperace.

1.6.3 DE: Diferenciální Evoluce

Diferenciální evoluce (Differential Evolution, DE) je stejně jako SOMA poměrně nový typ EA (od r. 1995), který vyvinul a poprvé použil Ken Price a Rainer Storm. Jeho schéma je dost podobné algoritmům genetickým, s nimiž má několik společných rysů, jako je například tvorba potomků (zde však pomocí 4 rodičů a ne 2 jak je tomu u genetických algoritmů), používání tzv. generací apod. (3)

1.6.3.1 Parametry a terminologie

Diferenciální evoluce je ovlivněna, tak jako ostatní evoluční algoritmy, řídicími parametry. V následující tabulce si uvedeme jejich přehled a poté popíšu jejich význam a označení. (3)

Tabulka 1.4 Řídicí parametry diferenciální evoluce, dle (1), str. 237

Řídicí parametr	Interval	Doporučená hodnota	Popis
NP	10D až 100D	10D	Velikost populace
F	0 až 2	0,3 – 0,9	Mutační konstanta
CR	0 až 1	0,8 – 0,9	Práh křížení
Generations	Volí uživatel		Počet generací (kol)

NP – Parametr, který udává velikost populace. Velikost desetinásobku dimenzí (proměnných účelové funkce) je poměrně dostatečná pro řešení většiny problémů.

V případě vysoce multimodální funkce je vhodné volit velikost populace až na stonásobek dimenzí problému. Minimální hodnota je 4 – při této velikosti populace je schopna diferenciální evoluce spolehlivě pracovat.

F – Mutační konstanta – řídicí parametr diferenciální evoluce. Rozhoduje o míře mutace jedinců.

CR – Jinými slovy jedná se o tzv. *práh křížení*. Vždy je vhodnější volit hodnotu v mezích intervalu nikoliv přímo mezní hodnoty 0 nebo 1.

CR = 0 => Nedojde k mutaci jedince – nový jedinec bude pouze kopií aktuálního (čtvrtého) rodiče – vývoj evoluce se tedy zastaví

CR = 1 => Zkušební jedinec bude tvořen pouze ze tří náhodně vybraných jedinců (patřících do populace) – diferenciální evoluce bude poté mít charakter spíše náhodného prohledávání nežli evolučního algoritmu

Generation – Udává počet evolučních cyklů, tzv. generací, během nichž se celá populace bude vyvíjet.

1.6.3.2 Princip a kroky DE

Hlavním cílem diferenciální evoluce je v daných krocích – generacích – vyšlechtit co nejlepší populaci jedinců ve smyslu maximalizace, či minimalizace hodnot účelové funkce. Každá generace se skládá z následujících částí:

1 Definice parametrů

Podobně jako u SOMA je nutné nejdříve stanovit potřebné řídicí parametry, které určují chod celé evoluce. Jsou to parametry F, CR, NP, D (viz Parametry a terminologie SOMA) a Specimen (určuje, z jakých typů čísel se bude jedinec skládat).

2 Tvorba populace

Dle typového vektoru Specimen dojde k vygenerování populace (matice). Každý jedinec (vektor) obsahuje kromě parametrů také hodnotu účelové funkce.

3 Cyklus generací

Během každé generace běží proces, který zabezpečí postupný evoluční vývoj každého jedince k lepšímu. V tomto procesu se postupně vybírají jedinci až do konce populace a pro každého z nich je proveden následující evoluční cyklus.

4 Evoluční cyklus

Zde se provádí mutace a křížení (viz Obrázek 1.4 Příklad stromové struktury s ukázkou možné mutace dle , str. 256). Náhodně se zvolí další tři jedinci (vektory), první dva se od sebe odečtou – získáme tak **diferenční vektor**. Tento vektor vynásobíme mutační konstantou F, která jej tím pádem ovlivní (zmutuje, změní) a získáme tak **váhový diferenční vektor**. Tento vektor pak přičteme k třetímu náhodně vybranému vektoru a získá se tak **šumový vektor**. Nyní je vytvořen tzv. **zkušební vektor**. Poté dochází k výběru dvojic jedinců z vektoru šumového a cílového a pro každou takto vybranou dvojici se generuje číslo od 0 do 1 a porovnává se s konstantou CR. Pokud je toto číslo menší než CR, ze zkušebního vektoru je na příslušnou pozici šumového vektoru přiřazen daný prvek. V opačném případě – číslo je větší než CR – se prvek bere z vektoru cílového. Pokud určíme hodnotu účelové funkce pro každý z těchto vektorů a porovnáme je, pak na pozici

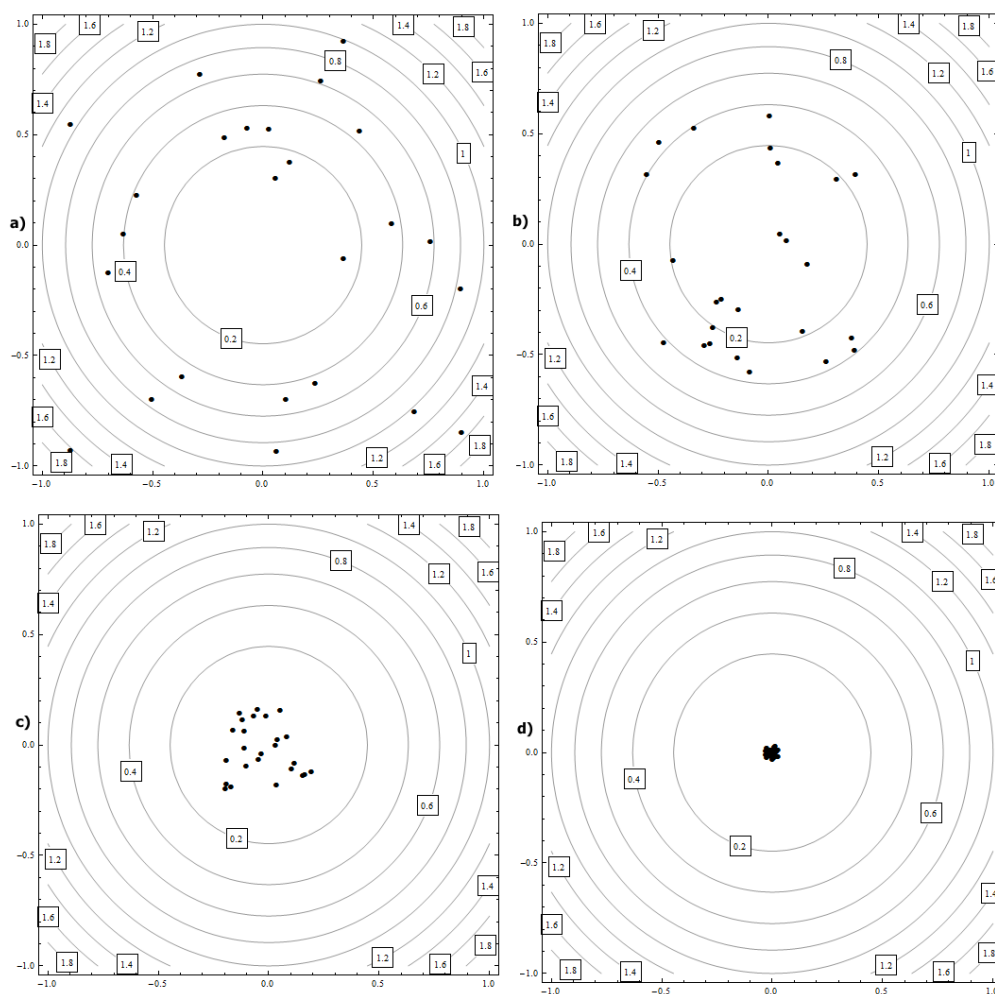
cílového vektoru v nové populaci bude vybrán jedinec (vektor), který má lepší hodnotu účelové funkce. Tento proces se opakuje pro každého jedince v populaci.

5 Zjištění stavu ukončovacích parametrů

Diferenciální evoluce končí pouze tehdy, když se naplní uživatelem zadaný počet generací. Jiný ukončovací parametr tento algoritmus nemá – lze jej však bez problémů zajistit naprogramováním.

6 Vyhodnocení

Každá generace produkuje jedince s nejlepší hodnotou účelové funkce. Tyto hodnoty lze libovolně zpracovat (vykreslit graf). Tímto proces končí a opakují se body 3 – 6 dokud se nenaplní zadaný počet generací. Průběh generací DE algoritmu ukazuje zjednodušeně obrázek: a) Začátek b) Průběh generace c) Pokročilá evoluce d) Konec evoluce



Obrázek 1.7 Konvergence populace do globálního extrému (vlastní zpracování)

1.6.4 Metaevoluce

Jak již bylo zmíněno, SOMA i DE jsou závislé na řídicích parametrech a na jejich nastavení. Tyto parametry jsou voleny jako pevné, nicméně je lze během evoluce měnit podle toho, jak kvalitně probíhá. Pokud se na nastavení parametrů diferenciální evoluce použije diferenciální evoluce samotná, pak se mluví o meta-diferenciální evoluci. Totéž platí i o SOMA. V rámci meta-diferenciální evoluce jsou šlechtěny parametry evoluce tak, že pro populaci těchto parametrů se provádí opět diferenciální evoluce daného problému za použití aktuálního jedince – parametrů (F, CR, NP). Lze říci, že se jedná o diferenciální evoluci na diferenciální evoluci.

Totéž lze analogicky přenést i na SOMA. Parametry evoluce mohou být rovněž šlechtěny použitím *meta* přístupu. Je nutné si ovšem uvědomit, že *meta* přístup je časově daleko více náročný než přístup klasický.

Kvalitu a průběh šlechtění lze ovlivnit mnoha způsoby:

Nastavením řídicích parametrů, velikostí populace, počtem generací, definicí účelové funkce, definice omezení. (3)

2 EVOLUČNÍ SYNTÉZA SYMBOLICKÝCH STRUKTUR

Evoluční syntéza symbolických struktur je v podstatě proces, během kterého, za pomoci téměř libovolného evolučního algoritmu a ve spolupráci s nástrojem zvaným analytické programování, viz kapitola 2.2, lze z určité množiny symbolů, matematických vzorců, funkcí a konstant, syntetizovat matematickou strukturu – funkci. Obecněji je tento proces nazýván jako symbolická regrese. (15)

2.1 Symbolická regrese

Symbolická regrese je proces, kdy z jednoduchých *stavebních kamenů* skládáme složitou strukturu, která popisuje určité chování – funkci. Pomocí této metody lze například aproximovat naměřená data a určit tak funkční závislost mezi nimi. Pojem symbolická regrese v souvislosti s evolučními algoritmy poukazuje na několik nástrojů, z nichž nejznámější je genetické programování, viz 0. Tato práce se zabývá symbolickou regresí s využitím experimentálního přístupu zvaného analytické programování, viz 2.2. (1) (5)

2.2 Analytické programování

2.2.1 Hlavní myšlenka

Algoritmus analytického programování (AP) dosáhl výborných výsledků během řešení problémů symbolické regrese typu: syntéza trigonometrických funkcí (6), polynomiálních funkcí (7), funkce boolovské parity (8), funkce boolovské symetrie (9), řešení diferenciálních rovnic (10) a optimalizace cesty umělého mravence (11). Základní vlastnosti AP, které z něj činí algoritmus k tomuto účelu vhodný, jsou popsány v následujících podkapitolách.

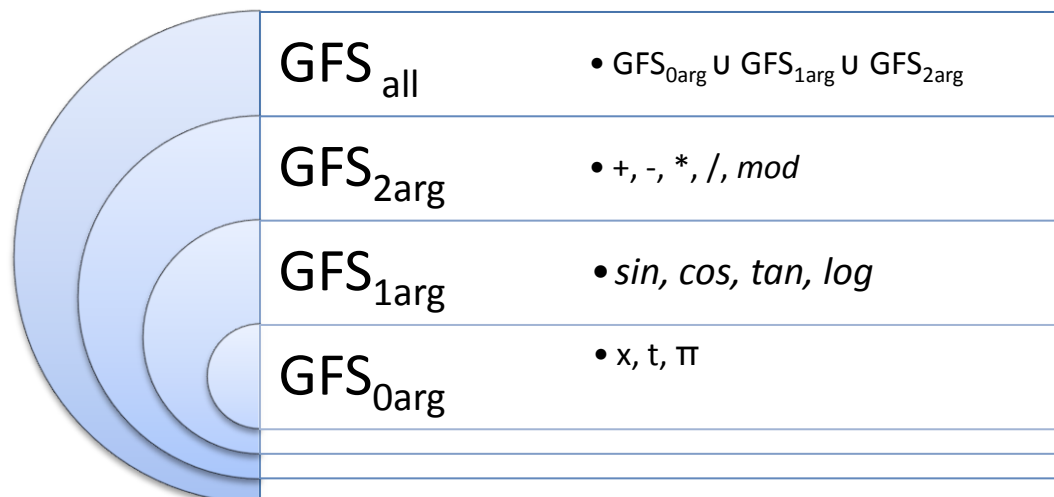
Analytické programování, dále jen AP, vzniklo pod záštitou fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně jako experimentální metoda, kterou lze chápat jako alternativní přístup vzhledem ke GP a GE. AP je založeno na práci s celočíselnými a diskrétními hodnotami. AP není samostatný evoluční algoritmus nýbrž pouze transformace neboli zobrazení z jedné množiny základních symbolů do konečné množiny všech možných kombinací či programů, které lze z těchto symbolů vytvořit.

Faktem je, že taková konečná množina bude zcela jistě obsahovat i programy, které lze označit za patologické², tzn. takové programy, které jsou principiálně nefunkční. Abychom taková řešení bezpečně vyřadily z procesu evoluční syntézy, musíme vhodně definovat a zkonstruovat účelovou funkci. (1)

AP tedy pracuje s množinou funkcí, operátorů a terminálních hodnot stejně tak, jako genetické programování (GP) nebo gramatická evoluce (GE) :

- funkce: *sin, cos, tan, And, Or*
- operátory: +, -, *, /, dt, ...
- terminály: 1,55; p, q, ...

Tyto objekty se v AP setřídí podle počtu argumentů, jenž je rozhodující faktor pro rozhodnutí, který z objektů bude v danou chvíli použit. Po setřídění objektů podle počtu argumentů dostaneme množinu, označme ji GFS, která je tvořena hierarchicky uspořádanými podmnožinami, označme je jako GFS_{0arg} , GFS_{1arg} , ..., GFS_{n-arg} , které obsahují funkce se stejným počtem argumentů. Souhrn těchto množin, tedy množinu, která obsahuje všechny objekty označme jako GFS_{all} . Symbolicky znázorněnou strukturu množiny GFS popisuje Obrázek 2.1 Symbolicky znázorněná struktura množiny GFS dle [1], str. 272

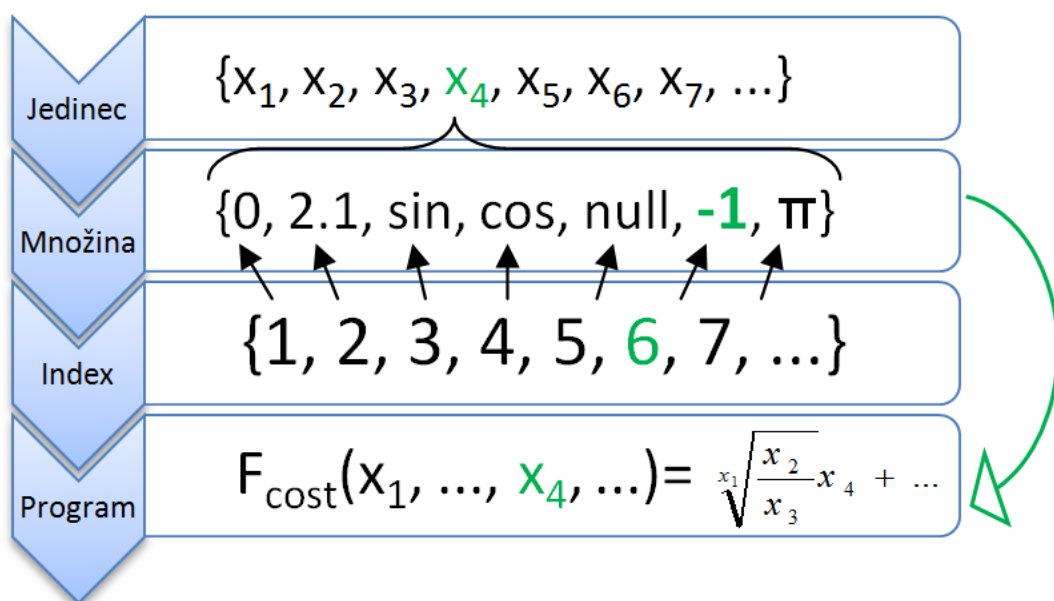


Obrázek 2.1 Symbolicky znázorněná struktura množiny GFS dle [1], str. 272

² Patologická funkce je matematicky neřešitelná. Mezi patologické funkce lze například zařadit funkci, kde dochází k dělení nulou, chybí argument goniometrické funkce, atd.

Evoluční syntéza poté probíhá zpočátku v GFS_{all} a postupně se zaměřuje na objekty s nižším počtem argumentů, tedy na objekty z nižších oblastí funkcí. V závislosti na počtu volných indexů v celočíselném jedinci pak vybírá nejvhodnější objekt. GFS_{all} je tedy sjednocením všech GFS_{n-arg} .

Samotná činnost AP je v podstatě naprosto triviální. Každý jedinec *žijící* v populaci je složen z celočíselných parametrů neboli ukazatelů *pointer*, které mají indexovanou svoji konkrétní hodnotu. Princip práce s diskretní množinou je demonstrován na Obrázek 2.2 Princip práce s diskretní množinou dle [1], str. 273. (1)



Obrázek 2.2 Princip práce s diskretní množinou dle [1], str. 273

Každý jedinec má x_1 až x_n parametrů. Parametry jsou omezeny spodní i horní hranicí, která by měla být ideálně v rozmezí $\langle 1, N \rangle$, kde N je maximální počet všech elementů GFS.

Na obrázku 1.6 je znázorněn jedinec, diskretní množina, celočíselný index a účelová funkce. Transformace celočíselných hodnot do výsledného funkcionálu je zde demonstrována pro čtvrtý parametr daného jedince. V tomto případě nabývá parametr x_4 hodnotu 6. Do výsledné účelové funkce je pak dosazena hodnota -1. (1)

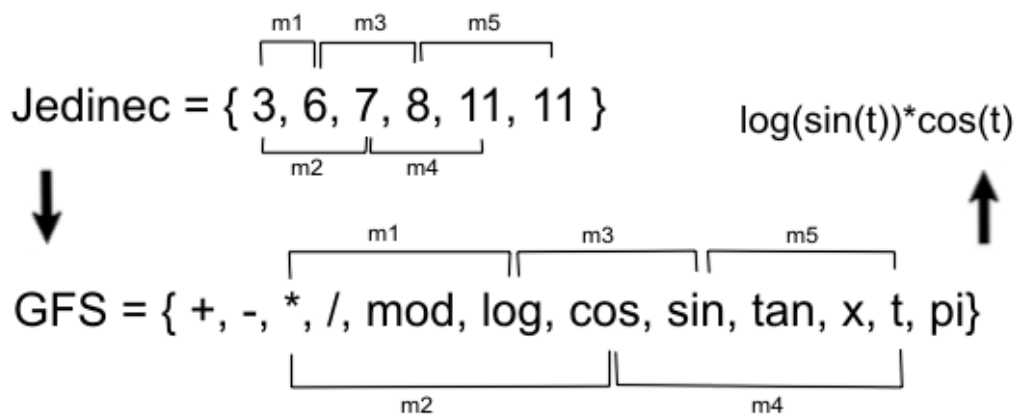
2.2.2 Programová syntéza

Programová syntéza probíhá v AP pomocí sekvence matematických operací, které vybraného jedince transformují na použitelný *program* (funkci). Jedná se o transformaci z prostoru jedinců do prostoru funkcí. Proces transformace sestává ze dvou částí –

z transformace celočíselných hodnot a bezpečnostních procedur, které zamezují algoritmu vytvářet patologické jedince potažmo *programy*, funkce. (4)

Princip DSH spočívá v tom, že je vytvořen vektor obsahující celá čísla, která lze chápat jako ukazatele na jednotlivé symbolické struktury či prvky obsažené v množině GFS_{all} . EA tedy pracuje s jedincem jako s vektorem diskretních hodnot. V případě potřeby ohodnotit jedince, určit hodnotu jeho účelové funkce, se provede transformace na konkrétní *program*, funkci, jejíž funkční hodnota vypovídá právě o vhodnosti daného jedince.

Lze tedy říci, že AP je sérií transformačních funkcí, kde platí určitá pravidla, která znemožní vznik patologických a defektivních jedinců – funkcí. Příklad běhu AP a transformace jedince názorně ukazuje



Obrázek 2.3 Transformace diskretního jedince na funkci pomocí AP dle (1), str. 273

Jak již bylo zmíněno, jedinec je reprezentován vektorem celých čísel. Analytické programování je pak sérií mapovacích funkcí:

1. První parametr jedince má hodnotu 1 – v tomto případě operátor +
 - a. operátor + je v podstatě funkce očekávající dva argumenty a je tedy nutné tyto dva argumenty zajistit. Jejich pozici ukazují další dva indexy jedince s hodnotami 6 a 7.
 - b. popisek **m1** (*m - mapping*) ukazuje na první argument operátoru + a popisek **m2**. V tomto případě se jedná o funkce *log* a *cos*.
 - c. Během tohoto kroku jsme syntetizovali výraz $\log(\) + \cos(\)$, což na první pohled nejde spočítat.

2. Nyní najdeme chybějící argumenty těchto funkcí
 - a. ukazateli na tyto argumenty jsou popisky *m3* a *m4*.
 - b. nyní máme syntetizován výraz $\log(\sin(\quad)) + \cos(t)$, což ovšem není opět řešitelné a je nutné další hledání argumentu.
3. Nalezneme poslední chybějící argument
 - a. ten se skrývá pod jedincovým indexem č. 11, tedy jeho mapování zobrazuje popisek *m5*.
4. Nyní získáváme celistvý uzavřený výraz – ten je již řešitelný.
 - a. celý proces AP končí

Po použití analytického programování je tedy symbolickým ekvivalentem vektoru jedince $J = \{3,6,7,8,11,11\}$ výraz $\log(\sin(t)) \cdot \cos(t)$. V případě, že by bylo pořadí prvků množiny GFS odlišné od uvedeného příkladu, došlo by samozřejmě k syntetizaci naprosto jiného výrazu – takový výraz by mohl být samozřejmě i patologický. (1)

2.2.3 Posílené hledání

Posílené hledání (*reinforced search*) je technika, která byla do AP implementována později po provedení několika testů. Prvotní AP vykazovalo neuspokojivé výsledky v porovnání s např. GP. Hlavní myšlenka spočívá v tom, že se do základní množiny symbolických objektů GFS přidá aktuálně syntetizovaný program (jeho část), který vyhovuje uživatelem stanovené hranici. Obohacení GFS o novou informaci ve formě alespoň částečně vyhovujícího programu způsobilo, že AP podává srovnatelný a mnohdy i větší výkon, než GP. Pokud je během hledání nalezen program, jehož vhodnost je lepší než vhodnost již zavedeného programu, je starý program nahrazen novým. To má za následek jednu velkou výhodu – kardinalita množiny GFS se nemění, zůstává stejná, ale zároveň se zvyšuje její kvalita. (1)

2.2.4 Bezpečnostní procedury

AP obsahuje několik bezpečnostních procedur podobně jako GP. Implementace těchto bezpečnostních procedur je až na hladině účelové funkce, kde se před vlastním ohodnocením provádí kontrola (případná náprava) toho, zda syntetizovaný program :

- neobsahuje komplexní či reálné části (pokud to není žádoucí)
- neobsahuje limitní funkce (nekonečna, dělení nulou)
- je syntetizován za dobu, která je pro uživatele únosná

Tyto procedury nejsou nezbytnou součástí AP (standardně je AP nemá). Lze je považovat za část účelové funkce – a tedy záleží na uživateli, jaké bezpečnostní procedury si zvolí.

(1)

2.2.5 Podobnosti a rozdíly mezi již existujícími metodami

- AP, GP i GE umí syntetizovat požadované programy
- Množina terminálů a funkcí je stejná pro AP, GP, i GE
- AP_{meta} a AP_{nf} používají konstantu K , která je dostatečně indexována
- AP pracuje s celočíselnými jedinci (metoda DSH), GP používá ve své kanonické verzi přímou reprezentaci v jazyce *Lisp* a GE používá binární reprezentaci následně konvertovanou do celočíselné reprezentace
- GP používá náhodně vygenerované konstanty, GE používá uživatelsky definované konstanty a AP používá pouze jednu obecnou konstantu K , která je podle pořadí výskytu indexována, a takto již rozdílné konstanty jsou vhodnou technikou odhadnuty.
- Patologické programy jsou v AP penalizovány – vyloučeny. U GP je v případě syntézy patologického programu syntéza opakována. U GE je, dle tvůrců, vznik patologických programů vyloučen už ze samotného principu.

2.2.6 Funkce nelineárního prokládání

Zdrojový kód AP byl napsán v prostředí Mathematica 7. Je to vysoce výkonné a efektivní softwarové prostředí, které vyvíjí společnost Wolfram Researches. Zdrojový kód, se kterým jsem během této práce pracoval, je na tomto softwaru kompletně závislý a bez jeho přítomnosti nelze zdrojový kód zkompileovat.



Obrázek 2.4 Logo matematického softwaru Mathematica 7 for Students

Díky tomuto software bylo do AP možné implementovat pokročilé programátorské a matematické techniky, jelikož jsou jeho součástí. Velmi výhodná se ukázala být metoda nelineárního prokládání (*non-linear fitting, non-linear regress*). Použití této metody je velmi jednoduché a z programátorského hlediska výhodné, protože vyžaduje minimální zásah od uživatele. Metoda vyžaduje pouze definici matematické funkce s příslušnými konstantami a data, kterými chceme tuto funkci proložit.

3 SYMBOLICKÁ INTEGRACE

Pomocí tzv. symbolické integrace lze nalézt matematický výraz v symbolickém tvaru ³, který bude integrálem dané křivky. Tzv. LEX systém, který vyvinuli pánové Mitchell, Utgoff a Banerji v roce 1983, patří mezi neznámější postupy jak řešit symbolickou integraci. Výsledkem symbolické integrace, která je v této práci realizována spojením analytického programování a symbolické regrese, je funkce v symbolickém tvaru, vyjádřená pomocí matematického výrazu. Výsledná funkce pak může být perfektním řešením daného problému, nebo se ke správnému řešení (integrálu) alespoň aproximativně přiblíží. Daná křivka může být reprezentována buď jako matematický výraz v symbolickém tvaru nebo jako diskrétní množina hodnot v případě, že tato křivka nebyla matematicky explicitně vyjádřena. Pokud je tedy křivka reprezentována matematickým výrazem, nejprve ji převedeme na konečnou množinu vzorků (bodů). Toho docílíme tak, že vezmeme dostatečný počet náhodných hodnot nezávislé proměnné $\{x_i\}$, která se nachází v daném matematickém výrazu, na určitém intervalu či požadované oblasti. Poté utvoříme dvojice – pro každou hodnotu nezávislé proměnné x_i najdeme funkční hodnotu y_i dané křivky (12).

Proces symbolické integrace začneme s konečnou množinou číselných dvojic $\{x_i, y_i\}$.

Příkladem, řekněme, že máme 50 číselných dvojic $\{x_i, y_i\}$ pro i 0 až 49. Budeme předpokládat, že hodnoty x_i máme seřazeny dle velikosti tak, že platí: $x_i < x_{i+1}$. Hodnoty x_i leží na vybraném intervalu (12).

Hlavním cílem tohoto příkladu je nalézt matematický výraz, který bude přijatelným či naprosto přesným způsobem vyjadřovat integrál dané křivky v symbolickém tvaru za pomoci pouze 50 známých bodů (hodnot) $\{x_i, y_i\}$ (12).

Mějme výchozí funkci:

$$\text{Cos}(x + 2x + 1)$$

³ Matematickým výrazem v symbolickém tvaru myslíme libovolný výraz, který je popsán pomocí matematických funkcí či symbolů.

Cílem je najít integrál v symbolickém tvaru:

$$\text{Sin}(x + x^2 + 1)$$

V tomto případě budeme uvažovat interval $(0, 2\pi)$

Symbolická integrace je v podstatě pouze symbolická regrese s přidaným krokem numerické integrace. Konkrétněji, numericky integrujeme křivku, která je definována konečnou množinou 50 bodů $\{x_i, y_i\}$ přes určitý interval (x_0, x_i) . Integrál $I(x_i)$ je funkcí proměnné x_i (12).

V prvním bodě je hodnota integrálu rovna nule, $I(x_0) = 0$. Pro každý další bod x_i , kde i je mezi 1 a 49 provedeme numerickou integraci sečtením oblastí všech lichoběžníků ležících mezi bodem x_0 a bodem x_i . Získáme tak aproximovanou hodnotu integrálu $I(x_i)$ dané křivky pro každý bod x_i . Dostáváme tedy nový vektor obsahující 50 párových číselných hodnot $\{x_i, I(x_i)\}$ pro i mezi 0 a 49, které jsou kandidáty na vhodné řešení tohoto problému. Nyní použijeme symbolickou regresi pro nalezení matematického výrazu pro křivku, která je definována právě body $\{x_i, I(x_i)\}$. Tento matematický výraz je integrálem původní křivky definované body $\{x_i, y_i\}$ v symbolickém tvaru (12).

Celý proces popisuje Tabulka 3.1 Hledání integrálu v symbolické formě.

Tabulka 3.1

Tabulka 3.1 Hledání integrálu v symbolické formě dle (12), str. 259.

1	x_i	0.00	1.57	3.14	4.71	6.28
2	$y = \text{Cos}(x_i + 2x_i + 1)$	2.00	4.14	6.28	10.42	14.57
3	$\int_{x=0}^{x_i} \text{Cos}(x + 2x + 1) dx$	0.00	4.82	13.01	26.13	45.76
4	$\text{Sin}(x + x^2 + x)$	0.00	5.04	13.01	25.92	45.76
5	Absolutní chyba	0.00	0.21	1.78	0.21	0.00

Řádek 1 obsahuje pět hodnot pro x_i rovnoměrně rozdělených na intervalu $(0, 2\pi)$ - a to $0\pi, \frac{1}{2}\pi, \pi, \frac{3}{2}\pi, 2\pi$. Řádek 2 obsahuje hodnoty pro každé z pěti x_i z řádku 1 – Tyto hodnoty odpovídají hodnotě křivky $y = \text{Cos}(x_i + 2x_i + 1)$. Řádek 3 obsahuje hodnoty integrálu křivky $y = \text{Cos}(x_i + 2x_i + 1)$ od počátku intervalu (0) po x_i . Numerická integrace byla počítána sčítáním ploch lichoběžníků ležících pod neznámou křivkou, kterou udává řádek 2. Proces symbolické regrese byl poté aplikován na řádek 1 a 3. Konkrétně, řádek 1 je považován za nezávislou proměnnou neznámé funkce, zatímco řádek 3 je chápán jako hodnota závislé proměnné. Po proběhnutí několika generací analytické programování vyprodukuje výraz v symbolickém tvaru $\text{Sin}(x + x^2 + x)$, který je integrálem k neznámé křivce. Řádek 4 ukazuje na hodnoty integrálu pro každé z pěti x_i . Řádek 5 zobrazuje absolutní chybu, která vzniká během procesu aproximace. Je to rozdíl mezi řádkem 3 a 4. Vzhledem k tomu, že absolutní chyba je relativně malá pro každé x_i , můžeme výraz $\text{Sin}(x + x^2 + x)$ považovat za integrál neznámé křivky (12).

Množina funkcí, ze kterých bude algoritmus analytického programování skládat matematický výraz reprezentující integrál dané křivky musí obsahovat funkce, které budou pro takové vyjádření dostačující – funkce, které budou potřeba samozřejmě dopředu neznáme. V takové situaci musíme množinu funkcí volit s rozvahou. Pravděpodobně je lepší zahrnout do této množiny i několik cizích funkcí, než opomenout potřebnou funkci. Pokud potřebujeme funkci, která v této množině chybí, analytické programování pomocí symbolické regrese tuto funkci aproximuje co nejpřesněji funkcí jinou, dostupnou (12).

Přiměřeně zvolená množina funkcí, která je pro tento problém dostačující je následující:

$$F = \{+, -, *, /, \text{Sin}, \text{Cos}, \text{Exp}, \text{Log}\}$$

4 SYMBOLICKÁ DERIVACE

Symbolická derivace je analogií k již výše zmíněné symbolické integraci.

Proces symbolické integrace začneme s konečnou množinou číselných dvojic $\{x_i, y_i\}$.

Příkladem, řekněme, že máme 50 číselných dvojic $\{x_i, y_i\}$ pro i 0 až 49. Budeme předpokládat, že hodnoty x_i máme seřazeny dle velikosti tak, že platí: $x_i < x_{i+1}$. Hodnoty x_i leží na vybraném intervalu (12).

Hlavním cílem tohoto příkladu je nalézt matematický výraz, který bude přijatelným či naprosto přesným způsobem vyjadřovat derivaci dané křivky v symbolickém tvaru za pomoci pouze 50 známých bodů (hodnot) $\{x_i, y_i\}$.

Mějme opět stejnou výchozí funkci:

$$\text{Cos}(x + 2x + 1)$$

Cílem je najít derivaci v symbolickém tvaru:

$$-3\text{Sin}(3x + 1)$$

V tomto případě budeme uvažovat opět interval $(0, 2\pi)$

Symbolická derivace je v podstatě taktéž symbolická regrese s přidaným krokem numerické derivace. Konkrétněji, numericky derivujeme křivku, která je definována konečnou množinou 50 bodů $\{x_i, y_i\}$ přes určitý interval (x_0, x_i) . Derivace $D(x_i)$ je funkcí proměnné x_i .

Pro každý bod x_i , kromě počátečního bodu x_0 a koncového bodu x_{49} , je derivace rovna průměru sklonu křivky mezi body x_i a x_{i-1} a sklonu křivky mezi body x_i a x_{i+1} . Pro dva hraniční body je derivace rovna právě sklonu křivky. Získáme tak aproximovanou hodnotu derivace $D(x_i)$ dané křivky pro každý bod x_i . Dostáváme tedy nový vektor obsahující 50 párových číselných hodnot $\{x_i, D(x_i)\}$ pro i mezi 0 a 49, které jsou kandidáty na vhodné řešení tohoto problému. Nyní použijeme symbolickou regresi pro nalezení matematického výrazu pro křivku, která je definována právě body $\{x_i, D(x_i)\}$. Tento matematický výraz je derivací původní křivky definované body $\{x_i, y_i\}$ v symbolickém tvaru (12).

II. PRAKTICKÁ ČÁST

5 NÁVRH POSTUPU ŘEŠENÍ

Nejdříve bylo nutné naprogramovat jednotlivé příklady pro integrování a derivování. K tomu byl použit software *Wolfram Mathematica* a nástroj symbolické regrese - algoritmus analytické programování s využitím algoritmů DE a SOMA. Poté je základním postupem metoda opakovaného *měření* – tedy, nastavení vstupních parametrů algoritmu, spuštění algoritmu a získání výsledků. Aby bylo možné odhadnout a nastavit parametry a dostatečný počet opakování tak, aby byly získané výsledky uspokojivé, bylo nutné nejdříve oba algoritmy (*DE*, *SOMA*) otestovat a zjistit tak alespoň zhruba jejich povahu. V konečné fázi jsou naměřená data zaznamenána a vizualizovaná pro výukové účely pomocí vhodných funkcí programu *Mathematica*. Výsledky byly umístěny do příloh této diplomové práce a to vzhledem k velkému objemu dat a množství tabulek.

5.1 Algoritmy symbolické integrace a derivace

Programování jednotlivých příkladů na integraci a derivaci bylo provedeno v prostředí *Mathematica*. Byly vytvořeny čtyři programy (*algoritmy*) – první dva na derivaci a integraci s využitím evolučního algoritmu DE a další dva na integraci a derivaci s využitím SOMA.

Programy jsou schopny velmi přesně aproximovat pomocí křivky danou množinu bodů, která je buď integrálem, nebo derivací výchozí vstupní množiny bodů, a tuto vzniklou křivku také matematicky vyjádřit v symbolickém tvaru.

Popis jednotlivých programů (*zdrojových souborů*) je uveden v Příloze PV.

5.2 Množina GFS

Množina GFS obsahuje prvky uvedené v následující tabulce:

Tabulka 5.1 Použitá množina GFS

Typ funkce (prvku)	Výčet funkcí
Základní aritmetika	plus(2), subtract(2), minus(1), times(2), divide(2)
Goniometrické funkce	cos(1), sin(1)
Exponenciální funkce	log(1)
Elementární funkce	power(2)
Proměnná	x(0)

Čísla v závorkách vyjadřují počet argumentů dané funkce.

5.3 Testování DE

Testování DE bylo provedeno na šesti odlišných funkcích (viz Tabulka 5.2 Testovací funkce pro DE). Během testů bylo zjištěno, že algoritmus je poměrně hodně citliví na nastavení parametrů F a Cr . Rychlost DE se při testech na vybraných funkcích ukázala v být v porovnání se SOMA dosti větší. Výsledky byly u všech funkcí uspokojivé. Vzhledem k citlivosti tohoto algoritmu byl zvolen následující postup:

1. Nastavení parametrů, provedení a zaznamenání 10 cyklů evoluce a výběr dvou nejlepších řešení.
2. Přenastavení parametrů, provedení a zaznamenání 5 cyklů evoluce a výběr dvou nejlepších řešení
3. Přenastavení parametrů, provedení a zaznamenání 3 cyklů evoluce a výběr dvou nejlepších řešení

Během těchto kroků se podařilo nastavit algoritmus tak, aby podával uspokojivé výsledky téměř při každém spuštění.

Tabulka 5.2 Testovací funkce pro DE

$f(x)$	$f(x)$
1 $\text{Cos}[x] + \text{Sin}[x]$	4 $x \text{Sin}[x^2]$
2 $\text{Log}[x]$	5 $x^3 \text{Cos}[x]$
3 $x^3 E^{-x^2}$	6 $(\text{Log}[x] \text{Sin}[x])/\text{Log}[2]$

5.3.1 Proces symbolické integrace pomocí DE

1. $\cos(x) + \sin(x)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PI: Tabulka 5.3.1 Symbolická integrace – funkce 1. - DE

2. $\log(x)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PI: Tabulka 5.3.2 Symbolická integrace – funkce 2. - DE

3. $x^3 e^{-x^2}$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PI: Tabulka 5.3.3 Symbolická integrace – funkce 3. - DE

4. $x \sin(x^2)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PI: Tabulka 5.3.4 Symbolická integrace – funkce 4. - DE

5.3.2 Proces symbolické derivace pomocí DE

1. $\cos(x) + \sin(x)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PII: Tabulka 5.3.5 Symbolická derivace – funkce 1. - DE

2. $\log(x)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PII: Tabulka 5.3.6 Symbolická derivace – funkce 2. - DE

3. $x^3 \cos(x)$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PII: Tabulka 5.3.7 Symbolická derivace – funkce 5. – DE

4. $\frac{\log(x)\sin(x)}{\log(2)}$

- a. První nastavení parametrů - výsledky nedostatečné
- b. Druhé nastavení parametrů - zlepšení kvality řešení
- c. Třetí nastavení parametrů - řešení je přijatelné

Příloha PII: Tabulka 5.3.8 Symbolická derivace – funkce 6. – DE

5.4 Testování SOMA

Testování SOMA bylo prováděno na pěti odlišných funkcích (viz Tabulka 5.3 Testovací funkce pro SOMA). Během testů bylo zjištěno, že algoritmus je v porovnání s DE robustnější. Pro uspokojivé výsledky stačilo většinou stejné (velmi podobné) nastavení parametrů pro všechny funkce. Ovšem rychlost SOMA se při testech na vybraných funkcích ukázala být, v porovnání s DE, značně malá. Výsledky byly u všech funkcí taktéž uspokojivé. Vzhledem k nižší citlivosti a větší robustnosti tohoto algoritmu byl zvolen následující postup:

1. Nastavení parametrů, provedení a zaznamenání 10 cyklů evoluce a výběr dvou nejlepších řešení.

Během jediného kroku se podařilo nastavit algoritmus tak, aby podával uspokojivé výsledky téměř při každém spuštění.

Tabulka 5.3 Testovací funkce pro SOMA

	f(x)
1	$\text{Cos}[x] + \text{Sin}[x]$
2	$x \text{Cos}[x]$
3	$\text{Log}[x]$
4	$(0.05 + 0.25 x^2) \text{Sin}[x]$
5	$\text{Cos}[x]/2 + 3 \text{Sin}[x]$

5.4.1 Proces symbolické integrace pomocí SOMA

1. $\text{Cos}(x) + \text{Sin}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIII: Tabulka 5.4.1 Symbolická integrace – funkce 1. - SOMA

2. $x\text{Cos}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIII: Tabulka 5.4.2 Symbolická integrace – funkce 2. - SOMA

3. $\text{Log}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIII: Tabulka 5.4.3 Symbolická integrace – funkce 3. – SOMA

4. $(0.25^2 + 0.05)\text{Sin}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIII: Tabulka 5.4.4 Symbolická integrace – funkce 4. - SOMA

5.4.2 Proces symbolické derivace pomocí SOMA

1. $\frac{1}{2}\text{Cos}(x) + 3\text{Sin}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIV: Tabulka 5.4.5 Symbolická derivace – funkce 5. - SOMA

2. $x\text{Cos}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIV: Tabulka 5.4.6 Symbolická derivace – funkce 2. – SOMA

3. $\text{Log}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIV: Tabulka 5.4.7 Symbolická derivace – funkce 3. – SOMA

4. $(0.25^2 + 0.05)\text{Sin}(x)$

- a. První nastavení parametrů - řešení je přijatelné

Příloha PIV: Tabulka 5.4.8 Symbolická derivace – funkce 4. - SOMA

5.5 Integrace a derivace klasickými metodami

Operace integrování (tj. operace určování primitivní funkce) a derivování jsou navzájem inverzní. Mezi klasické postupy integrování a derivování patří vyhledávání známých integrálů (*derivací*) v tabulkách. Existují rozsáhlé tabulky, ve kterých lze nalézt množství neurčitých integrálů či derivací. K výsledkům můžeme dospět použitím následujících pravidel a metod integrace a derivace:

- Tabulkové integrály
- Integrace Per Partes (po částech)
- Substituční metoda
- Integrace racionálních funkcí
- Integrace metodou derivování podle parametru
- Racionalizace integrálů
- Integrace transcendentních funkcí

Dnes však tyto tabulky a metody ztrácejí význam, neboť jsou dostupné matematické programy, které zvládnou integraci a derivaci složitých funkcí. Dokonce i na Internetu lze nalézt řadu online kalkulačtorů. Po zadání integrované či derivované funkce je nalezena odpovídající funkce.

$$\begin{array}{ll}
 \int x^\alpha dx = \frac{x^{\alpha+1}}{\alpha+1} + C, & x > 0; \quad \text{pro } \alpha \neq -1 \\
 \int \frac{1}{x} dx = \ln|x| + C, & x \neq 0 \\
 \int \sin(x) dx = -\cos(x) + C & \int \frac{1}{\cos^2(x)} dx = \operatorname{tg}(x) + C, \quad x \neq \frac{\pi}{2} + k\pi \\
 \int \cos(x) dx = \sin(x) + C & \int \frac{1}{\sin^2(x)} dx = -\operatorname{cotg}(x) + C, \quad x \neq k\pi \\
 \int \sinh(x) dx = \cosh(x) + C & \int \frac{1}{\cosh^2(x)} dx = \operatorname{tgh}(x) + C \\
 \int \cosh(x) dx = \sinh(x) + C & \int \frac{1}{\sinh^2(x)} dx = -\operatorname{cotgh}(x) + C, \quad x \neq 0 \\
 \int \frac{1}{1+x^2} dx = \operatorname{arctg}(x) + C & \int \frac{1}{\sqrt{1-x^2}} dx = \operatorname{arcsin}(x) + C, \quad x \in (-1, 1)
 \end{array}$$

Obrázek 5.1 Přehled nejzákladnějších tabulkových integrálů

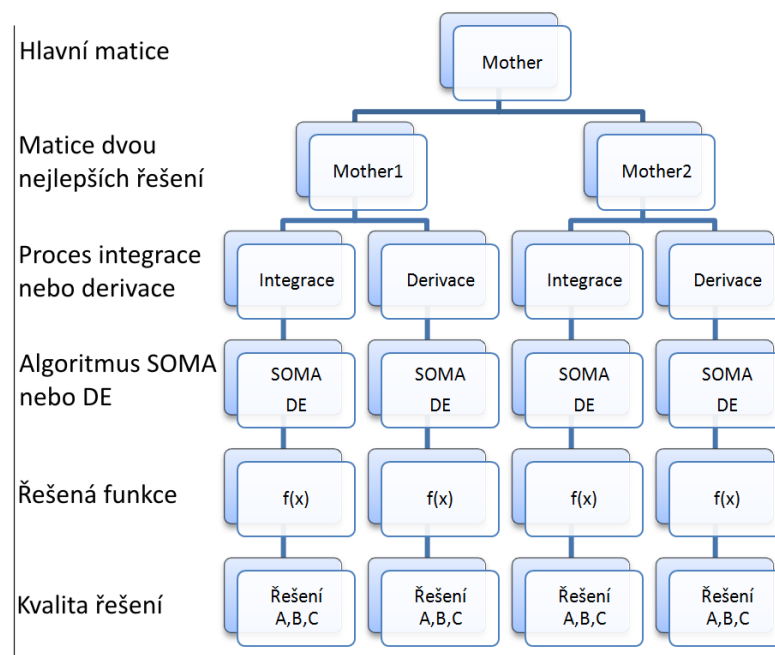
5.6 Vizualizace pro výukové účely

Hlavním záměrem této práce bylo vhodným způsobem vizualizovat průběh evolučních algoritmů DE a SOMA, které byly využity v AP, během procesu integrování a derivování. Pro ten to účel jsem zvolil funkci *Manipulate*, která je standardní vizualizační funkcí programu *Mathematica*.

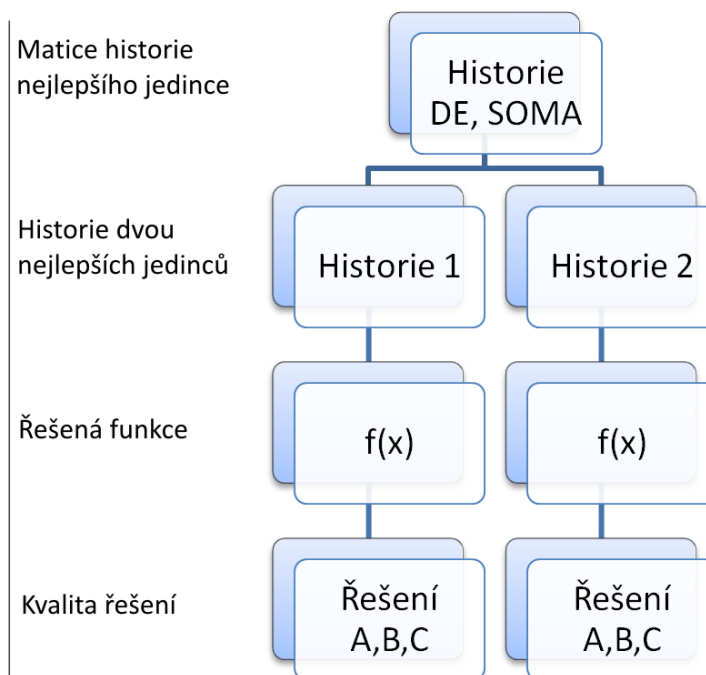
Funkce *Manipulate* generuje jednotlivé části (*verze*) výrazu *expr* a s pomocí přidáných ovládacích prvků umožní interaktivní ovládání hodnoty *u*.

`Manipulate[expr, {u, umin, umax}]`

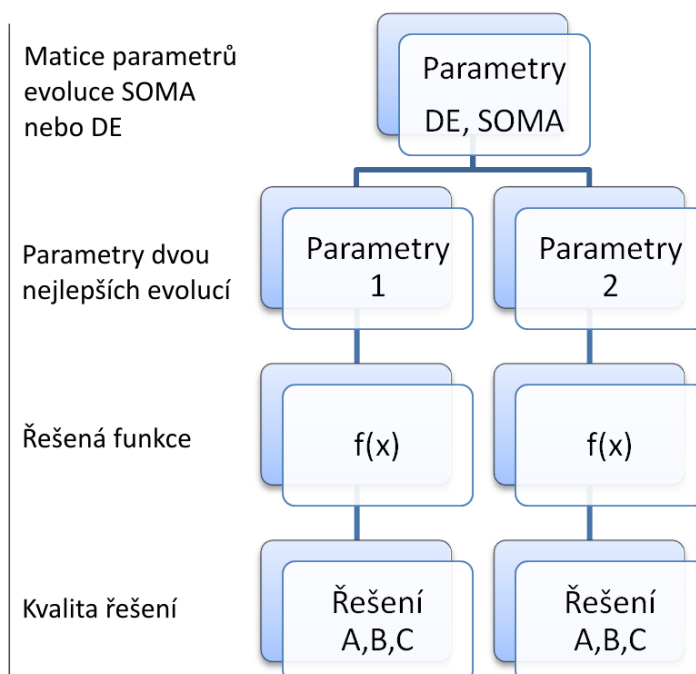
Abychom mohli vizualizovat velké množství dat, musíme je nejprve uložit do *databáze* s vhodně navrženou strukturou. První myšlenka spočívala v tom, že všechna data budou uložena do jediné n-rozměrné matice a poté dle potřeby vyhledána a zobrazena. Během procesu zaznamenávání, ukládání a testování dat a jejich vizualizace se to ovšem ukázalo být jako nevhodné řešení. Časová náročnost na vyhledání konkrétní položky neúprosně rostla a díky tomu kvalita vizualizace klesala. Proto bylo od tohoto řešení upuštěno a matice byla rozdělena na více částí (*matic*). Strukturu navržených matic popisují následující obrázky:



Obrázek 5.2 Struktura hlavní matice mother



Obrázek 5.3 Struktura matice historie průběhu nejlepšího jedince

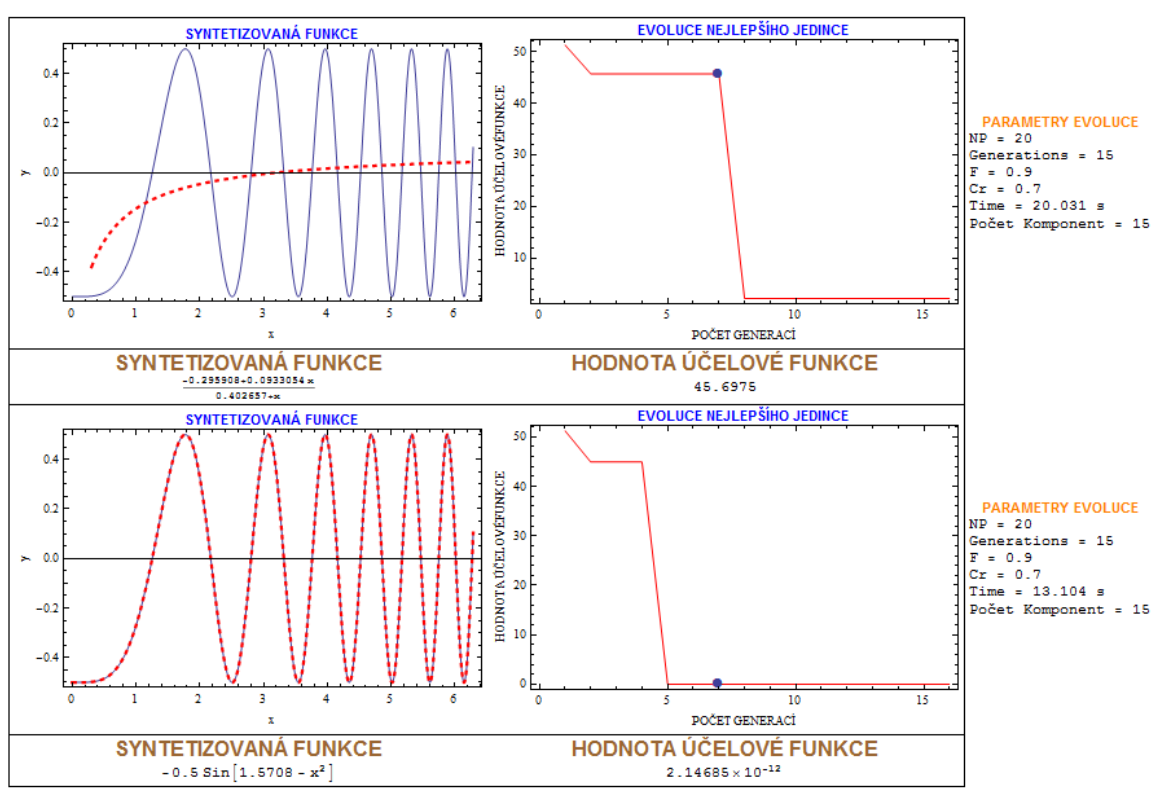


Obrázek 5.4 Struktura matice parametrů evoluce

Tyto tři základní matice byly naplněny vzorovými daty a pomocí funkce *Manipulate* byla provedena ukázková interaktivní vizualizace, která uživateli nabízí tyto možnosti:

- Výběr procesu integrování nebo derivování
- Výběr algoritmu SOMA nebo DE⁴
- Výběr řešené funkce
- Výběr kvality řešení⁵
- Možnost zobrazení libovolného kroku proběhnuté evoluce

Jak výsledná vizualizace vypadá, můžeme vidět na následujících obrázcích:



Obrázek 5.5 Náhled vytvořeného vizualizačního interaktivního prostředí

⁴ Výběr algoritmu SOMA nebo DE je řešen pomocí dvou zdrojových souborů, z nichž každý obsahuje právě algoritmus SOMA nebo DE. Toto řešení bylo zvoleno z důvodu již výše zmíněné časové náročnosti vizualizace, která se projevuje velkými časovými prodlevami mezi uživatelským zásahem a zobrazením výsledku.

⁵ Výběr kvality řešení je umožněn pouze u algoritmu DE. U algoritmu SOMA je zobrazováno řešení jedině.

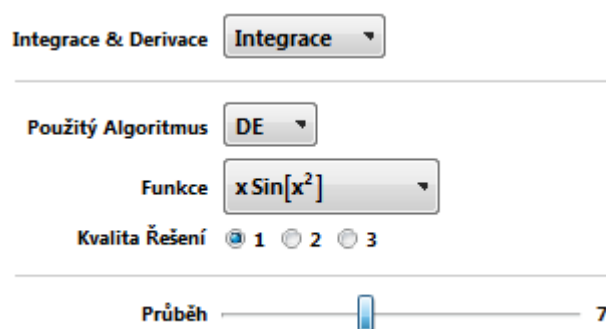
5.7 Ovládání aplikace

Ovládání aplikace je velmi jednoduché. Uživatel má k dispozici několik řídicích prvků s možností výběru. Tyto prvky popisuje Tabulka 5.4 Popis ovládacích prvků aplikace.

Tabulka 5.4 Popis ovládacích prvků aplikace

Pole výběru	Popis pole
Integrace & Derivace	Zde uživatel volí proces integrování nebo derivování
Použitý Algoritmus	Vzhledem k datové náročnosti vizualizace bylo od možnosti výběru algoritmu přímo v aplikaci upuštěno. Algoritmus je v aplikaci pevně daný. ⁶
Funkce	Menu s možností výběru funkce.
Kvalita Řešení	Checkboxy s možností voby kvality řešení. ⁷
Průběh	Slider s možností procházení průběhu evoluce.

Jak ovládací panel vypadá, ukazuje Obrázek 5.6 Ovládací panel vizualizačního interaktivního prostředí



Obrázek 5.6 Ovládací panel vizualizačního interaktivního prostředí

⁶ Výběr algoritmu je možný pouze tak, že uživatel spustí zvlášť zdrojový kód vizualizace pro DE nebo pro SOMA. Každý je uložen v jiném zdrojovém souboru, viz Příloha PV: Adresářová struktura příloženého CD.

⁷ Výběr kvality řešení je možný pouze u algoritmu DE.

ZÁVĚR

Hlavní zásady, které jsou uvedené v zadání práce, jsem splnil. Uvedl jsem základní výčet evolučních algoritmů, se zaměřením na algoritmy SOMA a DE, a evolučních výpočetních technik a popsal s nimi související nástroj evoluční syntézy struktur nazvaný analytické programování.

Analytické programování, v kombinaci s evolučními algoritmy SOMA a DE, se ukázalo být velmi silným nástrojem určeným k syntéze symbolických struktur. Během testování těchto algoritmů jsem zjistil, že jsou oba schopny precizně pracovat a oba dokážou poskytnout přesné výsledky. Kvalita řešení a jeho časová náročnost je přímo závislá na nastavení vstupních parametrů těchto algoritmů. Obecně lze říci, že čím více jsme seznámeni s definovaným problémem libovolné optimalizace, tím lépe dokážeme vstupní parametry algoritmu nastavit tak, aby pracoval optimálně, tedy aby byly časy výpočtů a jejich náročnost minimální.

Programování symbolického integrování a derivování probíhalo v programu *Mathematica*. Samotný proces integrace a derivace pomocí AP a EA probíhal v několika různých nastaveních vstupních parametrů. Nejprve byl algoritmus nastaven a spuštěn v podstatě naslepo a výsledky byly analyzovány a zaznamenány. Poté, dle kvality prvního nastavení, byly parametry mírně upraveny a opět proběhla analýza a záznam výsledků. Třetí nastavení bylo opět analyzováno a zaznamenáno. Po třetím nastavení vstupních parametrů algoritmy poskytovaly velmi přesné výsledky. Tento postup byl zvolen z důvodu hlavního zaměření mé práce a to na využití během výuky umělé inteligence nebo matematiky.

Díky funkci *Manipulate*, která je nativní součástí programu *Mathematica* a vykreslovacím funkcím, jež byly pro vizualizaci výsledků evolucí použity, jsem dokázal vytvořit software, který názorně ukazuje jak se algoritmy SOMA a DE chovají, jak lze ovlivnit kvalitu výsledku vhodným nastavením parametrů a v neposlední řadě také to, že možnosti využití evolučních algoritmů jsou nekonečné.

Věřím, že výsledky mé práce budou pro budoucí studenty přínosem, a že s jejich pomocí dokážou lépe pochopit principy použití symbolické regrese, konkrétně evoluční syntézy symbolických struktur, ve spojení s analytickým programováním a evolučními algoritmy.

text

CONCLUSION

The guiding principles that are listed in the work assignment, I did. I've included a basic list of evolutionary algorithms, focusing on algorithms SOMA, DE and evolutionary computation, and described the related structures of evolutionary synthesis tool called analytical programming.

Analytic programming, combined with evolutionary algorithms and SOMA DE turned out to be a very powerful tool designed to synthesize symbolic structures. During testing of these algorithms, I found that they are both capable of precision work, and both can provide accurate results. The quality of solution and its time complexity is directly dependent on the setting of the input parameters of these algorithms. Generally speaking, the more we are familiar with any optimization problem defined, the better we can set the input parameters of the algorithm to work optimally, so that time calculations and their minimum demands.

Programming symbolic integration and differentiation was carried out in Mathematica. The process of integration and differentiation through AP and EA was conducted in several different settings of input parameters. First, the algorithm was set up and run essentially blind and the results were analyzed and saved. Then, according to the quality of the initial setup, the parameters were slightly modified and re-recording was analyzed and the saved. The third set was again analyzed and saved. After the third set of input parameters, algorithms give very accurate results. This procedure was chosen because the main focus of my work on the use of artificial intelligence in education or mathematics.

Thanks to Manipulate, which is a native part of the program Mathematica and rendering features that were to visualize the results of evolution are used, I was able to create software that graphically shows how the SOMA and DE algorithms behave, how you can affect the quality of the result by adjusting parameters and, ultimately, that the possibility of using evolutionary algorithms are endless.

I believe that the results of my work will benefit future students and that their use can better understand the principles of using symbolic regression, namely the evolutionary synthesis of symbolic structures in conjunction with analytical and programming algorithms.

SEZNAM CITOVANÉ LITERATURY

1. **Zelinka, Ivan, a další.** *Evoluční výpočetní techniky - principy a aplikace.* Praha : BEN - technická literatura, 2008. ISBN 978-80-7300-218-3.
2. Alan Turing. *Wikipedia.* [Online] [Citace: 7. Březen 2011.] [http://cs.wikipedia.org/wiki/Alan_Turing.](http://cs.wikipedia.org/wiki/Alan_Turing)
3. **I., ZELINKA.** *Umělá inteligence v problémech globální optimalizace.* Praha : BEN Technická literatura, 2002. ISBN 80-7300-069-5.
4. **Vařacha, Pavel.** *Syntéza neuronových sítí metodou symbolické regrese.* Zlín : Univerzita Tomáše Bati ve Zlíně, 2006. Diplomová Práce.
5. **Oplatková, Zuzana.** *Metaevolution - Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms.* Saarbrücken : Lambert-Publishing, 2009. ISBN: 978-8383-1808-0.
6. **Zelinka, Ivan.** *Analytic programming by Means of Soma Algorithm.* Brno : Mendel '02, 2002. stránky 90-102. ISBN 80-214-2135-5.
7. **Zelinka I., Oplatková Z.** *Analytic programming – Comparative Study.* Singapore : The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, 2003. ISSN 0219-6131.
8. **Zelinka, Ivan a Oplatková, Zuzana.** *Boolean Parity Function Synthesis by Means of Arbitrary Evolutionary Algorithms - Comparative Study.* Orlando : 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), 2004.
9. **ZELINKA I, OPLATKOVÁ Z., NOLLE L.** *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms - Comparative Study.* Magdeburg : 18th European Simulation Multiconference (ESM 2004), 2004. ISBN 3-936150-35-4.
10. **I., ZELINKA.** *Analytic programming by Means of Soma Algorithm.* ICICIS'02. Cairo : First International Conference on Intelligent Computing and Information Systems, 2002. ISBN 977-237-172-3.
11. **OPLATKOVÁ Z., ZELINKA I.** *Investigation on Artificial Ant using Analytic Programming.* Seattle : GECCO 2006, 8-12.7.2006, 11th conference.
12. **Koza, John R.** *Genetic Programming: on the programming of computers by means of natural selection.* Bradford : MIT Press, 1992. ISBN: 0-262-11170-5.

13. **O'Neill, M a Conor, R.** *Grammatical Evolution*. Norwell : Kluwer Academic Publishers, 2003. ISBN: 1-4020-7444-1.
14. **Banshaf, W.** *Genetic Programming and Evolvable Machines*. Netherlands : Springer, 2006. ISSN: 1389-2576.
15. **Zelinka, Ivan, Oplatková, Zuzana a Nolle, L.** Volume 6, Number 9. *International Journal of Simulation Systems, Science & Technology Special Issue on: Intelligent Systems*. [Online] August 2005. [Citace: 25. Březen 2011.] <http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm>. ISSN: 1473-804x.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACO	Ant colony optimalization
AP	Analytické programování
DE	Diferenciální evoluce
EA	Evoluční algoritmus
ES	Evoluční strategie
EVT	Evoluční výpočetní techniky
GA	Genetický algoritmus
GE	Gramatická evoluce
GFS	General instruction set
GP	Genetické programování
ISM	Immunology systém method
PGA	Paralelní genetický algoritmus
PS	Particle swarm Význam první zkratky.
SOMA	Samo-organizující se migrační algoritmus.
SS	Scatter Search.
UIS	Umělý imunitní systém

SEZNAM OBRÁZKŮ

Obrázek 1.1 Obecný cyklus evolučního algoritmu dle (1), str 27	12
Obrázek 1.2 Porovnání řešení problémů pomocí EVT a člověka dle (1), str. 29	14
Obrázek 1.3 Alan Mathison Turing dle (2).....	15
Obrázek 1.4 Příklad stromové struktury s ukázkou možné mutace dle (1), str. 256	22
Obrázek 1.5 Konvergence jedinců SOMA k leaderovi, dle (1), (vlastní zpracování)	27
Obrázek 1.6 Funkce PRT vektoru, dle (5), str. 20 (vlastní zpracování)	29
Obrázek 1.7 Konvergence populace do globálního extrému (vlastní zpracování)	33
Obrázek 2.1 Symbolicky znázorněná struktura množiny GFS dle [1], str. 272	36
Obrázek 2.2 Princip práce s diskrétní množinou dle [1], str. 273.....	37
Obrázek 2.3 Transformace diskrétního jedince na funkci pomocí AP dle (1), str. 273	38
Obrázek 2.4 Logo matematického softwaru Mathematica 7 for Students.....	40
Obrázek 5.1 Přehled nejzákladnějších tabulkových integrálů	54
Obrázek 5.2 Struktura hlavní matice mother	55
Obrázek 5.3 Struktura matice historie průběhu nejlepšího jedince	56
Obrázek 5.4 Struktura matice parametrů evoluce.....	56
Obrázek 5.5 Náhled vytvořeného vizualizačního interaktivního prostředí	57
Obrázek 5.6 Ovládací panel vizualizačního interaktivního prostředí.....	58

SEZNAM TABULEK

Tabulka 1.1 Populace jedinců s velikostí $N \times M$	18
Tabulka 1.2 Význam parametrů SOMA algoritmu, dle (1), str 226.....	25
Tabulka 1.3 Význam biologické terminologie v algoritmu SOMA dle (1), str. 226.....	27
Tabulka 1.4 Řídící parametry diferenciální evoluce, dle (1), str. 237	31
Tabulka 3.1 Hledání integrálu v symbolické formě dle (12), str. 259.....	43
Tabulka 5.1 Použitá množina GFS	47
Tabulka 5.2 Testovací funkce pro DE	48
Příloha PI: Tabulka 5.2.1 Symbolická integrace – funkce 1. - DE.....	49
Příloha PI: Tabulka 5.2.2 Symbolická integrace – funkce 2. - DE.....	49
Příloha PI: Tabulka 5.2.3 Symbolická integrace – funkce 3. - DE.....	49
Příloha PI: Tabulka 5.2.4 Symbolická integrace – funkce 4. - DE.....	49
Příloha PII: Tabulka 5.2.5 Symbolická derivace – funkce 1. - DE.....	50
Příloha PII: Tabulka 5.2.6 Symbolická derivace – funkce 2. - DE.....	50
Příloha PII: Tabulka 5.2.7 Symbolická derivace – funkce 5. – DE.....	50
Příloha PII: Tabulka 5.2.8 Symbolická derivace – funkce 6. – DE.....	50
Tabulka 5.3 Testovací funkce pro SOMA	51
Příloha PIII: Tabulka 5.3.1 Symbolická integrace – funkce 1. - SOMA	52
Příloha PIII: Tabulka 5.3.2 Symbolická integrace – funkce 2. - SOMA	52
Příloha PIII: Tabulka 5.3.3 Symbolická integrace – funkce 3. – SOMA	52
Příloha PIII: Tabulka 5.3.4 Symbolická integrace – funkce 4. - SOMA	52
Příloha PIV: Tabulka 5.3.5 Symbolická derivace – funkce 5. - SOMA.....	53
Příloha PIV: Tabulka 5.3.6 Symbolická derivace – funkce 2. – SOMA	53
Příloha PIV: Tabulka 5.3.7 Symbolická derivace – funkce 3. – SOMA	53
Příloha PIV: Tabulka 5.3.8 Symbolická derivace – funkce 4. - SOMA.....	53
Tabulka 5.4 Popis ovládacích prvků aplikace	58

SEZNAM PŘÍLOH

Příloha PI: Výsledky symbolické integrace – DE

Příloha PII: Výsledky symbolické derivace – DE

Příloha PIII: Výsledky symbolické integrace – SOMA

Příloha PIV: Výsledky symbolické derivace – SOMA

Příloha PV: Popis adresářové struktury algoritmů na přiloženém CD

PŘÍLOHA P I: VÝSLEDKY SYMBOLICKÉ INTEGRACE – DE

Příloha PI: Tabulka 5.3.1 Symbolická integrace – funkce 1. - DE

Parametry NP = 20, Generations = 10, F = 0.8, Cr = 0.6					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\cos[x] + \sin[x]$	$-\cos[x] + \sin[x]$	30.903 s	$\sin[0.806487 x]$	61.0244
2			32.479 s	$\cos[8.63923 - x]$	41.6922
3			28.343 s	$\sin[0.78525 - x]$	41.6922
4			31.419 s	$\sin[0.806487 x]$	61.0244
5			29.577 s	$-\cos[x]$	99.9981
6			26.571 s	$-\sin[2.35634 + x]$	41.6922
7			23.26 s	$-\sin[2.35634 + x]$	41.6922
8			30.202 s	$1.60437 \sin[0.707417 - x] \sin[0.647724 (-0.841471 - 1. x$	41.9858
9			21.279 s	$1.55698 \cos[1.40169 + \sin[\cos[1. \cdot 10^6 + 1.2809 x]]]$	64.6902
10			18.143 s	$-\sin[\sin[0.785245 - x]]$	52.421

a)

Parametry NP = 40, Generations = 30, F = 0.8, Cr = 0.6					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\cos[x] + \sin[x]$	$-\cos[x] + \sin[x]$	141.727 s	$-1.41421 \sin[0.785398 - x]$	$1.35586 \cdot 10^{-14}$
2			160.307 s	$-\cos[x] + \sin[x]$	0.0
3			137.921 s	$\sin[5.49484 + 1.00106 x]$	41.682
4			139.215 s	$-1.41421 \cos[0.785398 + x]$	$3.07497 \cdot 10^{-14}$
5			122.008 s	$-\sin[2.35634 + x]$	41.6922

b)

Parametry NP = 70, Generations = 40, F = 0.8, Cr = 0.6					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\cos[x] + \sin[x]$	$-\cos[x] + \sin[x]$	36.988 s	$-1.41421 \sin[0.785398 - x]$	$1.18863 \cdot 10^{-14}$
2			33.259 s	$(1.41421) \sin[41.6261 - x]$	$7.1854 \cdot 10^{-11}$
3			37.83 s	$\cos[1.5708 - x] - \cos[x]$	$6.05072 \cdot 10^{-15}$

c)

Příloha PI: Tabulka 5.3.2 Symbolická integrace – funkce 2. - DE

Parametry NP = 20, Generations = 10, F = 1.5, Cr = 0.6, Počet komponent = 10					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	-x + x Log[x]	6.599 s	0.20028 (-2.69993 + x) (1.42944 + x)	19.0356
2			7.192 s	-2.07856 + x + Cos[x]	40.4632
3			8.268 s	-0.999999 + 0.160905 x^2	21.7956
4			6.428 s	-2.07907 + x	79.1184
5			5.132 s	1. x (-1. + Log[x])	3.01738*10^-14
6			8.924 s	0.20028 (-2.69993 + x) (1.42944 + x)	19.0356
7			5.647 s	-2.76664 + 1.48388 x - Log[x]	38.0532
8			6.1 s	1. x (-1. + Log[x])	5.69527*10^-14
9			7.847 s	-1.07102 + 0.162188 x^2	18.8788
10			6.318 s	0.265822 (-2.81234 + x) x	40.5277

a)

Parametry NP = 30, Generations = 15, F = 1.5, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	-x + x Log[x]	23.899 s	((-1.10523*10^-8 - 1. x) (-1.10523*10^-8 + x) (1. - 1. Log[x]))/x	1.30305*10^-13
2			21.185 s	0.20028 (-2.69993 + x) (1.42944 + x)	19.0356
3			22.886 s	(2.04839 (-2.73625 + x) x)/(2.53584 + x)	6.26242
4			24.102 s	x (2.04839 - 10.7993/(2.53584 + x))	6.26242
5			20.764 s	1. x (-1. + Log[x])	1.1716*10^-13

b)

Parametry NP = 45, Generations = 15, F = 1, Cr = 0.4, Počet komponent = 20					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	-x + x Log[x]	44.274 s	-1. x + (0. + 1. x) Log[x]	0.
2			39.64 s	(-1. + Log[x]) (-6.12323*10^-17 + x + (2.19629*10^-18 Sin[x]))/(0.988982 - 1. x)	3.29285*10^-14
3			46.535 s	-1.05901 + (-0.290309 + 0.616619 x) Log[x] - 0.370505 Sin[Log[x]]	6.01092

c)

Příloha PI: Tabulka 5.3.3 Symbolická integrace – funkce 3. - DE

Parametry NP = 30, Generations = 15, F = 1, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 E^{-x^2}$	$1/2 E^{-x^2} (-1 - x^2)$	22.277 s	$-0.801527 + \text{Sin}[\text{Sin}[0.226855 + 0.311314 x]]$	5.69016
2			21.247 s	$-0.221662 - 0.0695257 \text{Cos}[x] + 0.0695257 \text{Log}[x^2]$	6.37043
3			22.246 s	$-x \text{Sin}[0.968419/x] + \text{Sin}[0.716745 \text{Log}[2 x]]$	6.14167
4			20.327 s	$-((1. \text{Sin}[0.44578 - 1.70017 x])/(0.954508 - 3.64864 x))$	6.28834
5			20.187 s	$-0.239299 + 0.11272 \text{Log}[x] - 0.11272 \text{Sin}[1 - 0.690317 x]$	6.6872
6			21.107 s	$-0.347242 + 0.0768489 x - 0.137367 \text{Cos}[x]$	5.94364
7			20.373 s	$-((0.295785 (-0.00458698 + x))/(x (0.339776 + x)))$	12.0812
8			22.839 s	$-0.607954 + 0.139507 x + 0.269686 \text{Sin}[\text{Sin}[0.716272 x]]$	4.59298
9			23.744 s	$(-0.343617 + 0.00654698/x + 0.241321 \text{Log}[x]) \text{Sin}[2.51091 - x]$	4.06563
10			23.915 s	$7.38717 \text{Sin}[0.0206058 - 0.0756049/x] \text{Sin}[x]$	4.7491

a)

Parametr NP = 40, Generations = 20, F = 1.3, Cr = 0.8, Počet komponent = 20					
y					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 E^{-x^2}$	$1/2 E^{-x^2} (-1 - x^2)$	55.77 s	$\text{Log}[0.734802 + 0.0469859 x] (0.759313 + \text{Sin}[1.65792 - x])$	3.49621
2			56.395 s	$-0.0315757 (-5.56683 + x) (-3.58824 + x) \text{Cos}[0.709041 - x]$	2.32553
3			58.766 s	$\text{Log}[\text{Cos}[(0.931391 + 0.000823571 \text{Log}[x]) \text{Sin}[x])/x]]$	2.58851
4			55.083 s	$(0.142436 (-3.77708 + x) \text{Sin}[x])/x$	4.22532
5			60.934 s	$(x \text{Cos}[1.88889 + x])/(-0.00374167 + 1.36509 x^3 + x^4 + x \text{Cos}[x])$	2.21571

b)

Parametry NP = 60, Generations = 15, F = 1, Cr = 0.6, Počet komponent = 30					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 E^{-x^2}$	$1/2 E^{-x^2} (-1 - x^2)$	51.434 s	$-0.0478248/(-0.116008 + 0.530908 x^2 - 1. \text{Sin}[2.60884 - 0.720284 x - \text{Cos}[2.41915 + x]])$	0.919543
2			46.284 s	$-0.000789854 - 0.502569 E^{(-0.00140409 x^3 (222.637 + x))}$	0.263405
3			47.44 s	$-4.49525 (11.5327 + x)^{-x^2} (x/E)$	0.771823

c)

Příloha PI: Tabulka 5.3.4 Symbolická integrace – funkce 4. - DE

Parametry NP = 20, Generations = 15, F = 0.9, Cr = 0.7, Počet komponent = 15					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x \sin[x^2]$	$-(1/2) \cos[x^2]$	16.489 s	$0.124249 (-4.00417 + x) \cos[1.40479 x]$	42.4631
2			20.826 s	$-\sin[0.644792 \sin[\sin[0.0012006 + \cos[x^2]]]]$	22.9984
3			17.051 s	$-(\sin[x (0.381582 + x)]/x)$	41.2638
4			13.104 s	$-0.5 \sin[1.5708 - x^2]$	$4.51264 \cdot 10^{-14}$
5			17.332 s	$\cos[1.56909 + (0.257158 \sin[2.81071 x])/x]$	45.3996
6			20.031 s	$-0.00115173 - 0.569738 \sin[\sin[1.56839 - x^2]]$	2.25482
7			18.922 s	$-0.841471 \sin[1.50838 - x^2]$	35.2717
8			20.483 s	$-0.610608 + \cos[0.617884 + \cos[x^2]]$	18.9945
9			16.489 s	$-((0.247547 (-0.0107945 + \sin[2.7954 x]))/x)$	41.3492
10			16.115 s	$\sin[0.158118 - 0.361384 x + \log[x]]$	47.2055

a)

Parametry NP = 60, Generations = 15, F = 0.6, Cr = 0.4, Počet komponent = 15					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x \sin[x^2]$	$-(1/2) \cos[x^2]$	42.775 s	$0.626303 \sin[\sin[\cos[3.14485 + x^2]]]$	3.75002
2			43.321 s	$-0.5 \cos[1. x^2]$	0.
3			40.217 s	$-0.569999 \sin[\cos[0.00197752 + x^2]]$	2.2688
4			48.032 s	$-0.5 \cos[x^2]$	$4.04798 \cdot 10^{-9}$
5			40.108 s	$-3.66617 \cdot 10^{-17} (1.36382 \cdot 10^{16} + x) \cos[x^2]$	$2.87444 \cdot 10^{-15}$

b)

Parametry NP = 20, Generations = 15, F = 1.3, Cr = 0.8, Počet komponent = 20					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x \sin[x^2]$	$-(1/2) \cos[x^2]$	21.653 s	$0.0667751 \operatorname{Csc}[1. x^2] \sin[2. x^2]$	$1.28023 \cdot 10^{-15}$
2			17.535 s	$-1.70495 \cdot 10^{-17} - 0.5 \cos[1.83759 \cdot 10^{-15} - x^2]$	$1.27979 \cdot 10^{-13}$
3			20.766 s	$0.0667751 \operatorname{Csc}[1. x^2] \sin[2. x^2]$	$1.28023 \cdot 10^{-15}$

c)

PŘÍLOHA P II: VÝSLEDKY SYMBOLICKÉ DERIVACE – DE

Příloha PII: Tabulka 5.3.5 Symbolická derivace – funkce 1. - DE

Parametry NP = 20, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\text{Cos}[x] + \text{Sin}[x]$	$\text{Cos}[x] - \text{Sin}[x]$	16.567 s	$\text{Sin}[1. + 1.31943 x]$	76.9243
2			19.048 s	$1.41421 \text{Sin}[2.35619 + x]$	$4.82062 \cdot 10^{-14}$
3			15.913 s	$\text{Cos}[0.783692 + x]$	59.6516
4			16.427 s	$\text{Cos}[1.18922 x]$	59.6516
5			15.21 s	$-1.11022 \cdot 10^{-16} + \text{Sin}[1.5708 - x] - \text{Sin}[x]$	$1.7375 \cdot 10^{-14}$
6			15.662 s	$\text{Cos}[113.881 + x]$	41.6905
7			16.646 s	$(1.41421 + 1.93836 \cdot 10^{-17} x) \text{Sin}[0.785398 - x]$	$1.5217 \cdot 10^{-14}$
8			20.187 s	$\text{Cos}[74.6145 - x]$	41.6905
9			17.175 s	$-0.00209298 (671.872 + x) \text{Cos}[2.3562 - x]$	0.345644
10			17.643 s	$\text{Cos}[1.97437 \cdot 10^{-9} + x] - 1. \text{Sin}[x]$	$9.21485 \cdot 10^{-15}$

a)

Parametry NP = 40, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\text{Cos}[x] + \text{Sin}[x]$	$\text{Cos}[x] - \text{Sin}[x]$	33.321 s	$\text{Sin}[1.65667 \text{Sin}[2.35541 + 1.00021 x]]$	25.3587
2			30.701 s	$1.41421 \text{Sin}[2.35619 + 1. x]$	$3.93387 \cdot 10^{-13}$
3			33.759 s	$-0.107014 + \text{Cos}[x] + \text{Cos}[1.14019 x] - \text{Sin}[0.98784 + x]$	18.9021
4			35.209 s	$\text{Cos}[x] - 1. \text{Sin}[x]$	$2.26485 \cdot 10^{-14}$
5			43.135 s	$\text{Sin}[1.65648 \text{Cos}[0.785139 + x]]$	25.3545

b)

Parametry NP = 50, Generations = 10, F = 0.9, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\text{Cos}[x] + \text{Sin}[x]$	$\text{Cos}[x] - \text{Sin}[x]$	40.857 s	$1.41421 (0. x + \text{Sin}[0.785398 - x])$	$1.83898 \cdot 10^{-14}$
2			37.893 s	$-0.000239827 - 1.6033 \text{Sin}[\text{Sin}[5.49799 + x]]$	6.31721
3			40.95 s	$1.41421 \text{Cos}[0.785398 + 1. x]$	$1.6649 \cdot 10^{-11}$

c)

Příloha PII: Tabulka 5.3.6 Symbolická derivace – funkce 2. - DE

Parametry NP = 20, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 10					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	1/x	11.419 s	1./x	3.51108*10 ⁻¹⁴
2			8.908 s	1./x	0.
3			10.982 s	1/x	3.51108*10 ⁻¹⁴
4			8.69 s	1. (0. + 1/x)	0.
5			11.279 s	(1.00018 Cos[x])/x	60.9369
6			8.299 s	(0.830388 Cos[Cos[1.55194 + x]])/x	54.3634
7			14.508 s	2.09226*10 ⁻¹⁸ + 1./x	3.84137*10 ⁻¹⁴
8			10.842 s	-(Sin[10.9956 - 9.27581*10 ⁻¹⁰ x Sin[x]]/x)	0.
9			12.106 s	Cos[5.47276*10 ⁻⁹ - 1.05614*10 ⁻⁹ x]/(0. + 1. x)	8.74301*10 ⁻¹⁵
10			11.996 s	1./x	2.95644

a)

Parametry NP = 20, Generations = 10, F = 0.9, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	1/x	14.586 s	1./x - 1.77889*10 ⁻¹⁸ x	0.
2			13.254 s	-(Sin[10.9956 - 9.27581*10 ⁻¹⁰ x Sin[x]]/x)	0.
3			10.736 s	2.09226*10 ⁻¹⁸ + 1./x	3.84137*10 ⁻¹⁴
4			9.079 s	1./x	9.92539*10 ⁻¹⁴
5			11.185 s	0.0328503 + 0.999372/x - 0.0102286 x	21.5415

b)

Parametry NP = 20, Generations = 10, F=0.9, Cr = 0.8, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	1/x	11.076 s	1./x	2.10942*10 ⁻¹⁴
2			12.199 s	1.77287*10 ⁻¹⁷ + 1./x + 2.0776*10 ⁻¹⁷ Sin[x]	4.99600*10 ⁻¹⁶
3			10.686 s	-9.9646*10 ⁻¹⁸ + 1./x	5.53169*10 ⁻¹⁴

c)

Příloha PII: Tabulka 5.3.7 Symbolická derivace – funkce 5. – DE

Parametry NP = 20, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 15					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 \cos[x]$	$3x^2 \cos[x] - x^3 \sin[x]$	9.984 s	$-1. (28.144x + \cos[x]) \sin[3.83771 - x]$	2902.5
2			10.951 s	$29.2292x \sin[1 - x] \sin[\log[x]]$	3437.69
3			10.561 s	$5.84248x^2 \sin[0.623178 - x]$	769.948
4			10.702 s	$-1. (1. - 29.8868x) \sin[2.4215 - \cos[x]]$	2624.18
5			8.658 s	$x + (1. - 1.45319x)x^2 \sin[x]$	3804.19
6			9.219 s	$36.8655(0.731358 - 1.x) \cos[0.474343x]$	2818.77
7			11.263 s	$x(17.5665 + x)(0.322332 + \cos[0.712541 + x])$	1192.61
8			8.985 s	$10.9169(-3.10788 + x)x$	3533.03
9			9.563 s	$17.4682x + 59.9904 \cos[x]$	4823.56
10			7.675 s	$-135.381(0.0412934 - 0.013774x)x^2$	3209.2

a)

Parametry NP = 30, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 20					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 \cos[x]$	$3x^2 \cos[x] - x^3 \sin[x]$	19.594 s	$-61.2952 + 32.1326x + 60.3151 \cos[x]$	1802.37
2			22.277 s	$0.997922 + \sin[x] + 28.567x \sin[1.92025 + 1.10432x]$	3203.03
3			19.266 s	$77.7193 + 106.92 \cos[0.16656(-5.61409 + x)(4.11106 + x)]$	174.734
4			19.906 s	$x(-20.2802 + 8.02057x) + 28.2092 \cos[x]$	2571.9
5			19.142 s	$-12.7527x + 1.27868x^3$	3231.68

b)

Parametry NP = 60, Generations = 15, F = 1, Cr = 0.6, Počet komponent = 30					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x^3 \cos[x]$	$3x^2 \cos[x] - x^3 \sin[x]$	79.013 s	$1.58215(x - 1.(-19.4407 + 78.079x)(0.0249402 - 1. \cos[7.08081 - 1.07339x] + \sin[21.1142 - x]))$	121.557
2			80.138 s	$33.4511 \cos[14.8546 - x](1.02805 - 1.x + \sin[x + \sin[19.1921 + x]])$	1302.47
3			80.2 s	$x(12.9409 + x - 29.6402 \log[x])(-0.00542805 - 1. \sin[0.647531 - x])$	562.108

c)

Příloha PII: Tabulka 5.3.8 Symbolická derivace – funkce 6. – DE

Parametry NP = 20, Generations = 10, F = 0.8, Cr = 0.6, Počet komponent = 10					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(Log[x] Sin[x])/Log[2]	(Cos[x] Log[x])/Log[2] + Sin[x]/(x Log[2])	6.63 s	1.96666 Sin[Sin[3.58455 - 1.42673 x]]	71.1095
2			5.523 s	0.501436 x Cos[0.248029 - x]	58.3284
3		6.459 s	-(0.0561538/x) + 0.457429 x Cos[x]	70.0735	
4		7.457 s	-Cos[2.22903 + 0.86283 x + Log[x]]	87.8227	
5		6.849 s	Cos[x - Cos[Cos[x]]/x^2]	107.173	
6		6.178 s	0.501436 x Cos[0.248029 - x]	58.3284	
7		6.832 s	0.457433 x Cos[x]	76.6279	
8		7.082 s	1.72651 Cos[1.98117 - 1.41765 x]	69.8228	
9		6.334 s	0.485265 x (-0.0211114 x + Cos[x])	61.6323	
10		8.128 s	1.19721 Cos[x] Log[x]	92.0789	

a)

Parametry NP = 40, Generations = 15, F = 0.8, Cr = 0.6, Počet komponent = 20					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(Log[x] Sin[x])/Log[2]	(Cos[x] Log[x])/Log[2] + Sin[x]/(x Log[2])	48.267 s	0.644451 x Cos[x] - 1. Sin[0.767906 - x]	58.2837
2			44.476 s	1.82499 Sin[3.17283 - 0.988192/x - 1.27267 x]	58.3286
3			46.894 s	-1.99822 + 0.634157 x + 1.49699 (Cos[x] + Sin[x])	45.1697
4			49.67 s	-0.501438 x Cos[34.3095 + 1. x]	58.3285
5			46.379 s	-1.17844 (Cos[298.676 + x] + Cos[0.432478 + 1.47652 x])	41.4136

b)

Parametry NP = 150, Generations = 15, F = 1.2, Cr = 0.7, Počet komponent = 30					
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(Log[x] Sin[x])/Log[2]	(Cos[x] Log[x])/Log[2] + Sin[x]/(x Log[2])	222.894 s	-0.962422 Log[x] Sin[3862.34 - x] + Sin[x + Cos[0.381781 + Sin[x]]]	14.4382
2			238.026 s	-1.42469 + 0.406814 x + Sin[2.50049 - x] + Sin[1.343 x]	14.4213
3			234.563 s	2.00896 Sin[7.15068 + 1.63229 x - 1.57558 Log[x]]	18.2859

c)

PŘÍLOHA P III: VÝSLEDKY SYMBOLICKÉ INTEGRACE – SOMA

Příloha PIII: Tabulka 5.4.1 Symbolická integrace – funkce 1. - SOMA

Parametry PathLength = 3, Step = 0.11, PRT = 0.4, PopSize = 10, Migrations = 10, AcceptedError = -0.01, Počet komponent = 10					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(0.05 + 0.25	-0.05 Cos[x] -	69.061 s	(-1.53492 + 0.363837 x) x (-1.28858 - 0.125235 x + Sin[x])	12.3433
2	x^2) Sin[x]	0.25 (-2. + x^2)	65.958 s	-0.994294 x (0.0774333 x - Sin[4.92393 + 1. x])	57.6066
3		Cos[x] + 0.5 x	70.777 s	-1.14687 (-0.439153 + x) (0.59227 + Cos[x]) + Cos[Cos[x]]	52.7067
4		Sin[x]	66.815 s	1.70201 - 0.866206 x - (56.5887 x Cos[x])/(52.9401 + x)	28.8517
5			72.587 s	-0.840286 (2.36695 - 0.561846 x) x (-1.41783 + Sin[x])	32.7077
6			78.765 s	1.44822 - 0.742894 x - 0.996837 x Sin[1.46776 - x]	27.7715
7			70.31 s	1.66121 Cos[Cos[x]] + x (-0.672069 - 1. Sin[1.40602 - x])	26.3559
8			77.829 s	0.271041 x (0.0485588 + x) (0.0791538 - 1. Cos[4.8874 - 0.809498 x])	23.9754
9			68.094 s	(91.3482 (-0.486151 + x^2) Cos[0.457159 + x])/(354.117 + x)	42.2869
10			63.991 s	2.1621 - x - 0.940068 x Cos[x]	35.5791

a)

Příloha PIII: Tabulka 5.4.2 Symbolická integrace – funkce 2. - SOMA

Parametry PathLength = 3, Step = 0.11, PRT = 0.4, PopSize = 10, Migrations = 10, AcceptedError = -0.01, Počet komponent = 10					
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	x Cos[x]	Cos[x] + x Sin[x]	51.168 s	Cos[x] + x Sin[x]	0.
2			65.926 s	1. Cos[x] + x Sin[x]	0.
3			54.944 s	x Cos[1. - 0.929087 x]	28.9385
4			59.421 s	x Sin[0.526803 + 0.937399 x]	26.4201
5			69.608 s	x Sin[x] + Sin[1.5708 + x]	1.12133*10^-14
6			58.812 s	-2.14882*10^-18 + Cos[x] + x Sin[x]	0.
7			53.21 s	Cos[x] + x Sin[x]	3.55781*10^-13
8			62.478 s	1. Cos[x] + x Sin[x]	4.32378*10^-13
9			57.244 s	Cos[x] + x Sin[x]	0.
10			59.249 s	x Cos[115964. - 0.937399 x]	26.5411

a)

Příloha PIII: Tabulka 5.4.3 Symbolická integrace – funkce 3. – SOMA

Parametry	PathLength = 3, Step = 0.8, PRT = 0.33, PopSize = 10, Migrations = 12, AcceptedError = -0.01, Počet komponent = 12				
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	-x + x Log[x]	10.483 s	x (-1. + 1. Log[x])	0.
2			10.109 s	-1. x + 1. x Log[x]	0.
3			10.842 s	-1.10164 + 0.163468 x ²	40.3885
4			12.136 s	x Cos[10.1109 + 0.309082 x]	30.4509
5			8.69 s	1. x (-1. + 1.11022*10 ⁻¹⁵ x + Log[x])	1.48551*10 ⁻¹²
6			10.265 s	0.738953 (-2.7281 + x) Log[1.14933 + x]	1.47925
7			14.82 s	x (-1. + 1. Log[x])	4.77535*10 ⁻¹⁴
8			9.235 s	1. x (-1. + Log[x])	2.81025*10 ⁻¹⁴
9			7.722 s	x (-1. + 1. Log[x])	5.37244*10 ⁻¹⁴
10			13.713 s	-8.36018*10 ⁻¹⁸ - 1. x + x Log[x]	2.79915*10 ⁻¹⁴

a)

Příloha PIII: Tabulka 5.4.4 Symbolická integrace – funkce 4. - SOMA

Parametry	PathLength = 3, Step = 0.11, PRT = 0.4, PopSize = 10, Migrations = 10, AcceptedError = -0.01, Počet komponent = 10				
Měření č.	Předpis funkce	Integrál funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(0.05 + 0.25	-0.05 Cos[x] -	69.061 s	(-1.53492 + 0.363837 x) x (-1.28858 - 0.125235 x + Sin[x])	12.3433
2	x ²) Sin[x]	0.25 (-2. + x ²)	65.958 s	-0.994294 x (0.0774333 x - Sin[4.92393 + 1. x])	57.6066
3		Cos[x] + 0.5 x	70.777 s	-1.14687 (-0.439153 + x) (0.59227 + Cos[x]) + Cos[Cos[x]]	52.7067
4		Sin[x]	66.815 s	1.70201 - 0.866206 x - (56.5887 x Cos[x])/(52.9401 + x)	28.8517
5			72.587 s	-0.840286 (2.36695 - 0.561846 x) x (-1.41783 + Sin[x])	32.7077
6			78.765 s	1.44822 - 0.742894 x - 0.996837 x Sin[1.46776 - x]	27.7715
7			70.31 s	1.66121 Cos[Cos[x]] + x (-0.672069 - 1. Sin[1.40602 - x])	26.3559
8			77.829 s	0.271041 x (0.0485588 + x) (0.0791538 - 1. Cos[4.8874 - 0.809498 x])	23.9754
9			68.094 s	(91.3482 (-0.486151 + x ²) Cos[0.457159 + x])/(354.117 + x)	42.2869
10			63.991 s	2.1621 - x - 0.940068 x Cos[x]	35.5791

a)

PŘÍLOHA P IV: VÝSLEDKY SYMBOLICKÉ DERIVACE – SOMA

Příloha PIV: Tabulka 5.4.5 Symbolická derivace – funkce 5. - SOMA

Parametry	PathLength = 3, Step = 0.5, PRT = 0.33, PopSize = 10, Migrations = 12, AcceptedError = -0.01, Počet komponent = 10				
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$\text{Cos}[x]/2 + 3 \text{Sin}[x]$	$3 \text{Cos}[x] - \text{Sin}[x]/2$	16.754 s	$3.04138 \text{Cos}[0.165149 + x]$	$3.39243 \cdot 10^{-14}$
2			10.983 s	$-0.502632 + 0.158091 x + 2.98806 \text{Cos}[x]$	29.5226
3			13.447 s	$3.04138 \text{Cos}[0.165149 + x]$	$3.07185 \cdot 10^{-14}$
4			12.823 s	$-3.04138 \text{Cos}[3.30674 + x]$	$8.14696 \cdot 10^{-14}$
5			13.291 s	$0.0380573 x + 3.00081 \text{Cos}[x]$	44.0311
6			14.368 s	$3.04138 \text{Sin}[1.40565 - x]$	$4.66814 \cdot 10^{-14}$
7			16.49 s	$3.04138 \text{Cos}[0.165149 + 1. x]$	$3.36051 \cdot 10^{-14}$
8			18.471 s	$3.04138 \text{Cos}[0.165149 + x]$	$4.89261 \cdot 10^{-14}$
9			15.538 s	$3.04138 \text{Cos}[0.165149 + 1. x]$	$3.36051 \cdot 10^{-14}$
10			13.354 s	$0.0380573 x + 3.00081 \text{Cos}[x]$	48.1175

a)

Příloha PIV: Tabulka 5.4.6 Symbolická derivace – funkce 2. – SOMA

Parametry	PathLength = 3, Step = 0.4, PRT = 0.22, PopSize = 10, Migrations = 10, AcceptedError = -0.01, Počet komponent = 15				
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	$x \text{Cos}[x]$	$\text{Cos}[x] + x \text{Sin}[x]$	37.69 s	$1. \text{Sin}[1.5708 - x] - 1. x \text{Sin}[x]$	$2.94209 \cdot 10^{-15}$
2			37.737 s	$x \text{Sin}[0.549271 - 1.06624 x]$	30.6131
3			38.08 s	$\text{Cos}[1.23657 \cdot 10^{-10} - x] - x \text{Sin}[x]$	$1.23556 \cdot 10^{-8}$
4			51.683 s	$1. \text{Cos}[x] - 1. x \text{Sin}[x]$	0.
5			40.077 s	$-1. \text{Sin}[0.268167 - x] (-0.0229862 - 1. x + \text{Sin}[\text{Cos}[x]])$	35.1754
6			41.481 s	$(0.0314701 + x) \text{Cos}[1.33457 + x]$	41.1755
7			46.504 s	$-\text{Cos}[3.14159 + x] - 1. (8.87601 \cdot 10^{-9} + x) \text{Sin}[x]$	$4.33542 \cdot 10^{-14}$
8			33.962 s	$\text{Cos}[x] + x (2.34533 \cdot 10^{-19} - 1. \text{Sin}[x])$	0.
9			38.891 s	$\text{Sin}[1.5708 - x] - 1. x \text{Sin}[x]$	23.7101
10			42.666 s	$-1. x \text{Cos}[1.5708 - x] + \text{Cos}[x]$	$3.97182 \cdot 10^{-13}$

a)

Příloha PIV: Tabulka 5.4.7 Symbolická derivace – funkce 3. – SOMA

Parametry	PathLength = 3, Step = 0.8, PRT = 0.33, PopSize = 10, Migrations = 12, AcceptedError = -0.01, Počet komponent = 12				
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	Log[x]	1/x	21.325 s	1./x	0.
2			22.667 s	1./x	0.
3			17.878 s	1./x	0.
4			24.898 s	1./x	4.74898*10 ⁻¹⁴
5			19.437 s	1./x	6.03684*10 ⁻¹⁴
6			23.727 s	1./x	0.
7			23.104 s	1./x	0.
8			22.028 s	1./x	2.27596*10 ⁻¹⁴
9			28.547	1./x	0.
10			20.144	1./x	0.

a)

Příloha PIV: Tabulka 5.4.8 Symbolická derivace – funkce 4. - SOMA

Parametry	PathLength = 3, Step = 0.11, PRT = 0.4, PopSize = 10, Migrations = 10, AcceptedError = -0.01, Počet komponent = 10				
Měření č.	Předpis funkce	Derivace funkce	Čas evoluce	Syntetizovaná funkce	Hodnota účelové funkce
1	(0.05 + 0.25	(0.05 + 0.25	72.275	0.048234 x ² Sin[1.95921 - x] (4.96138 + Sin[0.363321 x])	30.8359
2	x ²) Sin[x]	x ²) Cos[x] +	61.434	(0.432215 (-0.575563 + x) (-0.19781 + x) (3.54296 - 1. x + x Cos[x]))/x	41.7947
3		0.5 x Sin[x]	66.675 s	-4.20242 - 41.3596 Log[Cos[Sin[Sin[0.419948 + 0.767185 x] - 0.500834 Sin[x]]]]	30.0972
4			80.84 s	Cos[0.909774 x] + Cos[Sin[x]] + 1.75824 (-1.45117 + x) Sin[1.13184 + x]	14.9208
5			62.65 s	(-0.370425 - 0.27906 x) (-1. + x) Cos[12588. + x]	46.6926
6			84.989 s	2.04012 (0.843946 x + Cos[2.73967 - Cos[x]]) Sin[Sin[1.234 x]]	40.9437
7			83.398 s	0.64004 x (0.459224 + Log[x]) Sin[0.494442 + 1.14306 x]	36.2391
8			76.191 s	1.60213 (-0.535663 + x) (0.197887 + Sin[1.02223 + x])	31.5549
9			74.959 s	0.451909 (-1.8 + x) x (-0.264462 + Cos[x]) - 0.730762 Cos[1.67471 + x]	9.9627
10			60.232 s	0.291586 x ² Cos[0.924866 x]	49.7015

a)

PŘÍLOHA P IV: POPIS ADRESÁŘOVÉ STRUKTURY ALGORITMŮ NA PŘILOŽENÉM CD

ADRESÁŘOVÁ STRUKTURA:

- Vizualizace
 - Vizualizace_DE.nb Soubor obsahující zdrojový kód pro vizualizaci průběhu syntézy funkcí pomocí algoritmu DE
 - Vizualizace_SOMA.nb Soubor obsahující zdrojový kód pro vizualizaci průběhu syntézy funkcí pomocí algoritmu SOMA
- Algoritmy AP
 - AP_DERIVACE_DE.nb Zdrojový kód pro jednu simulaci symbolického derivování pomocí AP - DE
 - AP_DERIVACE_SOMA.nb Zdrojový kód pro jednu simulaci symbolického derivování pomocí AP - SOMA
 - AP_INTEGRACE_DE.nb Zdrojový kód pro jednu simulaci symbolického integrování pomocí AP - DE
 - AP_INTEGRACE_SOMA.nb Zdrojový kód pro jednu simulaci symbolického integrování pomocí AP - SOMA