Analytické programování aplikované na operátory evolučních algoritmů

Bc. Lucie Metelková

Diplomová práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav řízení procesů
akademický rok: 2005/2006

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

Jméno a příjmení:	**Bc. Lucie METELKOVÁ**
Studijní program:	**N 2807 Chemické a procesní inženýrství**
Studijní obor:	**Automatizace a řídicí technika**
Téma práce:	**Analytické programování aplikované na operátory evolučních algoritmů**

Zásady pro vypracování:

Cílem diplomové práce je provést literární rešerši v oblasti evolučních algoritmů a jejich operátorů. Výstupem by měl být jejich algoritmický popis, který by byl vhodný k použití pro symbolickou regresi.

1. Pomocí dostupné literatury a zdrojů na internetu proveďte kompletní a vyčerpávající rešerši současných metod evolučních algoritmů
2. Popište jednotlivé dostupné operátory evolučních algoritmů
3. Navrhněte algoritmický způsob jejich zápisu
4. Vyberte jeden evoluční algoritmus, na kterém vyzkoušíte pomocí analytického programování šlechtění operátorů
5. Závěrem zhodnoťte výsledky

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. Kvasnička V., Pospíchal J., Tiňo P., Evolučné algoritmy, STU Bralislava, 2000, ISBN 80-227-1377-5

2. Zelinka Ivan Umělá inteligence v problémech globální optimalizace, BEN, 2002, 190 p. ISBN 80-7300-069-5

3. Mařík V. a kol., Artificial Intelligence IV., 2004, Academia, Praha, Czech edition

4. Zelinka I.,Oplatkova 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

Vedoucí diplomové práce:	**Ing. Zuzana Oplatková** Ústav aplikované informatiky
Datum zadání diplomové práce:	**14. února 2006**
Termín odevzdání diplomové práce:	**26. května 2006**

Ve Zlíně dne 25. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Petr Dostál, CSc.
ředitel ústavu

ABSTRAKT

Cílem této diplomové práce je provést rešerši v oblasti evolučních algoritmů a jejich operátorů. Výstupem je pak jejich algoritmický zápis, což znamená separace na jednotlivé operátory zvoleného algoritmu, které jsou pak vhodné k použití pro analytické programování. Je zvolen evoluční algoritmus diferenciální evoluce, na němž se zkouší pomocí analytického programování šlechtění operátorů.

Klíčová slova: evoluční algoritmus, operátory, analytické programování, diferenciální evoluce, šlechtění

ABSTRACT

Aim of this master thesis is carry out research in the area of evolutional algorithms and their operators. Output is their algoritmic record, which means separation on individual operators of selected algorithm that are then fitted for using by means of analytic programming. As an algorithm diferencial evolution was selected. Its operators were used to breeding an algorithm back by means of Analytic programming.

Keywords: evolutionary algorithms, operators, analytic programming, diferencial evolution, breeding.

Tímto bych chtěla poděkovat vedoucí mé diplomové práce Ing. Zuzaně Oplatkové za cenné rady, připomínky a ochotu věnovat mi čas během řešení problémů.

OBSAH

ÚVOD

Tato diplomová práce má za úkol vypracovat kompletní rešerši evolučních algoritmů a jejich operátorů, popsání jednotlivých operátorů, navrhnutí jejich algoritmického zápisu a v neposlední řadě vyzkoušet pomocí analytického programování šlechtění operátorů na jednom zvoleném evolučním algoritmu.

Teoretická část se zabývá nejprve přehledem základních evolučních algoritmů a jejich popisu. V další kapitole následuje podrobný popis jednotlivých operátorů evolučních algoritmů. Je zde zobrazeno, zda je operátor aplikován na celou populaci či jednotlivého jedince. U každého popisu operátoru nechybí seznam evolučních algoritmů, které daný operátor využívají. Pokud má operátor jiný význam než u klasického pojetí, je to zde blíže vysvětleno u konkrétního algoritmu, jako např. mutace nebo křížení. V poslední kapitole teoretické části je popsán princip analytického programování, pomocí něhož se v praktické části šlechtí operátory diferenciální evoluce.

Praktická část začíná popisem diferenciální evoluce, principem jejího fungování a použití jejích operátorů. Následuje popis separovaných operátorů v programu Mathematica.

V další části je popsána konstrukce účelové funkce a jsou zvoleny 2 funkce, na nichž je provedena analýza pro evoluční algoritmy SOMA a DE. Zjišťuje se, zda jsou tyto algoritmy schopny najít extrém na těchto funkcích. První zvolená funkce je jednoduchá, typu mísa s jedním extrémem. Druhá funkce se nazývá Schwefelova, kterou řadíme mezi multimodální, tzn. má mnoho extrémů.

Poslední část praktické části se zabývá aplikací analytického programování na operátory diferenciální evoluce a zjišťuje se, zda je analytické programování schopno znovu sestavit ze separovaných operátorů diferenciální evoluci či vytvořit nový algoritmus s operátory DE.

Závěr této práce se zabývá zhodnocením dosažených výsledků a možností dalšího řešení.

I. TEORETICKÁ ČÁST

1 PŘEHLED ALGORITMŮ A JEJICH ZÁKLADNÍ POPIS

V této části je uveden seznam základních a zároveň nejpoužívanějších typů evolučních algoritmů a jejich principů.

1.1 Horolezecký algoritmus

Horolezecký algoritmus patří do skupiny tzv. gradientních algoritmů. Uvažujme řešení ve tvaru vektoru (1) a kriteriální funkci (2).

$$a=(a_1,a_2,..,a_n) \quad (1)$$

$$f = f(a) \quad (2)$$

V každém kroku algoritmu se prohledávají sousední řešení aktuálního řešení a vybírá se takové, jehož hodnota kriteriální funkce je nejlepší. To se pak v dalším kroku stává aktuálním řešením. Tento postup se opakuje až do kroku, kdy žádné ze sousedních řešení nemá lepší hodnotu f. Tento algoritmus je ovšem použitelný pouze pro takové f, které nemají lokální extrém. Pokud má f lokální extrémy, pak v závislosti na volbě počátečního řešení, může dojít k uvíznutí algoritmu v lokálním minimu resp. maximu. Proto se v této podobě používá spíše pro lokální optimalizaci - rychlé nalezení nejbližšího extrému. Jedinec je v tomto algoritmu reprezentován binárním řetězcem.[1]

1.1.1 Stochastický horolezecký algoritmus

Je to verze horolezeckého algoritmu obohacena o stochastickou složku. Patří mezi gradientní metody, tzn., že prohledává prostor možných řešení ve směru největšího spádu. Díky své gradientní povaze velmi často uvázne v lokálním extrému. Princip funguje tak, že se vždy vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se pomocí konečného souboru transformací navrhne určité okolí a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí. Celý proces se pak iterativně opakuje. Během procesu se zaznamenává nejlepší nalezené řešení, které po ukončení slouží jako nalezené optimum. Stochastický horolezecký algoritmus je v podstatě jen mnohonásobné zopakování standardního horolezeckého algoritmu, pokaždé z jiné, náhodně zvolené pozice. Nevýhodou tohoto algoritmu je to, že v něm může za určitých podmínek dojít k zacyklení a řešení tak uvízne v lokálním extrému. Jedinec je v tomto algoritmu reprezentován binárním řetězcem. [1]

1.2 Tabu search (zakázané prohledávání)

Jde o vylepšenou verzi horolezeckého algoritmu. Jejím tvůrcem je Prof. Fred Glover z Univerzity of Colorado. Vylepšení spočívá v tom, že do horolezeckého algoritmu je zavedena tzv. krátkodobá paměť, jejímž úkolem je pamatovat si ty transformace, pomocí nichž byl vypočítán aktuální střed. To má v konečném důsledku ten efekt, že nedochází k zacyklení díky zakázanému použití těchto transformací. Odtud také plyne název. Tato metoda byla vylepšena ještě o tzv. dlouhodobou paměť, která obsahuje transformace, jenž nejsou v paměti krátkodobé, ale byly často použity. Jejich použití je pak penalizováno, což snižuje četnost jejich použití. Na rozdíl od algoritmu horolezeckého Tabu Search tak snadno neuvízne v lokálních extrémech.[2]

1.3 Evoluční programování

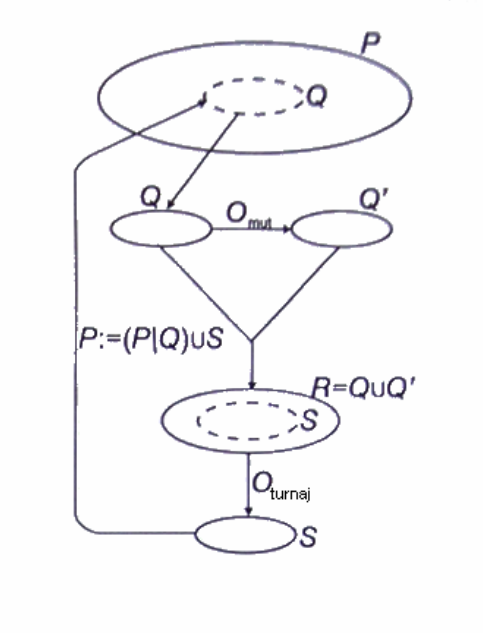
Evoluční programování patří mezi stochastické optimalizační algoritmy, které je opět možno chápat jako jednoduché zevšeobecnění horolezeckého algoritmu. Základní úloha spočívá v řešení optimalizačního problému (3).

$$\alpha_{opt} = \arg\min_{\alpha \in \{0,1\}^k} f(\alpha) \tag{3}$$

Kde $f:\{0,1\}^k \rightarrow R$ je funkce, která zobrazuje binární vektor délky k na reálné čísla. Každý prvek α z populace P je ohodnocen účelovou hodnotou f(α). Principje znázorněn na (Obr.1). Z populace P vybereme podmnožinu Q – populaci rodičů, kde kardinalita je menší nebo nanejvýš stejná kardinalitě původní populace. Řešení z podpopulace Q se transformuje mutačním operátorem na populaci potomků Q'. Potomci z populace Q'se pak ohodnotí účelovými hodnotami f(α). Podpopulace Q a Q' jsou sjednoceny do společné množiny, obsahují všechny rodiče a potomky. Z této sjednocené populace vytvoříme novou populaci následovníků S, přičemž platí(Výraz 4).

$$|S| = |Q| = |Q'| \tag{4}$$

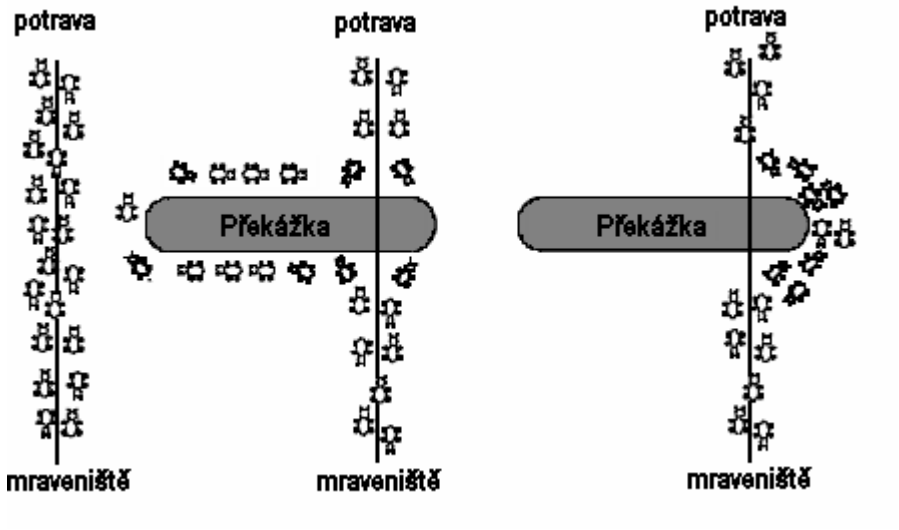
Tento výběr se realizuje pomocí operátoru selekce (turnaje) , jak je již naznačeno na (Obr.1). [1]



Obr. 1 Princip evolučního programování

1.4 Ant Colony Optimization (ACO)

Jde o algoritmus, jehož činnost je založena na chování mravenců v kolonii. Princip je následující. Nechť existuje zdroj mravenců (mraveniště) a cíl jejich snažení (potrava). Když jsou vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem. Tento efekt, kdy mravenci naleznou optimální cestu je dán faktem, že mravenci si svou cestu značí feromonem. Jeho intenzita pak ovlivňuje rozhodnutí mravence. Pokud dorazí k rozcestí dvou cest vedoucích ke stejnému cíli první mravenci, pak jejich rozhodnutí, po které cestě se vydají je náhodné viz (Obr.2). Ti, kteří zvolí kratší trasu, ji označí a při návratu jsou díky tomuto značení při rozhodování ovlivněni ve prospěch kratší cesty. Při návratu ji označí podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v jejich prospěch. Tyto principy jsou použity v ACO algoritmu. Feromon je zde zastoupen vahou, která je přiřazena dané cestě vedoucí k cíli. Tato váha umožňuje přidávat další „feromony“ od dalších mravenců. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy u jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálního extrému. ACO byl použit s úspěchem na optimalizační problémy jako problém obchodního cestujícího či návrh telekomunikačních sítí.[2]



Obr. 2 Nalezení optimální cesty mravence za potravou

1.5 Metoda imunologického systému

Tento svou podstatou nezvyklý algoritmus je založen na principech fungování imunologického systému v živých organismech. Pohlížíme na něj jako na multi-agentní systém, kde jednotlivé typy agentů musí řešit svou specifickou úlohu. Tito agenti mají různé úkoly, pravomoci a schopnost komunikovat s jinými agenty. Na základě této komunikace a určité „svobody“ rozhodování jednotlivých agentů vzniká hierarchická struktura, schopná řešit komplikované problémy. Jako příklad lze použít antivirovou ochranu pomocí této metody u velkých a rozlehlých počítačových systémů. [2]

1.6 Rojení částic

Algoritmus je založen na práci s populací jedinců, jejichž pozice v prostoru možných řešení je měněna pomocí tzv. rychlostního vektoru. Nedochází zde k vzájemnému ovlivňování, to je později odstraněno ve verzi s tzv. sousedstvím, kdy se definuje v populaci sousedství jedinec i, který zahrnuje sousední jedince v rozsahu $i \pm$ velikost sousedství. V rámci tohoto sousedství pak dochází k vzájemnému ovlivňování tak, že jedinci patřící do jednoho sousedství putují k nejhlubšímu extrému, který byl nalezen v tomto sousedství.

Algoritmus :

1) Z náhodně zvolených jedinců je vytvořena populace.
2) Přepočítání fitness pro každého jedince. Tato hodnota bude záviset na vzdálenosti od extrému.
3) Reprodukce populace je založena na tomto ohodnocení účelové funkce.
4) Jestliže hodnoty jsou si rovny, pak se proces zastaví. Jinak se vrací zpět na krok 2. [2]

1.7 Monte Carlo (Metropolisův algoritmus)

Metropolis a kolektiv navrhli metodu Monte Carlo, která simuluje evoluci systému tak, že generuje posloupnost stavů systému následujícím způsobem. Nechť je dán aktuální stav systému (určený polohou tělesa), potom se malá náhodná porucha generuje tak, že částice jsou jemně posunuté. Tato porucha musí být „symetrická“, tj. pravděpodobnost toho, že malou poruchou se stav A změní na stav B, musí být stejná jako při změně malou poruchou stavu B na stav A. Toto pravidlo akceptování porušeného stavu se nazývá Metropolitovo kritérium. Podle něj aplikováním velkého počtu poruch a jejich akceptováním do dalšího procesu s určitou pravděpodobností dostaneme systém v tepelné rovnováze. Proces změny stavu A na stav B je formálně reprezentovaný stochastickým operátorem. [1]

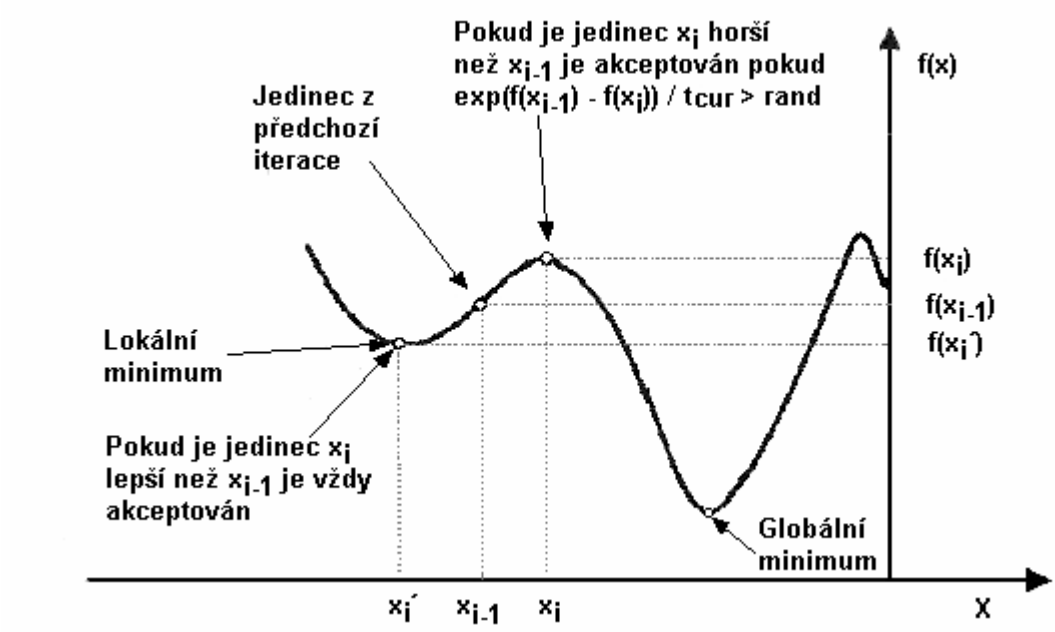
1.8 Metoda simulovaného žíhání

Simulované žíhání (Simulated Annealing) (SA) je metoda, která pro akceptaci nového řešení využívá náhody. Název metody je odvozen z představy simulování fyzikálních procesů probíhajících při odstraňování defektů krystalové mřížky kovu, které se projevují pnutím v materiálu. Při žíhání se kov zahřeje na tak vysokou teplotu, při které atomy v krystalové mřížce mohou překonat lokální energetické hladiny, a defekty krystalové mřížky mají velkou pravděpodobnost zániku. Po dosažení tohoto stavu se kov pomalu ochlazuje (žíhá), a tím se atomy dostanou do rovnovážných poloh s nejmenší energií. Při konečné teplotě žíhání (podstatně nižší, než byla počáteční) jsou všechny atomy kovu v rovnovážných polohách a těleso neobsahuje žádné vnitřní defekty ani pnutí. V simulovaném žíhání je krystal reprezentován jedincem x. Ke každému jedinci může být přiřazena funkční hodnota $f(x)$, která představuje energii krystalu. Analogií minimalizace energie krystalu je v metodě simulovaného žíhání minimalizace funkce $f(x)$ a pomalé ochlazováni představují postupné

iterace. V každé iteraci je původní jedinec x_{i-1} nahrazen novým, náhodně vygenerovaným jedincem x_i. Pravděpodobnost nahrazení uvádí výraz (5),

$$P(x_{i-1} \rightarrow x_i) = \left\{1, e\frac{f(x_{i-1}) - f(x_i)}{t_{cur}}\right\} \tag{5}$$

kde t_{cur} je parametr vyjadřující teplotu v daném kroku. Jestliže jedinec x_i má menší nebo stejnou funkční hodnotu jako původní jedinec x_{i-1}, je automaticky akceptován do další iterace. V opačném případě je pravděpodobnost akceptování jedince *xi* menší než jednotková, ale i v tomto případě má nový jedinec šanci postoupit do další iterace, viz (Obr. 3), kde *rand* je náhodně, s normálním rozložením, vygenerovaná konstanta.[1]



Obr. 3 Chování jedinců u simulovaného žíhání

1.8.1 Simulované žíhání s elitismem

V originální verzi simulovaného žíhání slouží výsledný stav z aplikace Metropolitova algoritmu jako počáteční stav následujícího Metropolitova algoritmu. Tento základní předpoklad simulovaného žíhání bude nyní modifikován tak, že Metropolitův algoritmus se inicializuje nejlepším řešením, které se získalo v průběhu předcházející metody (tj. pro epochy simulovaného žíhání s vyššími teplotami jako současná teplota). Tuto jednoduchou modifikaci simulovaného žíhání nezýváme simulované žíhání s elitismem. [1]

1.8.2 Paralelní simulované žíhání

Tato verze simulovaného žíhání se s úspěchem používá na řešení komplikovaných kombinatorických problémů, kde standardní verze není schopna poskytnout správné globální řešení. Základní idea paralelního simulovaného žíhání spočívá v současném aplikování simulovaného žíhání na množinu stavů x, x', x"…, které jsou „synchronizované“tak, že Metropolisovy algoritmy běžící nad nimi mají stejnou teplotu. Aby stavby mezi sebou integrovali, zavádí se operátor křížení podobným způsobem jako u genetických algoritmů.[1]

1.9 Genetický algoritmus

Genetické algoritmy (GA) vycházejí z Darwinovy teorie o vývoji druhů.
Zjednodušená biologická interpretace objasňuje GA následovně. Všechny živé organismy jsou složeny z buněk. Každá z buněk obsahuje jádro, které v rámci jednoho organismu obsahuje identické kopie chromosomu. Chromosomy se dále dělí na geny (podmnožiny chromosomu), popisující jednu vlastnost organismu. Při rozmnožování se páry chromosomu jednoho jedince zkříží a vytvoří tak nový chromosom. Nově vzniklý chromosom se podrobí mutaci, kdy se náhodně změní hodnoty genu. Schopnost organismu přežít v daném prostředí se označuje fitness. Popsaný mechanismus probíhá podle zákonů, které se GA snaží napodobovat. Způsoby tohoto provedení se mohou lišit.
V kontextu standartního genetického algoritmu (SGA) představuje chromosom jedno přípustné řešení daného problému, které označíme *x*. Existují dva způsoby kódování chromozomu – pracuje se s dekadickým, nebo binárním chromozomem.

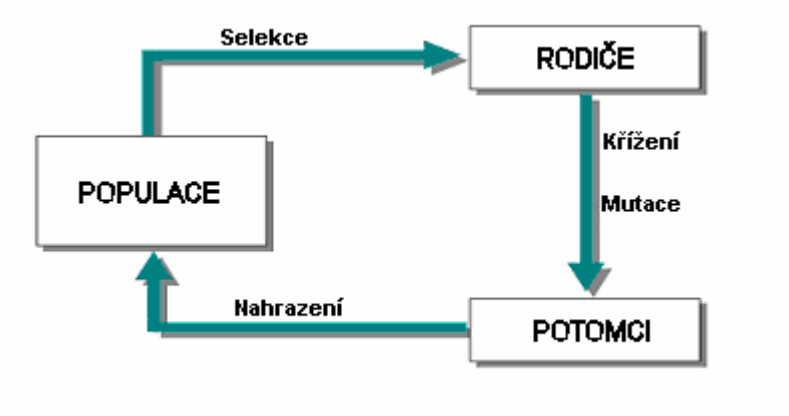
GA pracuje s množinou chromosomů a tato množina se nazývá populace. Velikost populace udává počet jejích chromozomů. Vývojový čas SGA běží v diskrétních krocích a v tomto časovém pohledu se hovoří o posloupnosti populací jako o generacích.

Mechanismus přirozeného výběru v přírodě, při kterém přežívají jen nejsilnější jedinci, je realizován operátorem selekce. Účelem selekce je vybrat do další generace pouze kvalitní jedince (odfiltrování horších jedinců v příští generaci). Selekcí vznikne nová populace-rodiče (Obr.4). Rozmnožování jedinců je simulováno změnovými operátory křížení a mutace, které jsou aplikovány na jedince vybrané selekcí.

Proces křížení je simulován tak, že se nad každým selektovaným jedincem provede náhodný pokus a příslušný jedinec se s pravděpodobností P_{cros} stane kandidátem na reprodukci.

Vznikne tak množina jedinců, z nichž se vybírají dvojice buď náhodně, nebo se za pár prohlásí po sobě jdoucí jedinci tak, jak byli do této množiny zařazeni. Je-li počet jedinců lichý, jeden kandidát se vyloučí. Bod křížení je také obvykle náhodný.

Po operátoru klížení následuje simulace biologické mutace a jejím hlavním úkolem je zajistit, aby populace nedegradovala v identické chromozomy. Mutace náhodně změní vybraný bit chromozomu z 0 na 1 a naopak. Pravděpodobnost mutace musí být nenulová, aby byly zaručeny změny, ale nesmí být příliš vysoká, aby byl zachován prvek dědičnosti. [3]



Obr. 4 Princip genetického programování

1.10 Evoluční strategie

Patří historicky mezi první úspěšné stochastické algoritmy. Na rozdíl od většiny evolučních algoritmů není evoluční strategie založena na binární reprezentaci jedince, ale manipuluje přímo s reálnou reprezentací proměnných.

Základem evoluční strategie je následující předpis (6), který „mutuje“ aktuální řešení x na nové řešení x'.

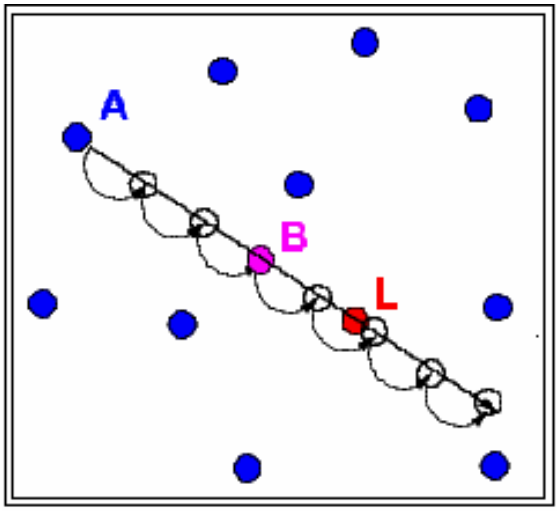
$$x'=x+N(0,\delta), \tag{6}$$

kde $N(0,\sigma)$ je vektor nezávislých náhodných čísel s nulovou střední hodnotou a standardní odchylkou σ. Jednotlivé čísla jsou náhodně rozloženy podle Gaussovy distribuce. Menší čísla se objevují s větší pravděpodobností. Standardní odchylka je mírou rozptylu generovaných hodnot. [1]

1.11 SOMA

Optimalizační algoritmus SOMA (samoorganizující se migrační algoritmus)byl vyvinut Doc. Ivanem Zelinkou, PhD. Využívá ke své funkci některé podobné přístupy jako genetické algoritmy. Při běhu algoritmu SOMA se však nevytvářejí noví jedinci, dochází jen k jejich přemisťování v stavovém prostoru. Samoorganizace vzniká vzájemným ovlivňováním jedinců v populaci při pohybu.

Na začátku se náhodně vygeneruje určitý počet jedinců. Vyhodnotí se jejich funkce vhodnosti (fitness) a určí se nejlepší jedinec – leader. Potom se vytvoří další populace, ve které se každý jedinec kromě nejlepšího pokouší najít si vhodnější polohu v prostoru, a to tak, že se skoky o určité délce pohybuje po přímce směrem k leadrovi. Jedinec na této cestě může udělat menší odbočky (ty jsou způsobeny mutací), nepohybuje se přesně po přímce, ale místy se od ní trochu odchýlí. Toto zvyšuje diverzitu v populaci. Když jedinec přejde celou trasu, v další populaci se usadí na místě, které bylo na celé trase nejvýhodnější.



Obr. 5 Pohyb jedince u SOMA algoritmu

Variace algoritmu SOMA:

- All To One (všichni k jednomu). Tato variace byla popsána výše a je možné ji považovat ze základní variaci algoritmu SOMA.
- All To All (všichni ke všem). Zde neexistuje leader. Každý jedinec migruje ke všem ostatním.
- All To One Rand (všichni k náhodnému). Každý jedinec migruje jen k jednomu náhodně zvolenému jedinci.

- All To All Adaptive (adaptivně všichni ke všem). Tato variace je založená na strategii All To All s rozdílem, že aktuálně migrující jedinec se nepřesouvá do nové pozice až po migraci směrem ke všem jedincům, ale bezprostředně po dokončení své migrace ke každém z jedinců a v migraci k dalším jedincům pokračuje už z této nové polohy. [2]

1.12 Prohledávání do šířky (breadth–first seach)

Algoritmus začíná ve startovním políčku. V prvním cyklu prohledá jeho sousedy, poté prohledá pozice vzdálené 2 políčka od startu, v dalším cyklu pozice vzdálené 3 políčka a tak pokračuje, dokud nedojde v k-tém cyklu k cíli nebo neprobral všechna dostupná políčka. Nalezená cesta bude mít délku k. Pokud se v průběhu algoritmu zaznamenávala u každého políčka pozice, odkud bylo na něj vstoupeno, je možné zpětně zkonstruovat cestu z cíle do startu. V algoritmu je ukládána hodnota předchozího políčka do položky „předchozí“ vrcholu. Tento způsob je použit i u dalších algoritmů.

Algoritmus po mapě postupuje jako vlna.Tento algoritmus dovede korektně pracovat pouze s jednoduchou funkcí w, která přiřazuje každému průchodnému políčku stejnou hodnotu (např. 1). Nebere tedy v potaz ohodnocení políček. Proto také počet prozkoumaných políček je vysoký. Tento nedostatek se snaží řešit níže uvedené algoritmy s využitím heuristiky. [4]

1.13 Prohledávání do hloubky (depth-first search)

Prohledávání do hloubky je duálním algoritmem k prohledávání do šířky. Místo prohledání všech sousedů před prohledáním nástupců, hledá opačně. Vždy se nejdříve pustí do hloubky a pak se vrací a prohledává sousedy. Dále je nutné určit zarážku, na které se algoritmus začne vracet. Jinak by se totiž algoritmus vnořoval stále hlouběji, dokud by nepřetekl zásobník nebo by nebyl nalezen cíl. Zarážkou bude pevně určená délka cesty. V každém kroku je z možných vrcholů pro vnoření, vybrán nejdříve ten, o kterém heuristika říká, “že je nejvhodnější”. Pokud je heuristika dostatečně kvalitní, bude cesta natahována prioritně směrem k cíli. V ideálním případě, kdy heuristika nejkratší cestu do cíle odhaduje přesně, poběží algoritmus najisto a najde cíl bez návratu.

Prohledávání do hloubky může pracovat s vzdáleností jako počtem políček cesty. Podle toho musí být použita hloubka zarážky. Prohledávání do hloubky s uvedenými vylepšeními v průměrném případě nachází cestu poměrně rychle. Negarantuje však, že nalezená cesta je nejkratší možná. [4]

1.14 Best-first search

Vstupem algoritmu je orientovaný graf G, nezáporné ohodnocení hran w (určuje cenu průchodu, postih) a vrchol s (start). Výstupem algoritmu bude předpokládaná vzdálenost do cíle, tedy podle hodnoty heuristiky. Tento algoritmus tedy z principu nerozlišuje ohodnocení funkce w, pokud není započítána v použité heuristice. Algoritmus tedy prozkoumá nejdříve políčka, o kterých heuristika říká, že jsou blíže cíli. Tímto způsobem se dravě dostává rychle směrem k cíli, ale nebere v potaz ohodnocení terénu. Algoritmus nachází nejkratší cesty pouze v případě, že je heuristika dokonalá - tedy odhaduje délku nejkratší cesty přesně. [4]

1.15 Greedy algoritmus

Je to jedna z velmi rozšířených metod, která se většinou používá při řešení optimalizačních úloh, které často můžeme charakterizovat následovně:

1) vstupem je n-prvková množina
2) cílem je vytvořit podmnožinu (posloupnost), která splňuje podmínky odrážející charakter problému. Každé řešení splňující toto ohraničení se nazývá přípustné řešení.
3) Nejvýše je posazena účelová funkce definovaná na množině přípustných řešení.
4) Výstupem je to přípustné řešení, které optimalizuje účelovou funkci. Takové řešení nazýváme optimální řešení.

Snažíme se nalézt rychlý algoritmus, který generuje přímo optimální řešení. Greedy metoda vede k algoritmu, který pracuje v iteracích. V každé iteraci se vybere jeden kandidát na zařazení do optimálního řešení. Jestliže kandidát nemůže být do řešení zařazen, definitivně se vyřadí z množiny kandidátů. [5]

Příklad**:**Je dána množina hodnot mincí a suma, kterou je třeba zaplatit (např. 23). Hodnotou účelové funkce je počet mincí, které se snažíme minimalizovat. Greedy metoda vede k následovnímu postupu:

Začni s bankovkami s maximální možnou hodnotou a plať, jak se dá. Potom vezmi bankovky s menší hodnotou a plať, jak se dá,... Jestliže vezmeme jako základní množinu mincí {1,2,5,10}, vede tato metoda k optimálnímu řešení , tedy 10, 10 2, 1.

1.16 Scatter search (rozptylové prohledávání)

Tento optimalizační algoritmus se svou podstatou liší od standardních evolučních algoritmů tím, že je dost podobný algoritmu Tabu Search. Je to vektorově orientovaný algoritmus, který má za úkol generovat nové vektory (řešení) na základě pomocných heuristických technik.

Při startu se vychází z řešení získaných pomocí vhodné heuristické techniky. Poté jsou generována nová řešení na základě podmnožiny nejlepších řešení ze startu. Z těchto nově nalezených řešení se opět vybere množina těch nejlepších a celý proces se opakuje. Tento algoritmus byl použit k řešení problémů jako je řízení dopravy, učení neuronové sítě či optimalizaci bez omezení a v mnoha dalších problémech. [2]

2 PŘEHLED OPERÁTORŮ A JEJICH POPIS

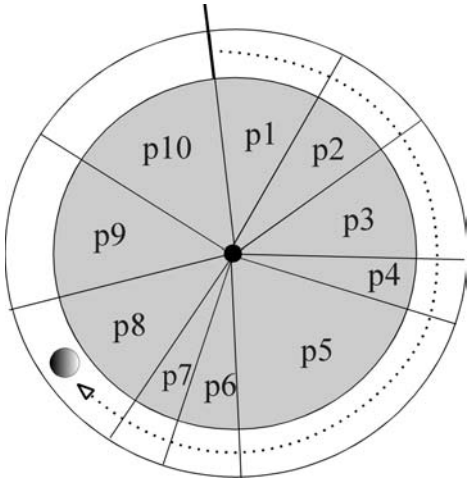
V předchozí kapitole byl uveden seznam evolučních algoritmů. V této části jsou popsány operátory, pomocí níž algoritmy fungují. U každého operátoru jsou zároveň uvedeny algoritmy, které tento operátor využívají.

2.1 Selekce

Vstupním parametrem je zde populace. Dochází k výběrů nejlepších jedinců k vytvoření další populace podle jejich účelové funkce. Do další populace postupují jen ti jedinci, kteří mají vhodné ohodnocení této funkce. Ti se stávají rodiči a mohou tvořit nové potomky. Tento operátor nevytváří nové řešení, jen vybírá relativně dobré řešení z populace a zavrhuje nevhodná.

Existují různé druhy selekcí:

1) *ruleta* – ta měla vytvořeny pozice, jejichž pravděpodobnosti byly úměrné vhodnosti (Obr.6), a poté probíhal náhodný výběr. Nevýhodou je, že pokud populace obsahuje řešení mající výrazně lepší fitness než ostatní, může toto řešení zabírat podstatnou část na ruletě, a tím se snižuje různorodost populace.Tento nedostatek může odstranit omezení, které umožňuje, že každé řešení fitness je lineárně rozloženo mezi horní a dolní hranici označených polí v ruletě. Později byla tato metoda zamítnuta, protože zde hrála velkou roli náhoda.



Obr. 6 Ruleta

2) *matice* – první políčka byla nejprve vyplněna rodiči s nejvyšší selekcí, např. jedinci se selecí 3 budou ve 3 políčkách atd., poté rodiči s menší selekcí až po jedince se

selekcí menší jak 1, kteří jsou do pole vybráni náhodně. Podle toho se pak budou pářením vytvářet noví potomci.

3) *turnaj* – jedná se zápas mezi specifickým počtem rodičů podle jejich fitness. V turnaji o q řešení, je nejlepší řešení vybráno buď deterministicky nebo pravděpodobnostně. Po odehrání turnaje existují 2 volby – buď všechny zúčastněné řešení jsou vyměněny do populace pro další turnaj nebo nejsou nahrazeny až po do určitého počtu odehraných turnajů. Nejjednodušší forma (zvaná binární turnajový výběr), máme 2 řešení a lepší je vybráno. Výhodou této metody je, že může ovládat maximum i minimum bez strukturální změny ve fitness.
4) *klasifikace* – tento operátor je podobný ruletě, kromě toho že řešení jsou klasifikována podle postupného nebo sestupného pořadí jejich fitness v závislosti na tom, zda jde o hledání maxima nebo minima. Každému řešení je přiřazena fitness založená na pozici jedince v populaci. Pak jsou vybírány pomocí pravděpodobnosti, která je přiřazena podle jejich fitness.(tzn. čím lepší ohodnocení účelové funkce, tím vyšší pravděpodobnost výběru). Rozlišujeme lineární a nelineární. Lineární klasifikace přiřazuje pravděpodobnost každému jedinci. Ta je úměrná jeho individuálnímu ohodnocení (klasifikace nejmenšího vhodného řešení je označena 0, nejvíce vhodné je definováno jako 1). Lineární klasifikace může být zrealizována specifickým parametrem β_{RANK}, který udává očekávané množství potomků přidělených na nejlepší hodnotu během jedné generace.Nelineární klasifikace přiděluje pravděpodobnost výběru na základě pozice každého jedince, ale tento výběr není úměrný jejich pozici.
5) *Boltzmanův operátor* – modifikovaná fitness je přiřazena každému řešení (jedinci) na základě Boltzmanovy rozdělení pravděpodobnosti (7),

$$F_i=1/(1+\exp(F_i/T) \qquad (7)$$

kde T je parametr odpovídající teplotě v Boltzmanově rozdělení. Parametr je vracen v předdefinovaném chování v postupných iterací. Výběr je uskutečňován na základě pravděpodobnostního rozdělení. Na začátku má T velké hodnoty a téměř jakékoliv řešení má stejnou pravděpodobnost výběru, ale v každé iteraci se T stává menší, a pak jsou vybírány jen dobré řešení.

6) *„Soft brood“* – uskutečňuje se zde turnaj mezi členy potomstva 2 rodičů. Vítěz turnaje se považuje za potomka přispívajícího k dalšímu páření. Tento operátor se za-

myšlí nad ochranou rekombinačního operátoru mizejícího potomstva. Vybírá takového potomka, na němž pak testuje jeho životaschopnost předtím, než ho umístí do soutěže

7) *Rozklad* – může být použit proti výběru jedinců s nízkými hodnotami. Kuo a Hwang představili ohodnocení účelové funkce jako (8),

$$u(x)=|f(x)-f(t)| \quad (8)$$

kde f(x) představuje objektivní hodnotu řešení x a f(t) všechna řešení v čase t. Fitness se pak zvyšuje se vzdáleností od současných řešení.

8) *Konkurence* – je realizována tak, že ohodnocení jedince je určeno, jak jeho vzájemným působením s ostatními členy populace nebo členy, kteří se spolu vyvíjí, tak i vývoji sebe samého.[6]

Algoritmy požívající selekce:

- ✓ Genetický algoritmus
- ✓ Evoluční strategie
- ✓ Evoluční programování
- ✓ Greedy algoritmus
- ✓ Scatter Search
- ✓ SOMA
- ✓ Diferenciální evoluce
- ✓ Umělý imunitní systém
- ✓ Mravenčí kolonie (ACO)
- ✓ Tabu Search

2.2 Mutace

Vstupním parametrem je zde jedinec. Operátor stochasticky transformuje binární vektor na nový binární vektor, přičemž stochastičnost tohoto procesu je určena pravděpodobností P_{MUT}. V každém kroku jsou mutace vykonávány náhodně. Mutace přináší novou informaci.

Obecně jde o proces, při kterém dochází k náhodné změně některých vlastností – parametrů jedince.

Příklad:Nechť (00**1**100**0**1) je bitový řetězec. V procesu mutací v náhodně vybraných polohách 3 a 7 se mění komponenty. Dostaneme nového jedince (00**0**100**1**0).

Mutace založená na pozici – výběr 2 prvků v řetězci a přesunutí druhého za první.

Mutace založená na pořadí – mohou být vybrány 2 prvky a vymění si své pořadí v řetězci

„Scramble“ mutace - vybírá díly z permutací a náhodně je seskupuje

U umělého imunitního systému se používá 3 druhů mutací:

1) jednotlivá bitová změna- vysvětleno výše

2) genová duplikace- zvyšuje paměť jedince

3) rozštěpená mutace – vytváří opak genové mutace rozdělením geonomu na 2 části, náhodně vybírá jednu z nich. [1]

Algoritmy požívající mutace:

- ✓ Horolezecký algoritmus
- ✓ Evoluční programování
- ✓ Genetický algoritmus
- ✓ Simulované žíhání
- ✓ SOMA*
- ✓ Diferenciální evoluce**
- ✓ Umělý imunitní systém

2.2.1 Mutace u SOMA algoritmu

V případě SOMA je mutace přejmenována na *perturbaci.* Při pohybu jedinců skrz prostor možných řešení je jejich pohyb náhodně rušen (perturbován).

Jak silná bude perturbace záleží na nastavení parametru PRT, pomocí něhož se generuje perturbační vektor. PRT vektor se generuje pro každého jedince zvlášť a je platný jen pro aktuální běh aktivního jedince.

Generování perturbačního vektoru probíhá tak, že se pro každý jeho parametr generuje náhodné číslo v intervalu $\langle 0,1 \rangle$ a porovnává se s PRT parametrem. Je-li menší, jak PRT parametr, pak je danému prvku perturbačního vektoru přiřazena 1, je-li větší pak 0.[2]

2.2.2 Mutace u diferenciální evoluce

K vytvoření dalšího potomka je potřeba ne 2 , ale 4 rodičů. Pro každého jedince jsou náhodně vybráni 3 další nestejní jedinci z populace. Pomocí nich se pak tvoří tzv. šumový vektor, který je mutací kombinace oněch 3 náhodně vybraných rodičů.

Konkrétně je mutace provedena tak, že se rozdíl dvou prvních náhodně vybraných rodičů vynásobí mutační konstantou a výsledný vektor se přičte ke zbývajícímu třetímu. [2]

2.3 Křížení

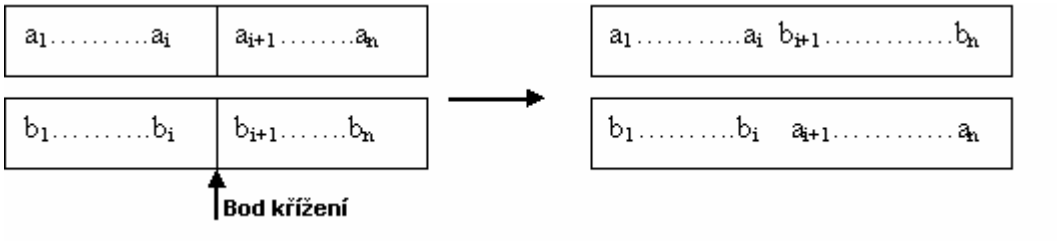
Vstupním parametrem je zde jedinec. Obecně se jedná o výměnu informace mezi 2 jedinci. Je to zobrazení, který dvojici jedinců přiřadí 2 nové jedince se stejnou délkou jako původní.*

Celý proces křížení je realizován tak, že nad každým vybraným jedincem se provede jednoduchý náhodný pokus, a příslušný jedinec se s určitou pravděpodobností stane kandidátem na páření. Vznikne tak množina jedinců, z nichž se vybírají dvojice buď náhodně nebo se za pár prohlásí po sobě jdoucí jedinci tak, jak byli do této množiny zařazeny. [1]

1-bodové křížení

Náhodně se vygeneruje bod křížení a jedinci si vymění své části za tímto bodem viz. (Obr.7).

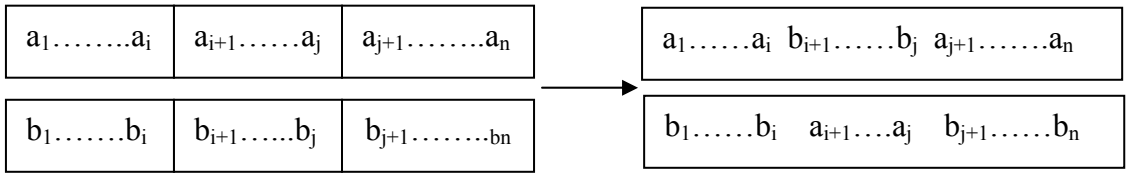
* u algoritmu umělého imunitního systému je křížení vykonáváno u jedinců opačného pohlaví



Obr. 7 Princip jednobodového křížení

2-bodové křížení

Náhodně se vygenerují 2 různé body křížení a jedinci si vymění části mezi nimi viz. (Obr.8).



Obr. 8 Princip dvoubodového křížení

Jednotné křížení

Množství křížících bodů není pevně definováno. Místo toho se rozhoduje vložení křížícího bodu v každém řetězci nezávisle.

Přerušované křížení

Je zde použit binární řetězec „přerušovaného“ označení k signalizaci místa křížícího bodu pro vícenásobné křížení. Tato informace je přidána chromozomu tak, že množství a umístění křížících bodů bude vycházet z genetického prohledávání řetězce.

Míchané křížení

Symboly v řetězci rodičů jsou míchány pomocí permutačního operátoru ještě před samotným křížením.[6]

Příklad: 0010|011=19 0010|001=17

* →

1111|001=121 1111|011=123

Algoritmy používající křížení:

- ✓ Genetický algoritmus[***]
- ✓ Genetické programování

- ✓ Paralelní simulované žíhání
- ✓ SOMA*
- ✓ Diferenciální evoluce**
- ✓ Umělý imunitní systém

2.3.1 Křížení u SOMA algoritmu

Putování jedinců po dané hyperploše. Při tomto pohybu si každý jedinec pamatuje souřadnice pozice, na níž nalezl nejlepší řešení v rámci své cesty, a toto řešení pak postupuje do dalších migračních kol. [2]

2.3.2 Křížení u diferenciální evoluce

Křížení nastává až po mutačním procesu, zatímco např. u genetických algoritmů je první tvořen potomek křížením a teprve tak dochází k jeho mutaci. Proces křížení u diferenciální evoluce znamená, že se ze čtvrtého doposud nepoužitého rodiče a šumového vektoru vytvoří nový jedinec, tzv. zkušební vektor. [2]

2.3.3 Křížení u genetického algoritmu

Proces, ve kterém 2 různí rodiče jsou opakovaně vybíráni ze skupiny rodičů k výměně informací mezi nimi a generují pak 2 nové jedince (potomstvo). Pak je náhodně vybrán bod křížení rodičů a ti si pak vymění své části za bodem křížení. Cílem křížení je, aby potomci nesli vlastnosti obou rodičů. [1]

2.4 Reprodukce

Vstupním parametrem je jedinec. Tvoří se pomocí operátorů mutace a křížení.
Nechť existují a, $b \in P$ jsou 2 jedinci z populace P. Pomocí tohoto operátoru sestrojíme z těchto 2 rodičů 2 nové potomky. Je to proces, ve kterém jsou jedinci náhodně vybráni s pravděpodobností úměrnou jejich fitness a stávají se rodiči.
Reprodukce u algoritmu umělého imunitního systému je tvořena pomocí klonování s nízkou mutační pravděpodobností. Genetický materiál je mezi agenty vyměněn pářením. Rychlost reprodukce ovlivňuje fitness agentů. [1]

Algoritmy používající reprodukci:

- ✓ Genetický algoritmus

✓ Umělý imunitní systém

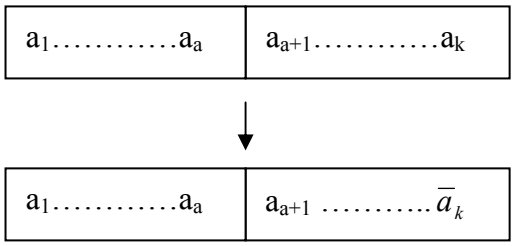
2.5 Inverze

Vstupním parametrem je jedinec. Inverzní operátor změní binární řetězec na jiný binární řetězec. Tato operace je užitečná především u binární reprezentace reálných čísel. V genetickém algoritmu slouží pro odstranění Hammingovy bariéry.

Pro náhodně vybraný bod inverze a vykonáme inverzi binárního vektoru tak, že od bodu a+1 až do konce binárního vektoru nahradíme jeho složky komplementami (9).

$$\overline{a} = 1 - a_i \tag{9}$$

Příklad: Máme čísla 0000 1111 a 0001 0000 sousedící v reálné reprezentaci, ale na přeměnu jednoho v druhé by bylo potřeba pěti překlopení bitů. Proto se zavádí operace, která od daného bitu překlopí všechny následující bity. Tak by se z prvního řetězce dal vyrobit druhý překlopením posledních pěti bitů. Tedy inverze od čtvrtého bitu (0000 1111)=0001 0000. [1]



Obr. 9 Princip inverzního operátoru

Algoritmy používající inverzi:

✓ Genetický algoritmus

2.6 Stochastický operátor

Vstupním parametrem je jedinec. Nechť je stav systému určený stavovou proměnnou x (obecně vektor obsahující mnoho nezávislých reálných proměnných) a energií f(x). Proces porušení stavu x na jiný stav x´ je formálně reprezentován stochastickým operátorem. Stochastický charakter tohoto operátoru spočívá v náhodné změněn stavu x na stav x´.[1]

Algoritmy používající stochastický operátor

✓ Simulované žíhání s elitismem

- ✓ Monte Carlo – Metropolitův algoritmus

2.7 Rychlost

Vstupním parametrem je jedinec.Na základě tohoto operátoru se mění částice svou pozici v prostoru, a tím dochází k nalezení extrému. Je dán rovnicemi (10) a (11).

v[]=v[]+c1*rand()*(pbest[]-present[])+c2*rand()*gbest[]-present[]) (10)

present[]=persent[]+v[], (11)

Kde

v[] – rychlost částice

persent[] – současné řešení

rand() – náhodné číslo mezi (0,1)

c1,c2 jsou učící faktory, obvykle c1=c2=2

pbest – nejlepší řešení jedné částice

gbest- globální nejlepší řešení všech částic. [7]

Algoritmy používající operátoru rychlost:

- ✓ Rojení částic

2.8 Feromonová matice

Feromonová matice (12) se využívá ke konstrukci potenciálních vhodných řešení. Počáteční hodnota τ je sada (13). Zároveň využívá heuristické informace(14).

$$\tau = \{\tau_{ij}\} \tag{12}$$

$$\tau_{ij}=\tau_{init} \ \forall (i,j), \text{ kde } \tau_{init}>0 \tag{13}$$

$$\eta_{ij} = \frac{1}{d(i,j)} \tag{14}$$

Parametry α a β definují relativní vliv mezi heuristickou funkcí a feromonovou úrovní.

Algoritmus využívající tohoto postupu se používá nejčastěji k řešení problému obchodního cestujícího (TSP). Zatímco navštívená města i, N_i reprezentuje řadu měst, které ještě nebyly navštíveny a pravděpodobnost výběru města j je definována jako (15)

$$P_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{\forall h \in N_i} \tau_{ih}^{\alpha} * \eta_{ih}^{\beta}} \tag{15}$$

když $j \in N_i$.

Ve všech generacích algoritmů, každý agent pak užívá tento vzorec pro svou kompletní cestu. Startuje v náhodném bodě. Postupné mizení feromonů je aplikováno pro všechna (i,j) podle rovnice (16) kde parametr $\rho \in (0,1]$ určuje rychlost mizení feromonů. Vzhledem k elitismu, nejlepší řešení nalezeného doposud je aktualizace τ podle rovnice (17),

$$\tau_{ij}=(1-\rho)* \tau_{ij,} \tag{16}$$

$$\tau_{ij} = \tau_{ij} + \Delta\tau \tag{17}$$

kde $\Delta\tau=1/l(r_{best})$, když $(i,j) \in r_{best}$ a $\Delta\tau = 0$, když $(i,j) \notin r_{best}$. [8]

Algoritmus využívající feromonovou matici:

- ✓ Mravenčí kolonie (ACO)

2.9 Další operátory

2.9.1 Průsečík

Tento operátor vybírá společné rysy rodičů, které pak slouží k dalšímu použití jiných operátorů, např. rekombinaci.[6]

2.9.2 Sjednocení

Každá změna je převedena do binární matice. Tato změna představuje vznik nových potomků z rodičů, kteří se pak logicky stávají také rodiči.[6]

2.9.3 Genové mazání

Genotyp (18) se skládá z genů $x_i \in G$, i=1,….,n (n představuje aktuální délku genotypu) z genového prostoru G. Operátor mazání **del(a)** lze formovat jako funkci přeměny daného

jedince (19) vymazáním genu x_i. (20) je individuální prostor, kde A_s je strategický parametr prostoru, který závisí na aplikaci konkrétního EA.

$$x = (x_1,\ldots,x_n) \in X, \tag{18}$$

$$a = (x,s) \in I \tag{19}$$

$$I = X * A_s \tag{20}$$

Příklad: del:I→I, s del(a) = del($x_1,\ldots,x_{i-1},\mathbf{x_i},x_{i+1},\ldots x_n,s$) = ($x_1,\ldots,x_{i-1},\mathbf{x_{i+1}},\ldots x_n,s$) = a

Operátor mazání je použit na základě pravděpodobnosti $p_{det} \in (0,1)$. Pozice i fixuje gen, který má být vymazán. Mazání nastává až po rekombinaci. [6]

Algoritmy používající genové mazání:

- ✓ Genetický algoritmus a jeho aplikace
- ✓ Evoluční programování

2.9.4 Genová duplikace

Princip je podobný jako u genového mazání

Příklad:Dup: I→I s dup(a) = dup ($x_1,\ldots,x_i,,\ldots x_n,s$) = ($x_1,\ldots,x_i, x'_i,\ldots x_n,s$) = a'

Operátor duplikace je použit na základě pravděpodobnosti $p_{dup} \in (0,1)$. Pozice i fixuje fen, který má být duplikován. Genovou duplikace můžeme rozdělit do 3 fází:

1) Duplikace: gen x'_i je duplikátem x_i, pro (21)

$$a' = (x_1,\ldots,x_i, x'_i,\ldots x_n,s) \tag{21}$$

2) Inicializace: inicializace nového genu x'_i je vygenerována pomocí aktuální hodnoty x_i a x_{i+1}
3) Přidání: přidání x'_i je inicializováno náhodně

Oba operátory zajišťují vyšší výkon než ten, který je možný s pevným kódováním, a uplatňuje se u takových algoritmů, u kterých jsou relativně pomalé přeskupení operátorů, jako např. u inverze.[6]

Algoritmy používající genovou duplikaci:

- ✓ Genetický algoritmus a jeho aplikace
- ✓ Evoluční programování

- ✓ Umělý imunitní systém

3 ANALYTICKÉ PROGRAMOVÁNÍ

Analytické programování (AP) je nová metoda symbolické regrese, kterému předchází Gramatická evoluce (GE) a Genetické programování (GP).

Autor analytického programování vybral jeho jméno, protože jednoduše signalizovalo využití EA pro analytické řešení syntézy (tj. symbolickou regresi). Jak již bylo řečeno, je možné využít v AP téměř všechny evoluční algoritmy (a to i experimentálně testované). Použitím každého nového algoritmu by vzniklo, dle užívaného přístupu nové jméno např. SOMA programování, DE programování, SA programování atd., což by jistě bylo matoucí.

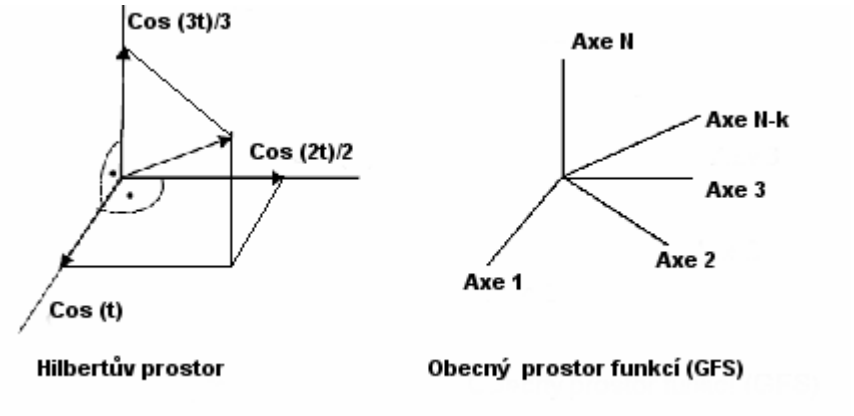
3.1 Hilbertův funkční prostor a obecný prostor funkcí

Analytické programování bylo inspirováno GP a číslicovými metodami v Hilbertových funkčních prostorech. Princip AP je někde mezi těmito dvěma filozofiemi. Z GP přebírá představu evolučního vytvoření symbolických řešení, zatímco z Hilbertových prostorů přijímá vystavění výsledné funkce prostřednictvím optimalizace (obvykle vytvořeného takovými metodami jako jsou Ritzova nebo Galerkinova) [9].

Analytické programování pracuje se složitým funkčním prostorem, který používá zvláštní způsob rozvoje požadovaného řešení, a díky svým vlastnostem ho nazýváme obecným prostorem funkcí (GFS převzato z anglického General Function Space). Osy GFS si nejsou navzájem kolmé a nepředstavují jen jedinou funkci jako v případě Hilbertova prostoru, ale každý bod (celé číslo) na ose reprezentuje funkci, operátor nebo konstantu.

GFS je více obecnější než Hilbertův prostor, který je široce využívaný ve fyzice, aplikované matematice atd. Hilbertův prostor, je oproti GFS, definovaný obvykle jako prostor s vlastnostmi: úplný, kompaktní, orthogonální a ortonormální.

Tyto požadavky dovolí řešit různé problémy obvykle jednoduchým způsobem. Z geometrického hlediska, může být Hilbertův prostor zobrazen ze vzájemně se sestávajících kolmých os, kde každá z os představuje jednu základní, obvykle periodickou funkci jako je sinus nebo kosinus (Obr.10). Pozice každého bodu v takovém funkčním prostoru je popsána souřadnicemi všech os. Složení všech složek vektoru (všech os) dává výslednou funkci, tedy vektor vycházející z nulových souřadnic k danému bodu v Hilbertově prostoru. Pro řešení daného problému se nejprve vybere vhodný funkční základ, z tohoto funkčního základu se vytvoří funkcionál a pomocí vhodných metod jsou odhadnuty neznámé parametry.



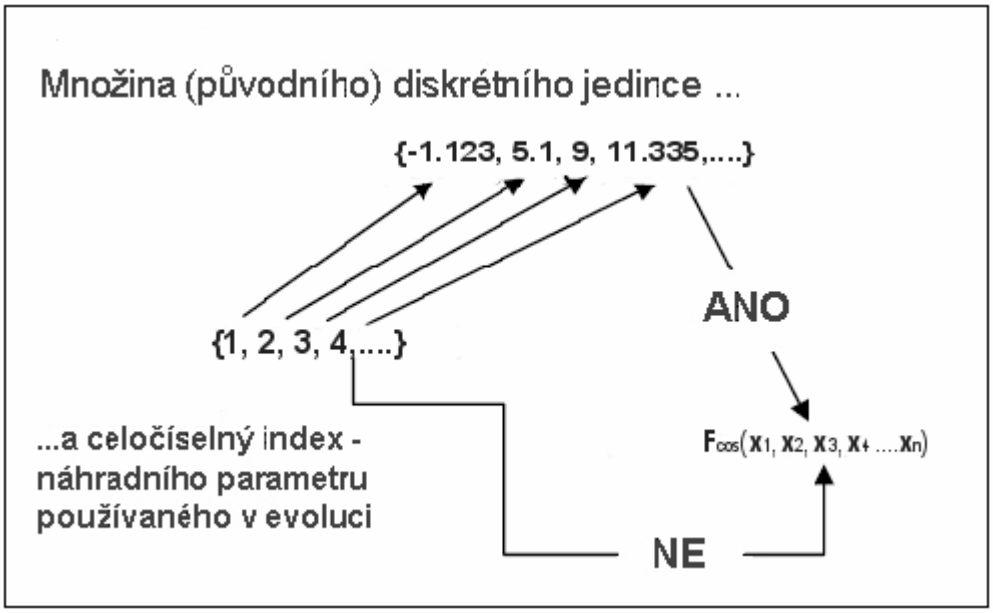
Obr. 10 Vzorové příklady funkčních prostorů

3.2 Manipulace s množinou diskrétních hodnot

Hlavní princip AP je založen na práci s množinou diskrétních hodnot (DSH převzato z anglického - Discrete Set Handling). DSH bylo vyvinuto při práci se SOMA a používá se v optimalizacích evolučními algoritmy.

3.2.1 Postup při práci s množinou diskrétních čísel v EA

Při množině diskrétních hodnot jako je např. (-1, -2.5, 20, 3, -5.68, 254.3569…), která představuje parametry jedinců, se vytvoří množina celočíselných indexů o stejné velikosti, nabývající hodnot (1, 2, 3, 4, 5, 6 …). Tyto indexy nahradí diskrétní parametry jedince a pracuje se s nimi jakoby s normálními celočíselnými parametry, až na to, že při ohodnocení účelové funkce se nedosadí za dané argumenty aktuální celočíselné hodnoty, ale hodnoty z diskrétní množiny, na které tyto celočíselné parametry, coby index, ukazuje (Obr.11.)



Obr. 11 Manipulace s množinou diskrétních hodnot

Díky tomuto principu může být v AP uskutečněno využívání téměř každého evolučního algoritmu.

3.2.2 Postup při práci s množinou diskrétních čísel v AP

Tak jako GP nebo také GE, tak i AP je založeno na množině matematických objektů, které představují funkce, operátory a tzv. terminály (obvykle konstanty nebo nezávislé proměnné) například:

- funkce: Sin, Tan, And, Or
- operátory: +, -, *, /, dt,…
- terminály: 2.73, 1.75, t, a,…

Všechny tyto matematické objekty tvoří množinu funkcí s různým počtem argumentů nazvanou GFS, ze které AP zkouší syntetizovat vhodné řešení.

Struktura GFS je tvořena podmnožinami z funkcí podle počtu argumentů viz (Obr.12). Například GFS_{all} je soubor obsahující množinu všech funkcí, operátorů a terminálů, GFS_{3srg} je podmnožina s funkcemi obsahující jen 3 argumenty, GFS_{0arg} obsahuje pouze terminály atd.

Obsah GFS je dán uživatelem.

GFS_{all} = {+,-, /,^, Sin, Cos, Tan, t, x, Mod,....}

GFS_{3arg} ={User_Function, Beta....}

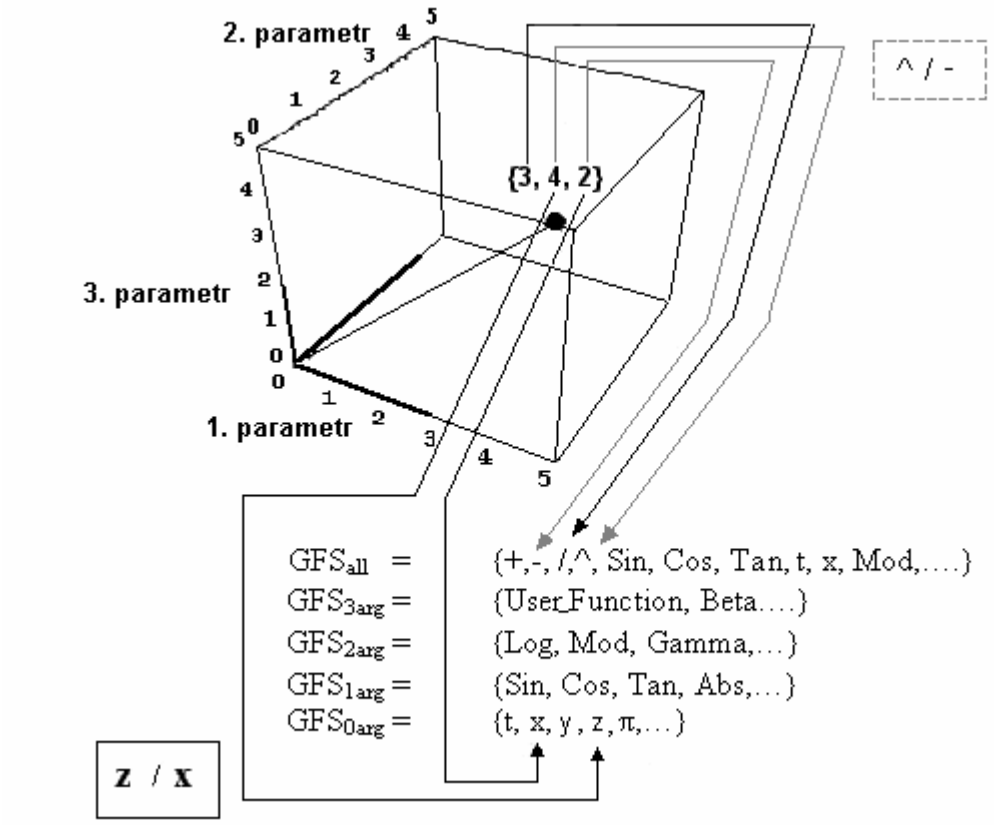
GFS_{2arg} = {Log, Mod, Gamma,...}

GFS_{1arg} = {Sin, Cos, Tan, Abs,...}

GFS_{0arg} = {t, x, x, π,...}

Obr. 12 Princip struktury GFS

V AP se jedinci skládají z nenumerických výrazů (operátorů, funkcí), které jsou v evolučním procesu reprezentovány jejich celočíselnými indexy. Tento index pak slouží jako ukazatel do souboru výrazů, který využívá AP k syntéze výsledného cílového programu pro vyhodnocení účelové funkce. Příklad budující funkce z GFS prostřednictvím AP je zobrazen na (Obr. 13). Pro 3-parametrového jedince si můžeme GFS představit jako krychli, kde lemy jsou funkční základny.[10]



Obr. 13 Příklad s jedincem o třech parametrech

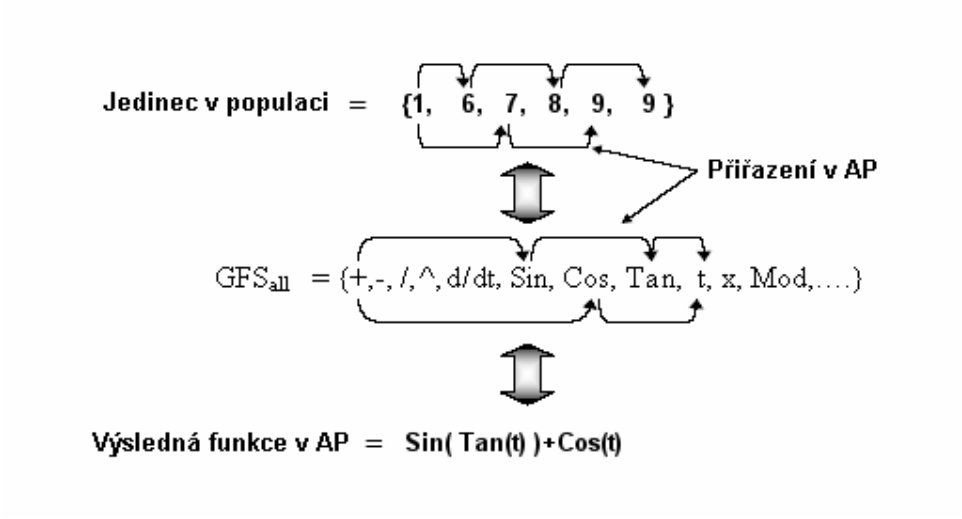
Složení podmnožin prezentované v GFS je zásadně důležitá pro AP. Užívá se, aby nedošlo k syntéze patologických programů, tj. programů obsahující funkce bez argumentů atd. Výkon AP je samozřejmě lepší, jestliže je výběr funkcí GFS odborně vybrán na základě předchozích zkušeností s řešeným problémem.

Důležitou částí AP je sekvence matematických operací, která je použita pro syntézu programu. Tyto operace jsou užity pro transformaci jedinců populace do vhodného programu. Matematicky řečeno, jde o přepis jedince do plně funkčního programu. Tento přepis se skládá ze dvou hlavních částí, první částí je DSH a druhá část jsou bezpečnostní procedury které nedovolí syntetizovat patologické programy.

V AP je DSH užito pro transformaci jedince, společně s bezpečnostními procedurami vytváří výše uvedený přepis, které transformuje libovolného jedince do programu. Jedinci v populaci se skládají z celočíselných parametrů, které ukazují na určitou funkci v GFS.

V případě (Obr.13) je jedinec s indexy {3, 4, 2}vybranými z GFS. Kdyby neexistovala žádná bezpečnostní procedura zamezující tvoření patologických funkcí, pak by vznikla funkce ^/- označená šedě. Ale výsledná funkce AP je z/x. Následující část podrobně popisuje, jak se tvoří funkce z jedince s indexy.

V jedinci jsou náhodně vybraná čísla od 1 až po velikost souboru všech funkcí definovaném v GFS_{all}. Proces vytvoření nové funkce je následující. První index je nahrazen příslušnou funkcí z GFS_{all}. Pak podprogram v AP prověří kolik potřebuje vybraná funkce argumentů a kolik ještě zbylo indexů v jedinci. Další výběr funkcí je založen na těchto faktech.



Obr. 14 Příklad ideálního jedince v AP

Příklad na (Obr.14) představuje ideální případ jedince v AP. První index jedince je 1 představuje v GFS_{all} funkci + tato funkce má dva argumenty, proto indexy 6 a7 jsou argumenty +, jak ukazuje výraz (22).

$$6 + 7 \tag{22}$$

Index 6 je pak nahrazen funkcí Sinus a index 7 funkcí Cosinus viz. (23).

$$Sin + Cos \tag{23}$$

Sinus a cosinus jsou 1-argumentové funkce. Po indexu 7 následuje index 8, který je nahrazen funkcí Tangens, která je taky 1-argumentová a vložen do funkce Sin viz.(24).

$$Sin(Tan) + Cos \tag{24}$$

Po indexu 8 následuje index 9, kterému odpovídá proměnná t, která je argumentem funkce Cos, jak ukazuje výraz (25).

$$Sin(Tan) + Cos(t) \tag{25}$$

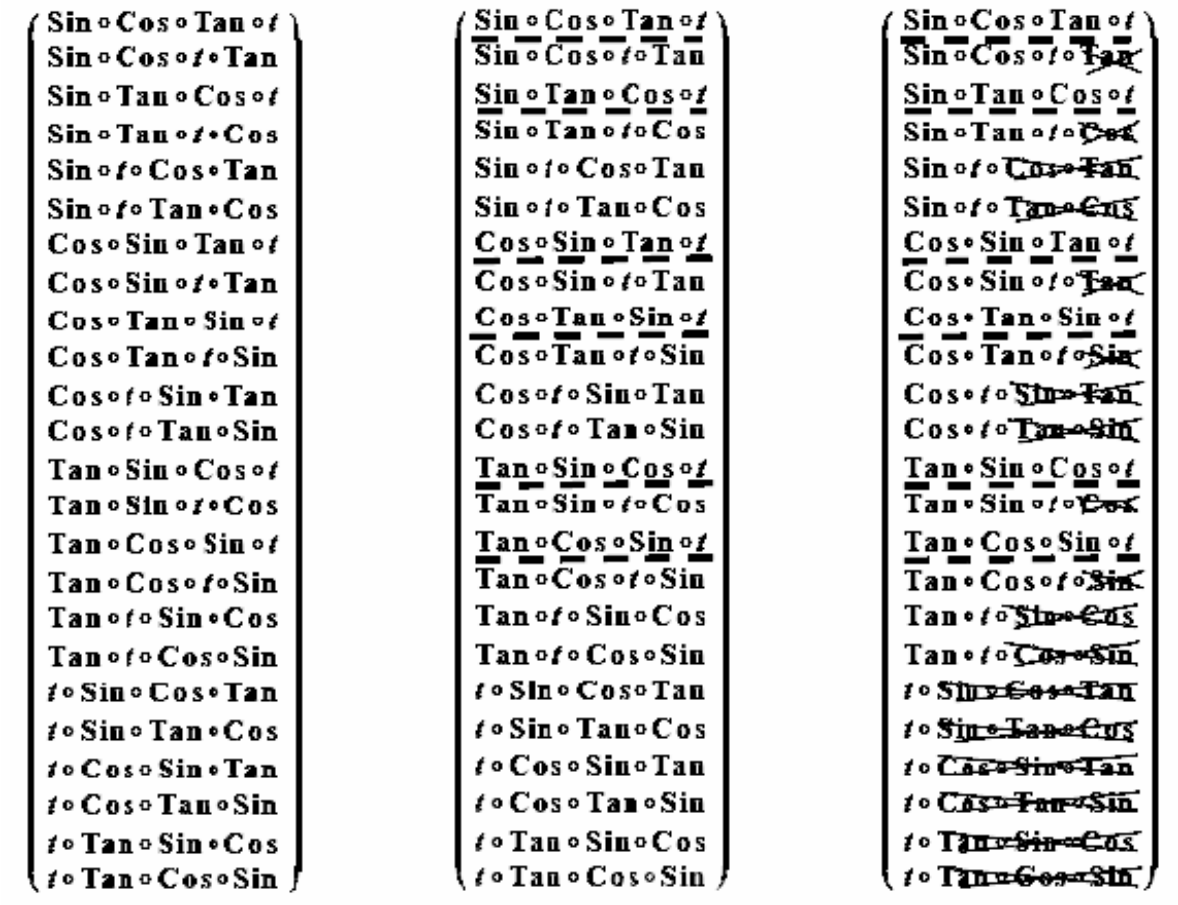
A poslední index je znovu 9, tedy t, které je argumentem Tangens. Výsledná funkce AP je pak výraz (26).

$$Sin(Tan(t)) + Cos(t) \tag{26}$$

Tento příklad byl ideální, protože volba indexů byla tak dobrá, že nevznikaly žádné patologické funkce.

3.3 Zajištění vytvoření nepatologických funkcí

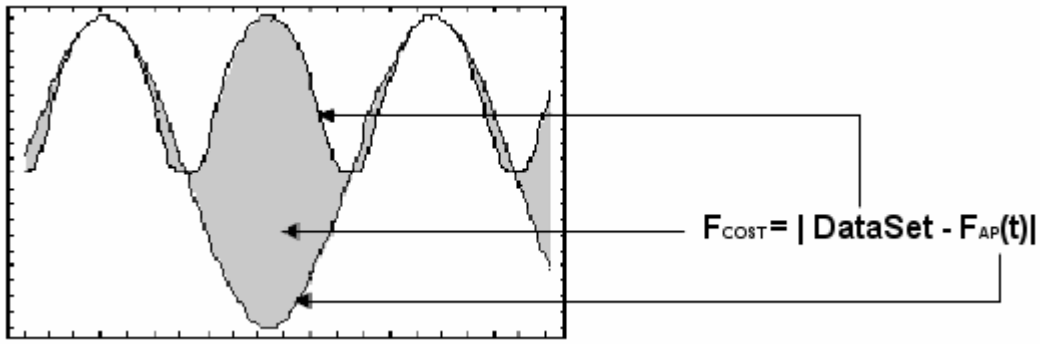
Zajištění vytvoření nepatologických funkcí je důležité. Znamená to že jsou vytvořeny jen funkce se správným počtem argumentů (viz.výše). (Obr.15) demonstruje, jak může vypadat patologická funkce před a po korekci. V obrázku jsou použity jen 4 základní funkce {Sin, Cos, Tan, t}. Tvar všech funkcí (patologických i nepatologických), které můžou být generovány, je dán permutací. V prvním sloupci je zobrazeno n! možných permutací, tedy 24 kombinací 4 základních funkcí. Všechny přirozené nepatologické funkce jsou podtrženy v druhém sloupci. Třetí sloupec zobrazuje patologické funkce po korekci. Patologické části jsou překříženy.



Obr. 15 Schéma korekce patologických funkcí

3.4 Vyhodnocení účelové funkce v AP

Analytické programování se využívá k vytvoření nových funkcí – programů. Může být použito pro nalezení funkcí, které dokáží vhodně prokládat neznámá data. Vhodná účelová funkce pro tyto případy je absolutní hodnota rozdílu mezi originálními daty a právě nalezenou funkcí, jak je zobrazeno na (Obr.16). Cílem AP je minimalizovat tento rozdíl, tedy minimalizovat šedou oblast na (Obr. 16). V nejlepším případě by hodnota účelové funkce měla být rovna nule.



Obr. 16 Ohodnocení účelové funkce u AP

3.5 Verze AP

Analytické programování bylo testováno ve třech verzích. Všechny tři verze používají k syntetizaci programu stejnou množinu funkcí, terminálů atd., jako používá Koza in GP [11]. Druhá verze (AP_{meta}, název první verze je AP_{basic}) je upravena ve smyslu odhadu konstanty. Například Koza používá pro tzv. Sextic úlohu (hledání analytického tvaru funkce dané polynomem 6. řádu). Koza [12] náhodné generování konstant, zatímco AP zde používá jen jednu konstantu pojmenovanou K, která je vložena do generovaného programu v různých místech evolučního procesu, např. (27). Když je program syntetizován, pak všechny K indexovány jako K_1, K_2, … K_n (28) a pak jsou všechny K_n odhadnuty použitím druhého evolučního algoritmu (29). Protože EA podřízený (slave) "pracuje pod" EA řídící (master), tedy EA_{master} ► program ► indexování K ► EA_{slave} ► odhad K_n) je tato verze pojmenována AP s metaevolucí – AP_{meta}.

$$\frac{x^2 + K}{\pi^K} \tag{27}$$

$$\frac{x^2 + K_1}{\pi^{K_2}} \tag{28}$$

$$\frac{x^2 + 3.56}{\pi^{-229}} \tag{29}$$

Protože je tato metoda časově náročná byla AP_{meta} upravena do třetí verze, která se liší od té druhé v odhadu K. Toho je dosaženo použitím vhodné metody nelineárního proložení (AP_{nf} non-linear fitting). Tato metoda ukázala nejslibnější výkony u úloh s neznámými konstantami. V této práci je využito první verze AP_{basic} [13].

II. PRAKTICKÁ ČÁST

4 DIFERENCIÁLNÍ EVOLUCE

Diferenciální evoluce (DE) je poměrně nový typ evolučního algoritmu (od r. 1995), který vyvinuli a poprvé použili 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 a podobně .

Postupně jsou vysvětleny řídící parametry a princip tohoto algoritmu.

Činnost a kvalita diferenciální evoluce je ovlivněna jejími řídícími parametry stejně jako u ostatních evolučních algoritmů. Jejich označení a význam je:

CR (0,1) jde o tzv. práh křížení. V případě , že se jedná o funkci, která je separabilní, pak je doporučeno nastavit tento parametr na hodnoty blízké 0. V opačném případě jsou výhodné hodnoty, které se blíží 1. V případě, že se CR nastaví na 0 dojde k tomu, že se mutace nedostane do zkušebního jedince, který díky tomu bude čistou kopií aktuálního (čtvrtého rodiče). Vývoj evoluce se pak zastaví. V případě, že CR bude nastavena na 1, bude zkušební jedinec tvořen pouze ze tří náhodně vybraných rodičů – jedinců z populace a diferenciální evoluce se bude spíše podobat náhodnému hledání, než evolučnímu algoritmu (všichni tři jedinci, byť evolučně vytvoření, jsou náhodně vybráni bez ohledu na jejich kvalitu). Je proto vhodné, by CR nikdy nenabývalo těchto hodnot.

NP - Jedná se o parametr udávající velikost populace. Hodnoty tohoto parametru jsou získány pouze zkušeností s tímto algoritmem. Jediné, co se dá s určitostí říci je, že NP by neměl být menší než 4. To je minimální velikost populace, při které diferenciální evoluce ještě pracuje.

F (0,2). Mutační konstanta je poslední řídící parametr diferenciální evoluce

Generations > 0. Udává počet evolučních cyklů, tzv. generací, během nichž se celá populace vyvíjí.

Dim – dimenze problému. Jde o počet argumentů účelové funkce. Parametr „Dim“ je tedy dán řešeným problémem a lze ho změnit pouze reformací problému.

Tab. 1 Hodnoty řídících parametrů

Řídící parametr	Interval	Optimum	Poznámka
NP	Uživatel		Velikost populace
F	<0,2>	0,3 – 0,9	Mutační konstanta
CR	<0,1>	0,8 – 0,9	Práh křížení
Generations	Uživatel		Počet kol šlechtění populace

4.1 Populace

Diferenciální evoluce pracuje s populacemi stejně jako ostatní evoluční algoritmy. Princip její tvorby a ošetření parametrů v jedincích je stejný jako pro ostatní evoluční algoritmy.

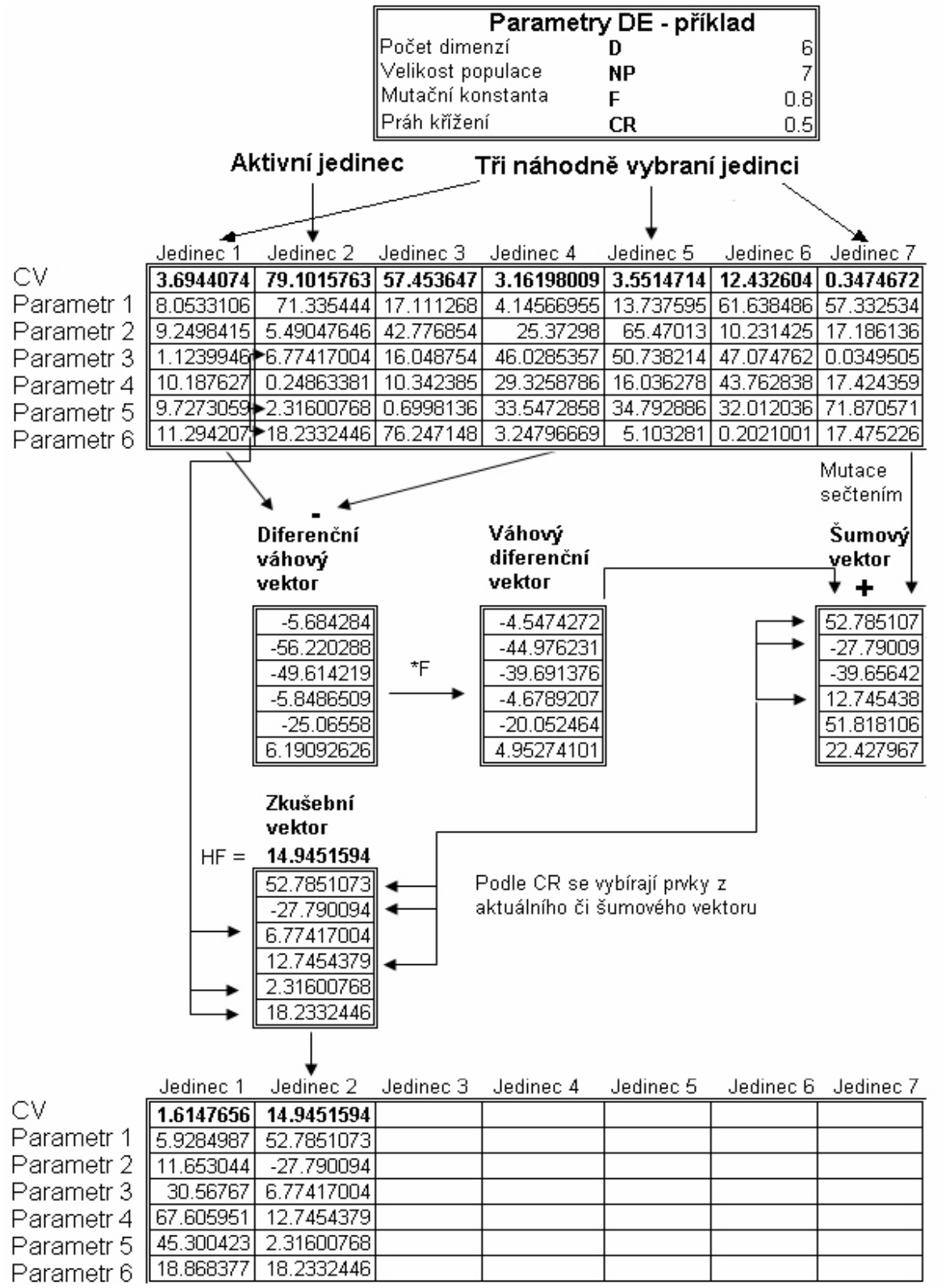
4.2 Mutace

V evolučních algoritmech stejně jako v klasické evoluční teorii hraje životně důležitou úlohu tzv. mutace. Diferenciální evoluce má tu zvláštnosti, že k vytvoření dalšího potomka je potřeba ne dvou, ale čtyř rodičů. (30).

$$v_j = x^G_{r3,j} + F * (x^G_{r1,j} - x^G_{r2,j}) \tag{30}$$

4.3 Křížení

Křížení u diferenciální evoluce nastává až po mutačním procesu, zatímco u většiny evolučních algoritmů je první vytvořen potomek křížením a teprve poté dochází k jeho mutaci. Proces křížení z DE znamená, že se ze čtvrtého doposud nepoužitého rodiče a šumového (zmutovaného) vektoru vytvoří nový jedinec, tzv. zkušební (trial) vektor. Ten se vytváří za pomocí tzv. konstanty či prahu křížení CR a to tak, že se v cyklu vybírají korespondující parametry ze čtvrtého a šumového jedince (oba první parametry, oba druhé parametry,…) a pro každou takto vybranou dvojici je generováno náhodné číslo. Pokud je menší než CR, do příslušného parametru ve zkušebním vektoru se přesune parametr z jedince šumového a naopak. To je znázorněno na (Obr. 17).



Obr. 17 Princip DE (převzato z [2])

4.4 Princip činnosti DE

Cílem DE je v cyklech zvaných „generace“ vyšlechtit co nejlepší populaci jedinců ve smyslu hodnot účelové funkce, jenž je spojen a s každým jedincem. Během každé generace se provádí tyto kroky:

Nejprve se stanoví parametry (F, CR, NP, Generations, Dim). Poté je nutné nadefinovat prototyp jedince (Specimen) – tj. z jakých typů čísel se budou skládat jedinci, např. reálných, celočíselných atd. Po sestavení Specimen se vytvoří populace vygenerováním množiny jedinců podle prototypového vektoru. U každého jedince se musí počítat s jedním prvkem navíc a tím je hodnota účelové funkce. Během generace se provádí ještě cyklus, který zabezpečuje postupné evoluční šlechtění každého jedince z populace. V tomto cyklu se postupně vybírá jeden jedinec (aktivní jedince, cílový vektor) za druhým až do konce populace (tím je ukončena jedna generace), a pro každého z nich je proveden evoluční cyklus, v němž je prováděna mutace a křížení, tj. náhodně se zvolí tři další různé vektory (jedinci) z populace. První dva se od sebe odečtou získá se tzv. diferenční vektor. Ten se vynásobí mutační konstantou F, která jej tím pádem změní , a získá se váhový diferenční vektor. Ten se přičte k třetímu náhodně vybranému vektoru a získá se tzv. šumový vektor. Poté si připraví tzv. „zkušební vektor“ a z cílového a šumového vektoru se bere postupně jeden prvek za druhým (první z obou, druhý z obou…) a pro takto vybranou každou dvojici se generuje náhodné číslo v rozsahu 0 – 1 a porovnává se s konstantou CR. Pokud je toto číslo menší než CR, pak se do příslušné pozice ve zkušebním vektoru umístí prvek z vektoru šumového a v opačném případě z vektoru cílového. Tak se získá zkušební vektor, jehož hodnota účelové funkce se porovná s hodnotou účelové funkce cílového vektoru (Obr. 17). Na pozici cílového vektoru v nové populaci je vybrán ten vektor – jedinec, který má hodnotu účelové funkce lepší .

Diferenciální evoluce je ukončena pouze tehdy, provede-li se uživatelem zadaný počet generací. Jiný ukončovací parametr tento algoritmus nemá.

Kvalitu a průběh šlechtění lze ovlivnit mnoha faktory:

1) **Nastavení řídících parametrů** (např. nízká hodnota CR může evoluci zbrzdit)
2) **Velikost populace** (při malé populaci bude horší výběr, při velké bude potřeba více času na vytvoření nové)
3) **Počet generací** (při zadaném malém počtu generací může evoluce skončit dříve než nalezne extrém)
4) **Definice účelové funkce** (pokud je nevhodně nadefinovaná, může evoluce probíhat pomalu nebo vůbec)
5) **Definice intervalu** (evoluce se drží uprostřed definovaného prostoru. Interval musí být nadefinován tak, aby bylo možno, v něm daný extrém najít)

Diferenciální evoluce byla již vyzkoušena na mnoha optimalizačních problémech s velmi dobrými výsledky. Tyto výsledky byly porovnány s výsledky od jiných algoritmů, které byly aplikovány na stejné problémy. Diferenciální evoluce vykazovala v mnoha případech lepší výsledky. V další kapitole je vytvořena analýza na 2 účelových funkcích a porovnání dosažených výsledků DE s algoritmem SOMA

Tento algoritmus má ty zvláštnosti, že ke tvorbě nového jedince je potřeba celkem 4 rodičů, kteří vytvoří potomka, jenž nakonec soupeří o místo v nové populaci. V procesu tvorby nového potomka je uplatněna náhoda pomocí křížící konstanty CR a pomocí tzv. šumového vektoru. Díky tomu je diferenciální evoluce poměrně robustní algoritmus s poměrně vysokou diverzitou.

V praxi existuje 10 možných variant DE, princip je stejný, liší se pouze ve způsobu výpočtu šumového vektoru.

4.5 Varianty diferenciální evoluce

V současné době existuje asi 10 variant diferenciální evoluce. V těchto vztazích je ukázáno deset různých způsobů výpočtu šumového vektoru. Možné výpočty šumového vektoru (kde případný výskyt x_{best} představuje nejlepšího jedince z populace) tedy jsou znázorněny v (Tab.2)

Tab. 2 Přehled 10 variant DE

DE/best/1/exp	$v = x^G_{best,j} + F*(x^G_{r2,j} - x^G_{r3,j})$
DE/rand/1/exp	$= x^G_{r1,j} + F*(x^G_{r2,j} - x^G_{r3,j})$
DE/rand-to-best/1/exp	$= x^G_{i,j} + \lambda*(x^G_{best,j} - x^G_{i,j}) + F*(x^G_{r1,j} - x^G_{r2,j})$
DE/best/2/exp	$= x^G_{best,j} + F*(x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j})$
DE/rand/2/exp	$= x^G_{r5,j} + F*(x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j})$
DE/best/1/bin	$= x^G_{best,j} + F*(x^G_{r2,j} - x^G_{r3,j})$
DE/rand/1/bin	$= x^G_{r1,j} + F*(x^G_{r2,j} - x^G_{r3,j})$
DE/rand-to-best/1/bin	$= x^G_{i,j} + \lambda*(x^G_{best,j} - x^G_{i,j}) + F*(x^G_{r1,j} - x^G_{r2,j})$
DE/best/2/bin	$= x^G_{best,j} + F*(x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j})$
DE/rand/2/bin	$= x^G_{r5,j} + F*(x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j})$

5 ALGORITMICKÝ ZÁPIS DIFERENCIÁLNÍ EVOLUCE

Tato část se bude zabývat konstrukcí separovaných operátorů v programu Mathematica.

5.1 Původní DE

```
DE[Pop_] := MapThread[If[#1[[1]] < #2[[1]], #1, #2] &,
  {Pop,
   ({CostFunction[#1], #1} & /@
     (Flatten[MapIndexed[CheckInterval[#1, #2] &, #1], 1] & /@
       (Flatten[Table[If[Cr < Random[], Pop[[i, 2, j]], #1[[i, j]]], {i, NP}, {j, Dim}] & /@
         {F * (#1[[1, 2]] - #1[[2, 2]]) + #1[[3, 2]] & /@ (Pop[[#1]] & /@ MapIndexed[SelectOther[#2[[1]]] &, Pop])}, 1])))}]
```

Obr. 18 Zápis DE v programu Mathematica

Původní zápis diferenciální evoluce v programu Mathematica. Tento program byl vytvořen Doc. Ing. Ivanem Zelinkou, PhD. Pracuje na principu, kde všechny operátory (selekce, mutace, křížení) pracují paralelně. Byla zvolena varianta DE/rand/1/bin, která je považována za prozatím nejlepší verzi diferenciální evoluce.

5.2 Separovaná DE na jednotlivé operátory

K vytvoření evolučního algoritmu pomocí AP bylo zvoleno diferenciální evoluci z důvodu její jednoduché struktury. DE pracuje jak již bylo zmíněno se 3 operátory :selekcí, mutací a křížení.

Tento postup lze chápat jako první studii, kdy pozdějším výsledkem může být i vyšlechtění nového a kvalitnějšího algoritmu. Pro naše účely byla vybrána z 10 verzí diferenciální evoluce variace typu DE/rand/1/Bin, která jak už bylo zmíněno, se považuje za nejlepší verzi DE. Pro vytvoření evolučního algoritmu a jeho znovu sestavení pomocí analytického programování je nutné, aby byly jednotlivé operátory byly separovány, tedy rozděleny do samostatných smyček. Operátory musí fungovat sekvenčně, tzn. výstup jednoho musí navazovat na vstup druhého. Nyní zde budou zobrazeny a popsány jednotlivé rozseparované operátory a jejich popis v Matematice.

```
SelectionDE[Pop_, i_] := Module[{parents}, indnr = i; parents = Pop[[#]] & /@ SelectOther[Pop[[i]]];
  Return[PrependTo[parents, Pop[[i]]]]]
```

Obr. 19 Popis operátoru selekce u DE v Mathematice

Operátor SELEKCE vybírá jedince z populace pro další instrukce. V tomto případě výstupem jsou 4 jedinci – jeden aktivní a 3 náhodně zvolení.

Tento operátor se zahrnuje v AP mezi terminály, což v AP představuje konstanty nebo proměnné. Jde tedy o 0-argumentové funkce (GFS_{0arg}).

```
MutationDE[IndividualsIIII_] :=
 Module[{parentsIIII}, IndNeededIIII = 4;
  parentsIIII = popforuse[[#]] & /@ SelectOtherIIII[popforuse[[indnrIIII]]];
  Do[PrependTo[parentsIIII, IndividualsIIII[[i]]], {i, Length[IndividualsIIII]}];
  IndividualsTempsIIII = Take[parentsIIII, IndNeededIIII];
  indsIIII =
   Flatten[MapIndexed[CheckIntervalIIII, FIIII * (IndividualsTempsIIII[[1, 2]] - IndividualsTempsIIII[[2, 2]]) +
      IndividualsTempsIIII[[3, 2]]]]; Return[{{CostFunctionIIII[indsIIII], indsIIII}}]]
```

Obr. 20 Matematický popis operátoru mutace u DE

Do operátoru MUTACE vstupují 4 jedinci ze SelektionDE. Mutace představuje vytvoření šumového vektoru ze 3 náhodně vybraných rodičů, pak se provádí mutace, tedy kombinace těchto tří náhodně vybraných rodičů. Konkrétně je mutace provedena tak, že se rozdíl dvou prvních náhodně vybraných rodičů vynásobí mutační konstantou F, a výsledný vektor se přičte ke zbývajícímu třetímu.

Pro AP představuje mutace jednoargumentovou funkci (GFS_{1arg}), tzn. že potřebuje mít na vstupu nějakého jedince.

```
CrossoverDE[Individual_] :=
 Module[{}, IndividualTemp = If[Dimensions[Dimensions[Individual]][[1]] == 1, Individual, Flatten[Take[Individual, 1], 1]];
  inds = Flatten[MapIndexed[CheckInterval, Table[If[Cr < Random[], Population[[indnr, 2, j]], IndividualTemp[[2, j]]], {j, Dim}]]];
  Return[{CostFunction[inds], inds}]]
```

Obr. 21 Matematický popis operátoru křížení u DE

Operátor KŘÍŽENÍ u DE nastává až po mutačním procesu, zatímco u většiny evolučních algoritmů je první vytvořen potomek křížením, teprve poté dochází k jeho mutaci. To znamená, že se ze čtvrtého doposud nepoužitého rodiče a šumového vektoru vytvoří nový jedinec, tzv. zkušební (trial) vektor. Ten se vytváří za pomocí prahu křížení CR a to tak, že se v cyklu vybírají korespondující parametry ze čtvrtého a šumového jedince (oba první parametry, oba druhé parametry,…) a pro každou takto vybranou dvojici je generováno náhodné číslo v rozsahu 0-1. Pokud je větší než CR, do příslušného parametru ve zkušebním vektoru se přesune parametr z aktivního jedince.

Křížení pro AP představuje stejně jako u mutace jednoargumentovou funkci (GFS_{1arg}), tedy na vstupu musí být nějaký jedinec.

Všechny operátory jsou pak aplikovány na každého jedince sekvenčně.

```
FinalDE[Population_] := Table[ind = ReleaseHold[RetFce[Population, popnr]];
  If[ind[[1]] ≤ Population[[popnr, 1]], ind, Population[[popnr]]], {popnr, 1, NP}]
```

Obr. 22. Finální funkce pro nastavení AP

Tato funkce uzavírá celý program a zajišťuje fungování algoritmu.Jde o selekci do nové populace podle hodnoty účelové funkce. Slouží pro nastavení pro AP. V této funkci se rozhoduje, zda do nové populace postupuje jedinec s lepší účelovou funkcí nebo jedinec, který se přebírá z původní populace. Zde je jedinec porovnáván s aktivním jedincem a ten s lepší nebo stejnou hodnotou účelové funkce..

DE lze popsat rovnicí (31) na základě výše uvedených operátorů.

$$\text{CrossoverDE(MutationDE(SelectionDE)))} \tag{31}$$

6 ÚČELOVÁ FUNKCE

Hlavním účelem všech evolučních algoritmů je optimalizace účelové funkce, tzn. nalezení jejich nejvhodnějších argumentů. Při řešení těchto problémů je obvyklé, že se pracuje s argumenty optimalizovaných funkcí, přičemž ty mohou nabývat různých oborů hodnot od celých čísel po čísla komplexní atd. Někdy se také může stát, že pro určité subintervaly z povoleného intervalu hodnot může příslušný argument optimalizované funkce opět nabývat různých typů hodnot(celočíselných, reálných, komplexních, diskrétních apod.). Navíc mohou být v rámci optimalizace, uplatněny různé penalizace a omezení nejen na dané argumenty, ale také na funkční hodnotu optimalizované funkce.

Základem každého optimalizačního problému a tedy i jeho řešení je účelová funkce, což je funkce nezávislých proměnných jejíž optimalizace(nalezení minima či maxima) povede k nalezení optimálních hodnot jejích parametrů. Definování takové funkce je jedním z kritických bodů optimalizace a může rapidně ovlivnit kvalitu výsledků. Na každou účelovou funkci nahlížíme jako na geometrický problém, v jehož rámci se hledá nejnižší (minimum) či nejvyšší (maximum) pozice na N rozměrné ploše. Počet dimenzí N je dán počtem optimalizovaných argumentů účelové funkce.

Účelová funkce v AP přiřazuje jedincům hodnotu, která vypovídá o vhodnosti a kvalitě jedince v dané populaci, na základě vygenerované komplexní rovnice.V případě vytvoření nového evolučního algoritmu je nutno vyzkoušet jeho vlastnosti na testovacích funkcích.

6.1 Dvě zvolené testovací funkce pro AP

V této předběžné studii je právě vytvořený nový evoluční algoritmus vyzkoušen na 2 testovacích funkcích. Nejdříve byla provedena analýza těchto funkcí s algoritmy SOMA a DE, aby bylo ukázáno, zda jsou tyto algoritmy schopny dosáhnout minima na daných funkcích. Pro každou funkci je zobrazen její analytický zápis – vzorec, její vzhled ve 3D a 2D graf.

Analýza byla provedena na 100 simulacích, aby bylo zjištěno chování algoritmů- tj. počet ohodnocení a s tím související konvergence k nalezení minima. Pro účelovou funkci pro AP bylo důležité, zda je nový algoritmus schopen dosáhnout minima na obou funkcích. Jako první funkci byla zvolena unimodální funkce DeJong a druhá multimodální Schwefelova funkce.

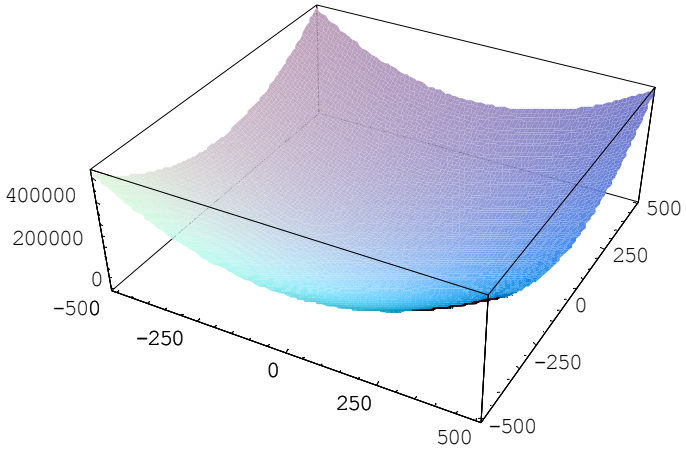
6.1.1 1. DeJong funkce

Jde o jednoduchou funkci typu mísa. Má pouze jediný extrém ve středu, jak ukazují (Obr.22 a Obr. 23). Rozsah je v intervalu <-512,512>.

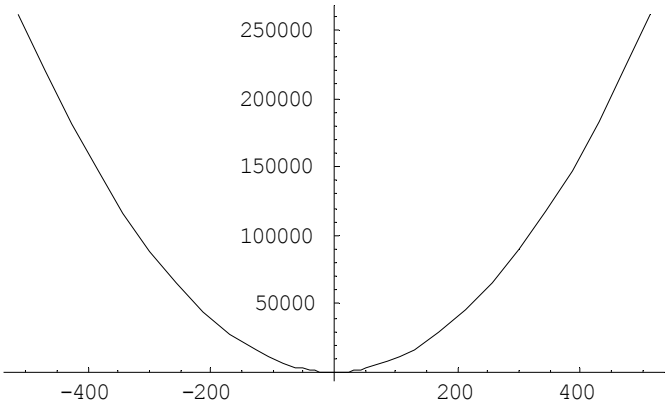
Funkce má matematický tvar (32).

$$\sum_{i=1}^{Dim} x^2 \tag{32}$$

Na (Obr.23 a 24) je znázorněno grafické zobrazení funkcí v 3D a 2D realizaci.



Obr. 23 Zobrazení DeJong funkce v 3D



Obr. 24 Zobrazení DeJong funkce v 2D

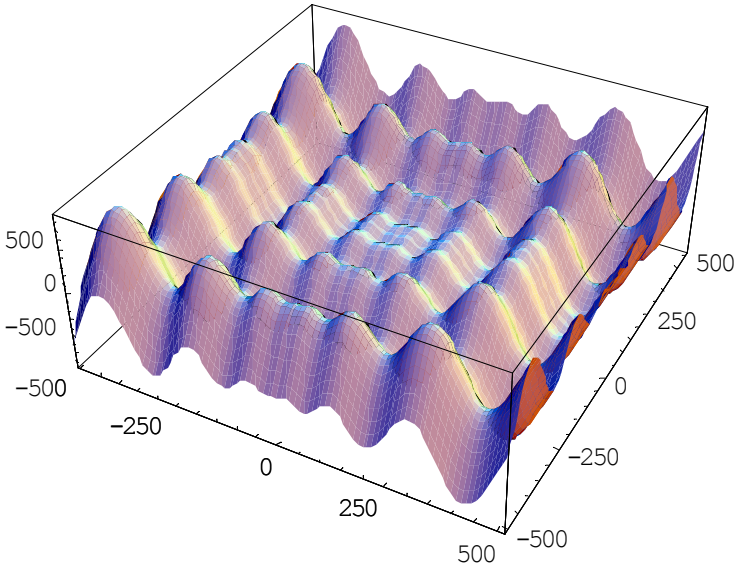
6.1.2 Schwefelova funkce

Jde o multimodální funkci , tzn. má mnoho extrémů, jak je vidět na (Obr.25 a Obr. 26). Rozsah je stejně jako u první zvolené funkce je v intervalu <-512,512>.

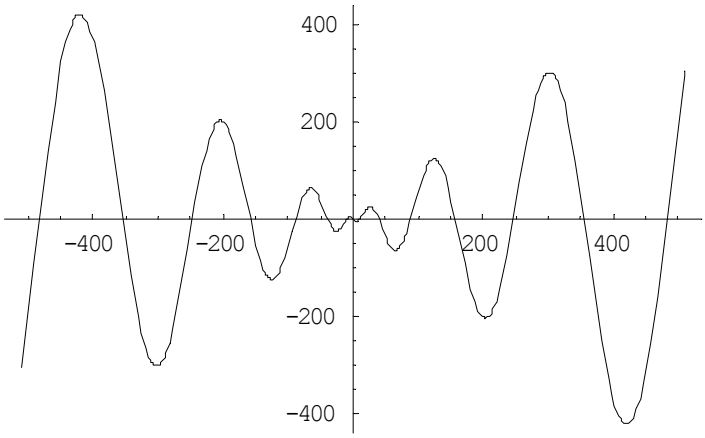
Funkce má matematický tvar (33).

$$\sum_{i=1}^{Dim} -x_i \sin\left(\sqrt{|x_i|}\right) \tag{33}$$

Na (Obr. 25 a 26)je opět znázorněno grafické zobrazení Schwefelovy funkce v 3D a 2D realizaci.



Obr. 25 Zobrazení Schwefelovy funkce v 3D



Obr. 26 Zobrazení Schwefelovy funkce v 2D

6.2 Natavení parametrů pro SOMA pro analýzu

Nejdříve byla provedena analýza těchto funkcí s algoritmy SOMA a DE, aby bylo ukázáno, zda jsou tyto algoritmy schopny dosáhnout minima na daných funkcích. V (Tab.3) jsou hodnoty, které byly nastavené pro algoritmus SOMA.

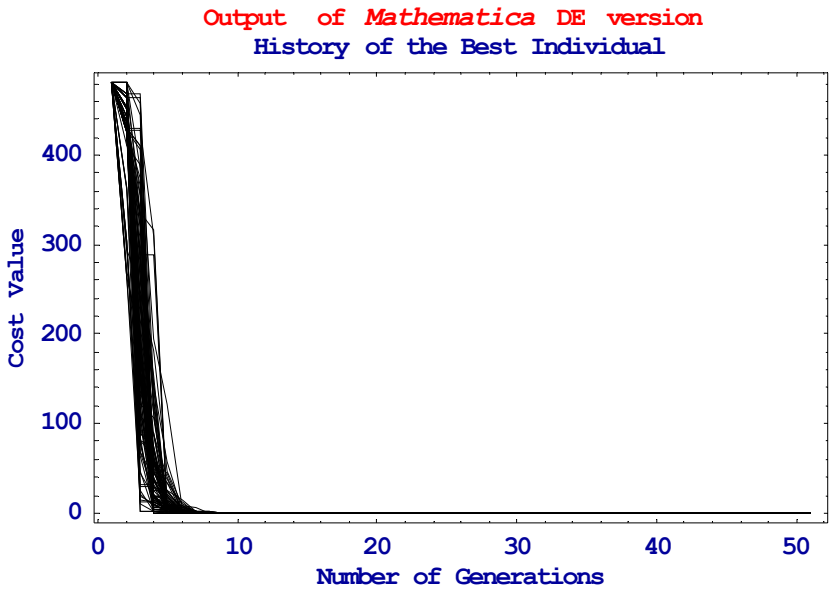
Tab. 3 Hodnoty nastavené pro algoritmus SOMA

Parametr	Nastavená hodnota	Vysvětlení parametru
PathLength	3	Délka cesty jedince
Step	0,11	Velikost kroku
PRT	0,1	Perturbační vektor
PopSize	20	Velikost populace
Specimen	Table[{{Real,[-512,512}}},{i,2}]	Prototyp jedince
Dim	2	Počet optimalizovaných proměnných
Migrations	50	Počet migračních kol
MinDiv	0,01	Rozdíl mezi nejhorším a nejlepším jedincem

Ohodnocení účelové funkce udává výraz (33). Pro zvolené nastavení je výsledek 25909,1.

$$\text{(PopSize-1) PathLength Migrations/Step} \qquad (33)$$

Analýza byla provedena na 100 simulacích pro funkci DeJong. Zjišťovalo se chování algoritmů, tj. počet ohodnocení, s tím související konvergence k nalezení minima. Dále jsou zde uvedeny maximální, minimální a průměrné hodnoty minima, kterého bylo dosaženo.Vývoj populace ukazuje (Obr. 27).



Obr. 27 Vývoj populace na DeJong funkci u SOMA

Po 100 simulacích byly nalezeny tyto hodnoty pro DeJong funkci:

minSOMA= $1{,}43308*10^{-29}$

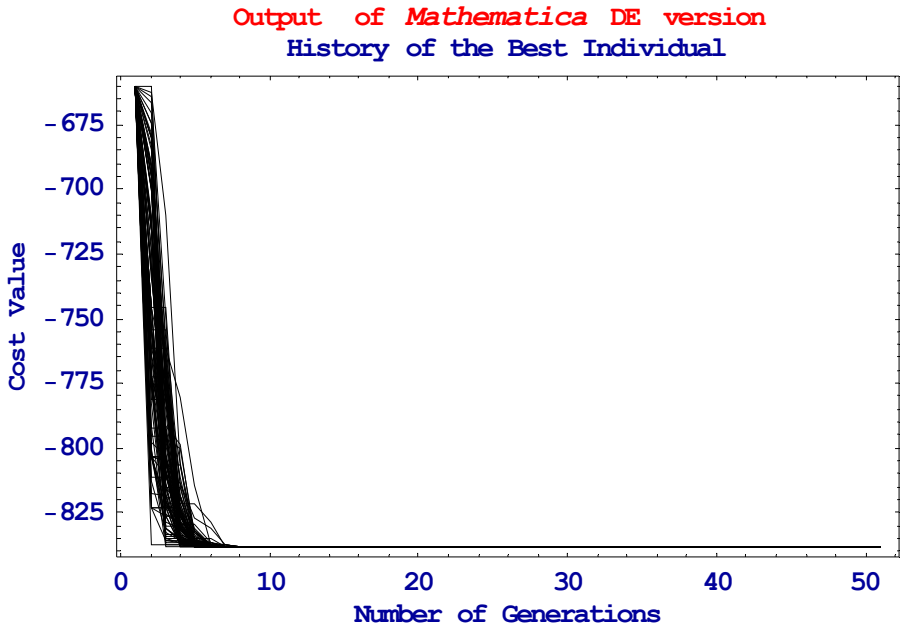
maxSOMA= $2{,}92294*10^{-19}$

průměrSOMA= $1{,}46201*10^{-20}$

V tomto případě došlo k nalezení extrému již za necelých 10 migračních kol, proto můžeme doplnit (34), což je asi 5x méně, než při původním počtu migrací.

(PopSize-1) PathLength 10/Step= 5181,2 (34)

Stejně jako u první funkce, tak i pro Schwefelovu funkci byla provedena analýza na 100 simulacích. Zjišťovalo se chování algoritmů- tj. počet ohodnocení, s tím související konvergence k nalezení minima..Vývoj populace ukazuje (Obr. 28)



Obr. 28 Vývoj populace na Schwefelově funkci u SOMA

Nalezené hodnoty pro Schwefelovu funkci po 100 simulacích:

minSOMA=- -837,966

maxSOMA=- -837,966

průměrSOMA= -837,966

Z dosažených výsledků je vidět, že všichni jedinci dospěli ke stejné hodnotě extrému (hodnoty se liší až v řádu desetitisícin).Jak lze vidět z grafu (Obr. 28) minima dosahuje nejlepší jedinec již asi v 8. migračním kole pro Schwefelovu funkci, proto platí (35).

(PopSize-1) PathLength 8/Step = 4145.45 (35)

6.3 Nastavení parametrů DE pro analýzu

Stejný postup jako u algoritmu SOMA byl proveden i pro Diferenciální evoluci. Tedy nejprve jsou zde znázorněny hodnoty nastavených parametrů, pak následují nalezené hodnoty pro obě funkce a počet ohodnocení účelové funkce.

Tab. 4 Nastavené hodnoty pro DE

Parametr	Nastavená hodnota	Vysvětlení parametru
Specimen	Table[{{Real,[512,512}}},{i,2}]	Prototyp jedince
Dim	2	Počet optimalizovaných proměnných
NP	20	Velikost populace
F	0,8	Mutační konstanta
CR	0,2	Práh křížení
Generations	1300	Počet generací

Ohodnocení účelové funkce udává výraz (36). Pro zvolené nastavení je výsledek 26000.

```
NP* Generations                                    (36)
```

Nalezené hodnoty pro DeJong funkci po 100 simulacích:

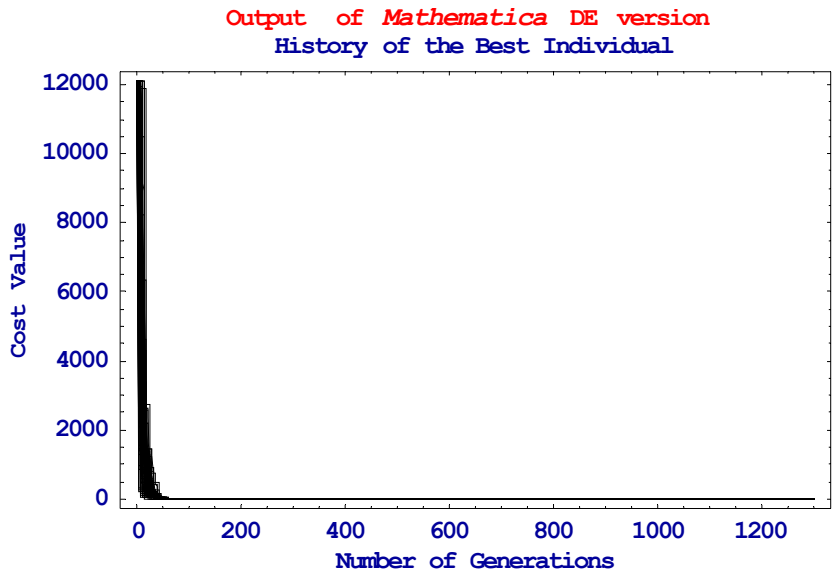
minDE=7,20434*10^{-73}

maxDE=4,12874*10^{-70}

průměrDE=9,71808*10^{-71}

Jak lze vidět z grafu (Obr. 28) minima dosahuje nejlepší jedinec již asi v 100. generaci pro DeJong funkci, proto platí výraz (37).

NP 20*100 Generations= 2000 (37)



Obr. 29 Vývoj populace na DeJong funkci u DE

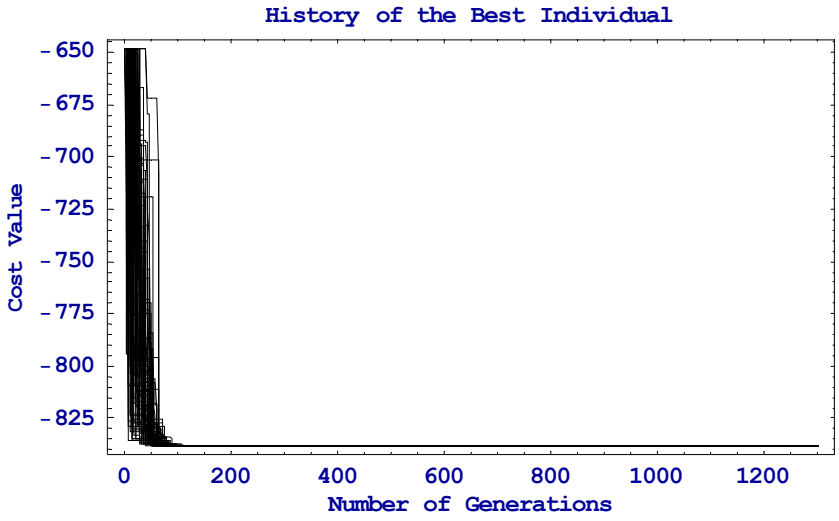
Nalezené hodnoty pro Schwefelovu funkci po 100 simulacích:

minDE= -837,966

maxDE= -837,966

průměrDE=-837,966

Z grafu (Obr. 30) minima dosahuje nejlepší jedinec již asi v 170. generaci pro Schwefelovu funkci, proto platí výraz (38).



Obr. 30 Vývoj populace na Schwefelově funkci u DE

NP20*170 generations= 3400 (38)

Stejně jako algoritmus SOMA, tak i DE našla stejný extrém u Schwefelovy funkce.

6.4 Porovnání analýzy algoritmů SOMA a DE

Výsledné hodnoty pro DeJong funkci uvádějí výrazy (39), (40), (41):

$$\text{min}=(\text{minSOMA}+\text{minDE})/2 = 7{,}16541*10^{-30} \quad (39)$$

$$\text{max}=(\text{maxSOMA}+\text{maxDE})/2 = 1{,}46147*10^{-19} \quad (40)$$

$$\text{průměr}=(\text{průměrSOMA}+\text{průměrDE})/2 = 7{,}31003*10^{-21} \quad (41)$$

Výsledné hodnoty pro Schwefelovu funkci jsou uvedeny ve výrazech (42),(43),(44):

$$\text{min}=(\text{minSOMA}+\text{minDE})/2 = -837{,}966 \quad (42)$$

$$\text{max}=(\text{maxSOMA}+\text{maxDE})/2 = -837{,}966 \quad (43)$$

$$\text{průměr}=(\text{průměrSOMA}+\text{průměrDE})/2 = -837{,}966 \quad (44)$$

Pro oba algoritmy se volilo nastavení tak, aby hodnoty jednotlivých parametrů byly reálné a zároveň aby byl přibližně stejný počet ohodnocení účelové funkce. (Tab.5) ukazuje přibližný (počet generací či migračních kol u jednotlivých funkcí je odhadnut z grafů) počet ohodnocení účelové funkce pro oba algoritmy.

Tab. 5 Porovnání počtu ohodnocení účelové funkce pro oba algoritmy

	Počet ohodnocení u SOMA	**Počet ohodnocení u DE**
Vypočteno dle nastavení	25909,1	26000
DeJong funkce	5181,2	2000
Schwefelova funkce	4145,45	3400

Podle počtu ohodnocení účelové funkce lze říci, že DE konverguje (tedy dosáhne extrému) poměrně rychleji než u algoritmu SOMA. Oba algoritmy ale prokázaly dostatečnou robustnost, extrému bylo dosaženo u obou. U DeJong funkce by se dalo říci, že DE byla blíže k extrému, ale rozdíl hodnot vzhledem k nízkému minimu je zanedbatelný.

Pro Schwefelovu funkci jsou výsledky jak pro algoritmus SOMA, tak i pro DE stejné při hodnotách v řádu stotisícin.

7 HLEDÁNÍ ALGORITMU POMOCÍ AP

Analytické programování se využívá k vytvoření nových funkcí. V našem případě se snažíme dokázat, zda AP je schopno sestavit z jednotlivých operátorů DE zpět diferenciální evoluci, popř. již ze zmíněných operátorů (selekce, mutace, křížení) vytvořit jiný algoritmus.

7.1 Testování nalezeného algoritmu na účelových funkcích

Schopnost AP je pak vyzkoušena na 2 výše zmíněných účelových funkcí. Ukázka programu na (Obr. 31) ukazuje výpočet hodnoty účelové funkce.

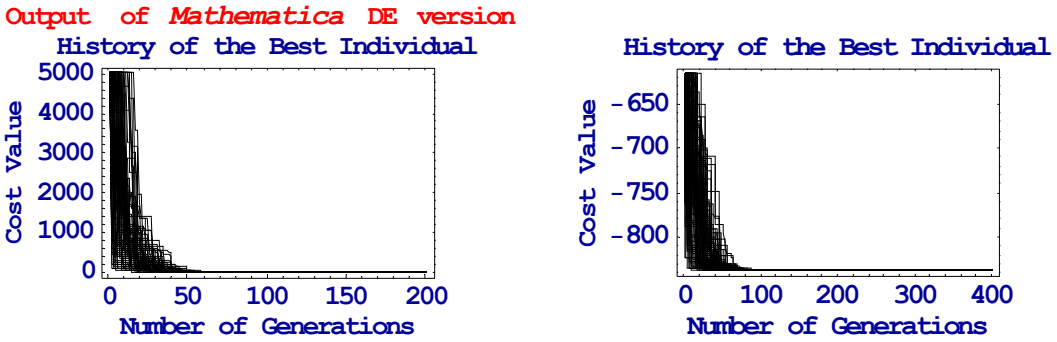
```
CostValue[foe_] := Module[{cv1, cv22, radim},
  Print[foe[[1]]];
  tempfoe = foe[[1]];
  RetFoe[PopulationIIII_, popnrIIII_] := Evaluate[tempfoe];
  funkce1 = foe[[1]] /. {Hold[SelectionDE[PopulationIIII, popnrIIII]] → SelectDE, Hold[MutationDE] → MutateDE,
     Hold[CrossoverDE] → CrossDE};
  CostFunctionIIII = DeJong;
  PopulationIIII = DoPopulationIIII[NPIIII, SpecimenIIII];

  FinalPopulationIIII = NestList[FinalDE, PopulationIIII, GenerationsIIII];
  Print[Last[FinalPopulationIIII]];

  best1 = Position[{##}, Min[##]][[1]][[1]] &@@Transpose[Last[FinalPopulationIIII]][[1]];
  ind1 = Last[FinalPopulationIIII][[best1]];
  Print[ind1];

  If[ind1[[1]] ≤ 0.00001,
   (CostFunctionIIII = Schwefel;
    PopulationIIII = DoPopulationIIII[NPIIII, SpecimenIIII];
    FinalPopulationIIII = NestList[FinalDE, PopulationIIII, GenerationsIIII];

    best2 = Position[{##}, Min[##]][[1]][[1]] &@@Transpose[Last[FinalPopulationIIII]][[1]];
    ind2 = Last[FinalPopulationIIII][[best2]];
    Print[ind2];

    If[ind2[[1]] ≤ -836., cv2 = ind2[[1]], cv2 = Abs[ind1[[1]]]]), cv2 = Abs[ind1[[1]]]];


  If[cv2 < -836., {pocitadlo, cv2, foe[[1]], funkce1, ind1[[1]], ind2[[1]]} >>> Algoritmy.txt;
   Print["Number of cost function evaluations: ", pocitadlo, "\n", "Cost value: ", cv2, "\n",
    "Solution: ", funkce1, "\n", "DeJong CV: ", ind1[[1]], "\n", "Schwefel CV: ", ind2[[1]]];

   Abort[], Null];

  pocitadlo++;



  Return[cv2]
]
```

Obr. 31 Výpočet hodnoty účelové funkce

Z (Obr. 31) vyplývá, že nejdříve získáme zápis algoritmu vytvořeného AP. Hodnota účelové funkce byla navržena tak, že nejdříve vygenerovaný algoritmus je zkoušen, zda je schopen najít minimum na jednoduché unimodální funkci DeJong. Zjišťuje se, zda minimum, které se našlo, je menší než 10^{-5}. (U DeJong funkce je minimum 0, omezení je zvoleno na 10^{-5}).Jestliže minima je dosaženo, začíná se testovat na multimodální Schwefelově funkci. Pak účelová hodnota je výstupem z Schwefelovy funkce. Jestliže není uspokojivý výsledek z funkce DeJong (nebylo dosaženo minima), pak výstupní hodnota je absolutní hodnota funkce DeJong.

7.2 Vyhodnocení výsledků

To, co zde bylo popsáno, můžeme chápat jako předběžnou studii. Cílem těchto simulací je nalezení originální DE. Parametry byly nastaveny podle předchozí analýzy, z níž vyplývá, že k nalezení algoritmu není potřeba původních 1300 generací, což je i z časového hlediska náročnější. Proto je zvoleno pouze 200 pro DeJong funkci a 400 pro Schwefelovu funkci. Následující (Obr. a Obr.) ukazují typické chování během 100 simulacích.



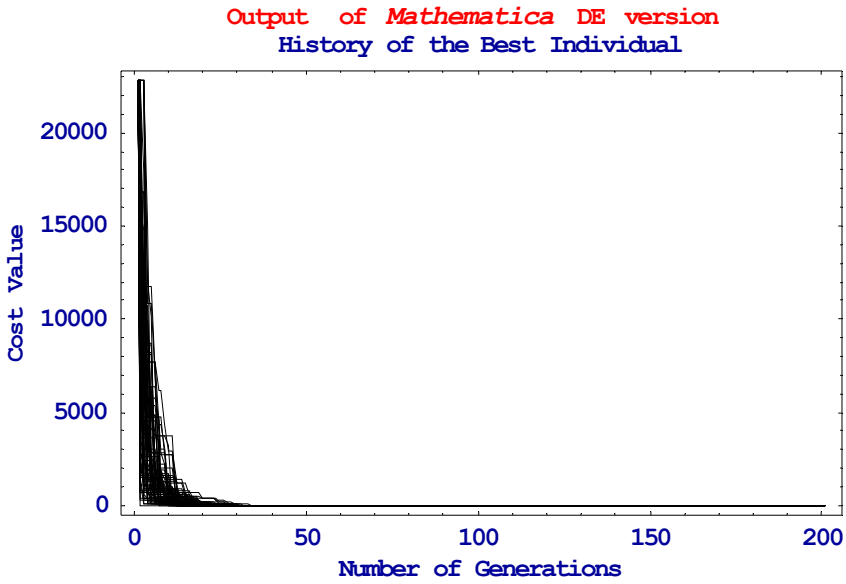
Obr. 32 DE pro DeJong funkci (vlevo) a Schwefelovu funkci

Délka jedince byla v analytickém programování nastavena na 15 a množství jednotlivých funkcí je 3 (selekce, mutace, křížení). To znamená, že existuje 32 767 možností generovaného algoritmu podle variací s opakováním. Již první generace, která byla náhodně vygenerována bez evoluce uměla najít algoritmus, který byl schopen dosáhnout minima. V následujících výrazech jsou uvedeny neúspěšní (výraz (45)) i úspěšní jedinci (výraz(46)) vygenerovaného algoritmu.

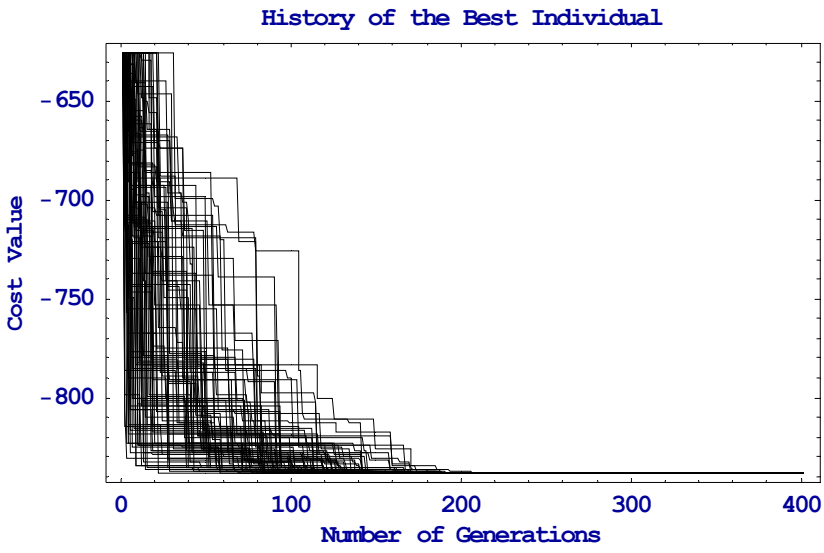
CrossoverDE[CrossoverDE[SelectionDE]]
SelectionDE (45)
CrossoverDE[SelectionDE]]

CrossoverDE[MutationDE[SelectionDE]] – originální DE
MutationDE[MutationDE[MutationDE[MutationDE[SelectionDE]]]] (46)

Z výrazu (46) lze vidět, že AP zvládlo vygenerovat zpět DE, což bylo naším úkolem. Zároveň je schopno i vygenerovat nový algoritmus. Na tomto typu bylo vyzkoušeno opět 100 simulací se stejným nastavením jako měla originální DE. Následující grafy (Obr.33. a Obr.34) ukazují vývoj populace v tomto vygenerovaném algoritmu.



Obr. 33 Vývoj populace na DeJong funkci



Obr. 34 Vývoj populace na Schwefelově funkci

Porovnání výsledků DE a nově vygenerovaného algoritmu během 100 simulací uvádí (Tab.6)

Tab. 6 Porovnání výsledků originální DE a generovaného algoritmu

	Originální DE		**Generovaný algoritmus**	
	DeJong	**Schwefel**	**DeJong**	**Schwefel**
Minimum	$2{,}04492*10^{-8}$	-837,966	$2{,}39949*10^{-16}$	-837,966
Maximum	$6{,}61369*10^{-6}$	-837,966	$1{,}45227*10^{-14}$	-837,966
Průměr	$9{,}29224*10^{-7}$	-837,966	$3{,}6897*10^{-15}$	-837,966

Z (Tab.6) lze vidět, že vygenerovaný algoritmus je schopen nalézt minimum stejně jako originální DE. Je v podstatě stejný, jen je zde použito více mutací a není použit operátor křížení. V unimodální funkci je konvergence rychlejší než u původní DE, ale v případě Schwefelovy funkce potřebuje více času na dosáhnutí minima. Důležitější je rychlejší konvergence v multimodální funkci, protože s takovými problémy se můžeme denně potkat v optimalizaci. Požadavkem je najít minimum tak rychle, jak jen je to možné, ovšem musí být ale dosaženo globálního minima, nikoli lokálního.

Tato diplomová práce usiluje o metodu,jak vytvořit nový evoluční algoritmus pomocí použití symbolické regrese s použitím analytického programování. Na základě dosažených výsledků můžeme uvést:

1) analytické programování lze užít jako nástroj pro vytvoření nových evolučních algoritmů
2) Tato práce ukazuje, že AP je schopno nalézt DE stejnou jako byla originální a dále také nalézt jiné algoritmy s dobrým chováním, jak je možno vidět na (Obr.33 a Obr.34) a také v (Tab.6)
3) Vše bylo nalezeno již během první náhodně vygenerované populace.
4) Všechny operátory by měly mít stejnou strukturu týkající se vstupů a výstupů.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit literární rešerši v oblasti evolučních algoritmů a jejich operátorů. V první části této práce je kompletní popis všech současných metod evolučních algoritmů a jejich operátorů, dále je popsáno analytické programování jako nástroj symbolické regrese.

V další části je uveden algoritmický způsob zápisu těchto operátorů, což znamená separaci jednotlivých operátorů zvoleného algoritmu (v našem případě diferenciální evoluce), aby bylo možno jednotlivé operátory použít pro analytické programování, pomocí něhož pak bylo vyzkoušeno znovu sestavení originální diferenciální evoluce. Těchto výsledků bylo dosaženo a navíc byl vygenerován nový algoritmus, který prokazuje dobré chování podobně jako DE.

Tato diplomová práce je pojata jako předběžná studie jednoho evolučního algoritmu a jeho operátorů, zda je AP schopno pomocí symbolické regrese znovu sestavit originální algoritmus. Jelikož se požadovaných výsledků dosáhlo, lze vidět, že AP lze použít jako nástroj k vytvoření nových evolučních algoritmů. Nám se podařilo vyšlechtit algoritmus sestavený ze dvou operátorů diferenciální evoluce (mutace a selekce) a po testování na 100 simulacích bylo zjištěno, že vygenerovaný algoritmus je schopen nalézt minima stejně jako diferenciální evoluce.

Díky této metodě lze vyšlechtit i nový, mnohem dokonalejší algoritmus, což by byl problém pro další studie, které však nejsou v rozsahu této diplomové práce. Ty se mohou zabývat hledáním lepší kvality řešení, což zahrnuje rychlost konvergence, množství ohodnocení účelové funkce nebo množství argumentů účelové funkce. Tyto vlastnosti prokazují robustnost generovaného algoritmu. Rovněž základní sada elementárních funkcí může být doplněna o operátory jiných evolučních algoritmů, které byly například popsány v teoretické části.

SEZNAM POUŽITÉ LITERATURY

[1] Kvasnička V., Pospíchal J., Tiňo P., Evolučné algoritmy, STU Bratislava, 2000, ISBN 80-227-1377-5

[2] Zelinka Ivan, Umělá inteligence v problémech globální optimalizace, BEN, 2002, 190 p. ISBN 80-7300-069-5

[3] Klimánek D., Detekce diskredibility senzoru u Kotlena Biomasu optimalizačními algoritmy, dostupné z WWW: http://dsp.vscht.cz/konference_matlab/matlab05/prispevky/klimanek/klimanek.pdf

[4] Hildebrand Antonín, Pathfinding algoritmy, Dostupné z WWW: <http://pathlib.hildebrand.cz/doc/Referat/pathref.html>

[5] Kubincová K.,Návrh a analýza algoritmov , Dostupné z WWW: <http://user.edi.fmph.uniba.sk/kubincova/Prednasky%20z%20DSA%20HTML/navrh_a_analyza_algoritmov.html>

[6] Bäch T., Fogel B. D., Michalewicz Z., Handbook of Evolutionary Computation, IOP Publishing Ltd and Oxford University, 1997, ISBN07-5030-895-8

[7] X. Hu, R. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: a case study on n-queens problem. IEEE Swarm Intelligence Symposium 2003, Indianapolis, IN, USA dostupné z WWW: http://www.swarmintelligence.org/xhu.php>

[8] Relationship between Genetic algorithm and Ant Colony Optimization algorithm [online]. Gomez Osvaldo, Barón Benjamin. Převzato z WWW: <http://www.cnc.una.py/cms/invest/download.php>

[9] Rektorys K., Variational methods in Engineering Problems and Problems of Mathematical Physics, Czech edition, 1999, ISBN 80-200-0714-8

[10] Oplatková. Z.: Analytic Programming, diplomová práce, UTB Zlín, 2003, 74 p.

[11] Koza J.R.,Bennet F.H., Andre D., Keane M. 1999, Genetic Programming III, Morgan Kaufnamm pub., ISBN 1-55860-543-6, 1999

[12] Koza J. R., Genetic Programming, MIT Press, 1998, ISBN 0-262-11189-6

[13] Zelinka I., Oplatkova 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

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACO	Ant colony optimization (optimalizace mravenčích kolonií)
AP	Analytické programování
Breadth-first search	Prohledávání do šířky
CR	Křížící konstanta u DE
DE	Diferenciální evoluce
Depth-first search	Prohledávání do hloubky
Dim	Dimenze problému, počet argumentů účelové funkce
Diverzita	Rozložení
DSH	Množina diskrétních hodnot
EA	Evoluční algoritmus
F	Mutační konstanta u DE
Fitness	Ohodnocení účelové funkce
GA	Genetický algoritmus
GE	Gramatická evoluce
Generations	Množství evolučních cyklů
GFS	Obecný funkční prostor u AP
GP	Genetické programování
Kardinalita	Porovnání konečných množin
Konvergence	Rychlost dosáhnout extrému
Leader	Nejlepší jedinec u algoritmu SOMA u varianty All To One
Multimodální funkce	Funkce s více extrémy
NP	Velikost populace
Patologická funkce	Nesmyslná funkce
SA	Simulované žíhání

Scatter search	Rozptýlené prohledávání
SGA	Standardní genetický algoritmu
Soft Brood	Forma selekce
SOMA	Samoorganizující se migrační algoritmus
Specimen	Prototyp jedince
Symbolická regrese	Nadstavba evolučních algoritmů
Tabu Search	Algoritmus Zakázané prohledávání
Terminál	Konstanty nebo nezávislé proměnné u AP
Trial	Zkušební vektor
Unimodální funkce	Funkce s jedním extrémem

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁZEV PŘÍLOHY