

Framework pro distribuované evoluční algoritmy

Framework for distributed evolutionary algorithms

Daniel Pohuba

Bakalářská práce
2011



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Daniel POHUBA**

Osobní číslo: **A08083**

Studijní program: **B 3902 Inženýrská informatika**

Studijní obor: **Informační a řídicí technologie**

Téma práce: **Framework pro distribuované evoluční algoritmy**

Zásady pro vypracování:

1. Vypracujte literární rešerši na téma programovacího modelu MapReduce a paralelizace evolučních algoritmů a jejich paralelizace pomocí ostrovních modelů.
2. Vytvořte framework, umožňující spouštění evolučních algoritmů v ostrovních modelech na distribuované architektuře.
3. Implementujte pomocí vytvořeného frameworku genetický algoritmus, diferenciální evoluci a algoritmus SOMA.
4. Vytvořte vzorové aplikace, demonstrující funkčnost frameworku i naprogramovaných evolučních algoritmů na vybraných testovacích funkcích.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **VENNER, Jason.** Pro HADOOP. United States of America : Apress, 2009. 407 s. ISBN 978-1-4302-1942-2.
2. **LÄMMEL Ralf.** Google's MapReduce Programming Model ? Revisited. 42 s.
3. **OPLATKOVÁ, Zuzana, OŠMERA, Pavel, ŠEDA, Miloš, VČELAŘ, František, ZELINKA, Ivan.** Evoluční výpočetní techniky – principy a aplikace, 2008. 536 s. ISBN 80-7300-218-3.
4. **ALBA, Enrique, TROYA, José M.** A Survey of Parallel Distributed Genetic Algorithms. 1999. 32s.
5. **JIN, Chao, VECCHIOLA, Christian, RAJKUMAR, Buyya.** MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms. 2008, 8s.
6. **VERMA, Abhishek, LLORA, Xavier, GOLDBERG, David, CAMPBELL, Roy.** Scaling Genetic Algorithms using MapReduce. 2009, 6s.
7. **Distributed SOMA Algorithm in MapReduce framework. PAVLECH, Michal.** Sborník konference: Mezinárodní Masarykova konference pro doktorandy a mladé vědecké pracovníky. 1. Hradec Králové : [s.n.], 2010. s. 1380. ISBN 978-80-86703-41-1, ETTN 042-10-10003-11-4.
8. **Global Optimization : Methods and Codes [online].** 2007 [cit. 2011-01-05]. Test Functions for Unconstrained Global Optimization. Dostupné z WWW: (http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm).

Vedoucí bakalářské práce:

Ing. Michal Pavlech

Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce:

25. února 2011

Termín odevzdání bakalářské práce:

7. června 2011

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Hlavným cieľom tejto práce je vytvorenie programového frameworku slúžiaceho na tvorbu distribuovaných evolučných algoritmov v jazyku Java.

V teoretickej časti sú krátko popísané použité softwarové technológie a evolučné algoritmy použité na testovanie funkčnosti programu.

Praktická časť popisuje princíp fungovania a verejné funkcie frameworku, popísané sú tiež triedy ktoré programuje jeho užívateľ. Uvedená je aj vzorová aplikácia s krátkym popisom jednotlivých častí kódu. Ďalej je otestovaná jeho funkčnosť a účinky rôznych nastavení na priebeh výpočtov a konečných výsledkov.

Klíčová slova:

Optimalizácia, distribuované výpočty, evolučné algoritmy, ostrovné algoritmy, framework

ABSTRACT

The main objective of this work is to create a programming framework for creating distributed evolutionary algorithms in Java.

The theoretical part shortly describes used software technologies and evolutionary algorithm used to test the program.

The practical part describes principles of operation and public functions of the framework. Classes programmed by user are also described. An exemplary application with short description of individual parts of code is also shown. Functionality and effects of various settings on the process and final results are also tested.

Keywords:

Optimization, distributed computing, evolutionary algorithms, island algorithms, framework

Touto cestou by som chcel poďakovať vedúcemu mojej bakalárskej práce Ing. Michalovi Pavlechovi za odborné vedenie, rady, pripomienky a možnosť pracovať na zaujímavom projekte.

"Artificial intelligence usually beats natural stupidity."

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST	10
1 EVOLUČNÉ ALGORITMY	11
1.1 OBECNÝ CYKLUS EVOLUČNÉHO ALGORITMU	12
1.2 PARALELNÉ EVOLUČNÉ ALGORITMY	13
1.2.1 Globálna paralelizácia	13
1.2.2 Coarse grained.....	13
1.2.3 Fine grained.....	14
1.2.4 Hybridná metóda	15
1.3 GENETICKÉ ALGORITMY	15
1.3.1 Princíp algoritmu.....	15
1.3.2 Parametre genetických algoritmov	16
1.4 SOMA : SAMOORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS.....	16
1.4.1 Princíp algoritmu.....	16
1.4.2 Parametre SOMA	17
1.5 DIFERENCIÁLNÁ EVOLÚCIA	17
1.5.1 Princíp algoritmu.....	18
1.5.2 Varianty diferenciálnej evolúcie.....	18
1.5.3 Parametre diferenciálnej evolúcie	18
2 MAP/REDUCE	20
2.1 HADOOP	22
2.1.1 Architektúra.....	22
2.1.2 Hadoop Core MapReduce	22
II PRAKTICKÁ ČÁST	23
3 IMPLEMENTÁCIA OSTROVNÝCH ALGORITMOV DO PROGRAMOVACIEHO MODELU MAP/REDUCE.....	24
4 UŽÍVATELSKÉ TRIEDY.....	26
4.1 ALGORITMUS PUBLIC CLASS USERCLASS EXTENDS ALGORITHM.....	26
4.1.1 Dedené premenné:	26
4.1.2 Povinné funkcie:.....	26
4.1.2.1 public Individual generation() :.....	26
4.1.2.2 public void newRound();	27
4.1.2.3 public void setParameter(String configName, float value) throws IOException;	27
4.1.2.4 public Population getPopulation();	27
4.1.2.5 public void setPopulation(Population population_);	27
4.2 FITNESS FUNKCIA PUBLIC CLASS FUNCTION EXTENDS FITNESSFUNCTION	27
4.2.1 Dedené premenné:	27
4.2.2 Povinné funkcie:.....	28

4.2.2.1	public double evaluate(DoubleWritable[] s); public double evaluate(double[] genotype);	28
5	FRAMEWORK	29
5.1	VEREJNÉ FUNKCIE	29
5.1.1	Deaf()	29
5.1.2	addSubpopulationsConfig()	30
5.1.3	setTopology()	30
5.1.4	setImmigrationMethod()	30
5.1.5	setEmigrationMethod()	31
5.1.6	setVerbose	31
5.1.7	setRepeatedTest.....	31
5.1.8	setWriteHistory	31
5.1.9	setHistoryPath	31
5.1.10	setMapTasks.....	32
5.1.11	run	32
5.2	TOPOLOGIE.....	32
5.2.1	Predefinované topológie.....	32
5.2.2	Vytváranie nových topológií	33
5.3	METÓDY MIGRÁCIE	34
6	VZOROVÁ APLIKÁCIA.....	35
6.1	SPÚŠŤANIE PROGRAMU	37
7	VÝSTUP VÝSLEDKOV.....	38
7.1	“VERBOSE“ MÓD	38
7.2	TEXTOVÝ VÝSTUP	38
8	UKÁŽKA VPLYVU NASTAVENÍ NA PRIEBEH VÝPOČTOV.....	39
8.1	PRIEBEH VYHLADÁVANIA GLOBÁLNEHO MINIMA.....	39
8.1.1	Griewank	40
8.1.2	Rastrigin	40
8.1.3	Schwefel.....	41
8.2	UKÁŽKA VPLYVU NASTAVENÍ NA PRIEBEH VÝPOČTOV.....	41
8.2.1	Vplyv počtu migrujúcich jedincov	42
8.2.2	Vplyv topológie ostrovov.....	43
8.2.3	Vplyv parametrov.....	43
	ZÁVĚR	45
	ZÁVĚR V ANGLIČTINĚ.....	46
	SEZNAM POUŽITÉ LITERATURY.....	46
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	48
	SEZNAM OBRÁZKŮ	49
	SEZNAM GRAFŮ	50

ÚVOD

Počiatky optimalizácie siahajú až ku C. F. Gaussovi a jeho gradientnej metóde, na začiatok jej rozmachu by sme však nemuseli ísť tak ďaleko do histórie. Jej ďalší veľký krok, lineárne programovanie, sa zrodilo až počas druhej svetovej vojny. Napriek tomu ako by jeho názov naznačoval, nejednalo sa o programovanie počítačov, tie prišli na pomoc optimalizácii až s ich ďalším postupným vývojom. Niektoré typy algoritmov, napríklad iteračné, sú jednoducho prevediteľné do počítačového programu.

Tak ako to u techniky býva, aj v tomto obore sa človek začal snažiť napodobovať prírodu. V tomto prípade priamo jeden z najzákladnejších aspektov života ako celku, evolúciu. Vznikajú rôzne algoritmy inšpirované rôznymi prírodnými javmi (kríženie a mutácia v genetických algoritmoch, chovanie sa mravcov pri hľadaní potravy v „Ant colony optimization“ algoritme).

Napriek obrovskej rýchlosti nárastu výkonu počítačov sú zložitejšie problémy stále náročné na výpočtový výkon počítačov. Tento problém sa snažia riešiť ostrovné modely evolučných algoritmov. Tie sú opäť inšpirované prírodou, konkrétnejšie migráciami medzi ostrovmi a tým prenášaním nového genetického materiálu do geneticky uzatvorenej populácie čím do nej prináša väčšiu rôznorodosť. V jazyku optimalizácie to znamená, že môže posunúť populáciu bližšie k hľadanému extrému, poprípade ju dokonca vytrhnúť z lokálneho extrému, alebo riešiť problémy, na ktoré nie sú štandardné modely vhodné. [1]

Cieľ frameworku, ktorý bude výsledkom tejto práce je vytvárať práve takéto ostrovné prostredie pre populácie, ktoré budú optimalizovať užívateľom definované algoritmy.

I. TEORETICKÁ ČÁST

1 EVOLUČNÉ ALGORITMY

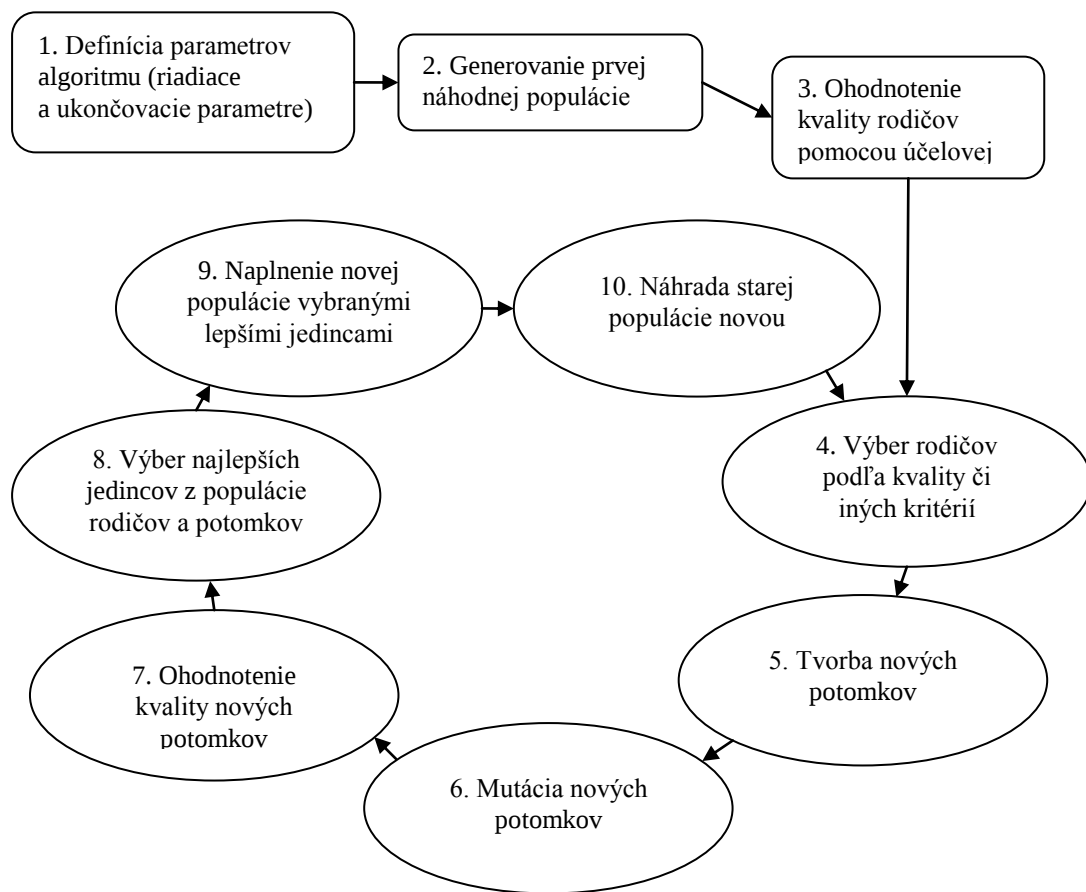
Evolučné výpočtové techniky sú numerické algoritmy, ktoré vychádzajú zo základných princípov Darwinovej a Mendelovej teórie evolúcie. Jeho hlavnou ideou je predávanie rodičovského genómu novým potomkom a následné uvoľnenie životného priestoru potomkom. Existujú aj výnimky, ktoré sa týmto neriadia, ale sú nimi inšpirované, tie sa nazývajú ako algoritmy patriace k evolučným technikám.

Jedným z dôvodov popularity evolučných algoritmov je aj to, že pri správnom aplikovaní dokážu nahradiť človeka. [2]

V evolučných algoritmoch sa pracuje s populáciou (prípadne viacerými) obsahujúcou určitý počet jedincov. Pomocou tých sa v závislosti od konkrétneho algoritmu tvoria noví jedinci a ich súperením o miesto v populácii sa zachovávajú iba jedinci lepší, ako v populácii už existujúci starší jedinci.

Každý jedinec predstavuje možné riešenie optimalizovaného problému pričom jeho genotyp určuje konkrétne parametre riešenia, teda jeho pozíciu v n-rozmernom priestore definičného oboru účelovej (optimalizovanej) funkcie.

1.1 Obecný cyklus evolučného algoritmu



Obr. 1 : Obecný cyklus evolučného algoritmu (bez ukončenia).

1. Vymedzenie parametrov evolúcie, každý algoritmus musí byť definovaný parametrami, ktoré riadia beh algoritmu alebo ju správne ukončí (napr. maximálny počet generácií). V tomto bode sa zároveň určuje účelová funkcia/vhodnosť (fitness/cost function).
2. Generovanie prvej populácie, tj. vektoru jedincov. Každý jedinec má počet vlastností predstavovaný vektorom čísiel v rozsahu danom definičným oborom konkrétneho problému, ich počet je rovnaký ako počet parametrov účelovej funkcie, tým každý jedinec predstavuje možné konkrétne riešenie problému.
3. Všetci jedinci sa ohodnotia pomocou definovanej účelovej funkcie a každému je priradená : a) priama hodnota účelovej funkcie, alebo b) vhodnosť, čiže upravená (normalizovaná) hodnota účelovej funkcie.

4. Výber rodičov podľa ich vhodnosti/ hodnoty účelovej funkcie, prípadne podľa ďalších kritérií.
5. Krížením rodičov vznikajú potomkovia. Algoritmus kríženia je daný konkrétnym evolučným algoritmom.
6. Potomkovia sú mutovaní, tj. sú pozmenení náhodným procesom.
7. Ohodnotenie nových jedincov, tak ako hodnotenie rodičov v kroku 3.
8. Výber najlepších jedincov.
9. Vybraní jedinci zaplňajú novú populáciu.
10. Nahradenie starej populácie za novú, ktorá sa vymazáva/dalej nepoužíva.

V prípade neukončenia algoritmu dosiahnutím maximálneho počtu generácií, alebo dosiahnutím iného ukončovacieho parametru sa opakujú kroky 4 až 10.

1.2 Paralelné evolučné algoritmy

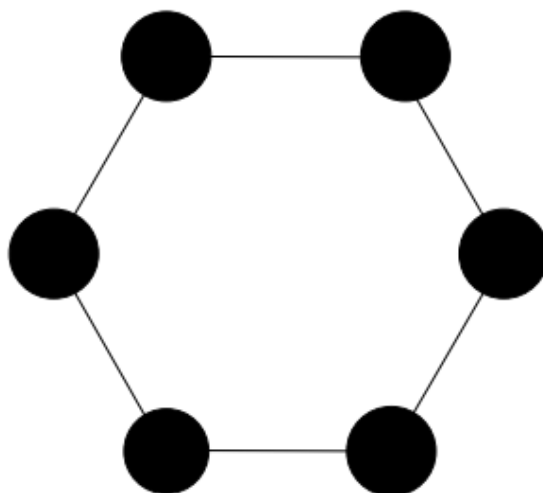
Evolučné algoritmy sú vysoko paralelizovateľné procesy. Ich paralelizáciu je možné rozdeliť do štyroch skupín.

1.2.1 Globálna paralelizácia

Základom je paralelizácia ohodnocovania všetkých jedincov a aplikovania jednotlivých genetických operátorov. Každý jedinec má možnosť kríženia s každým ďalším jedincov z celej populácie (neexistujú žiadne subpopulácie). Na rozdiel od ostatných spôsobov nemení správanie sa pôvodného algoritmu z ktorého vychádza.

1.2.2 Coarse grained

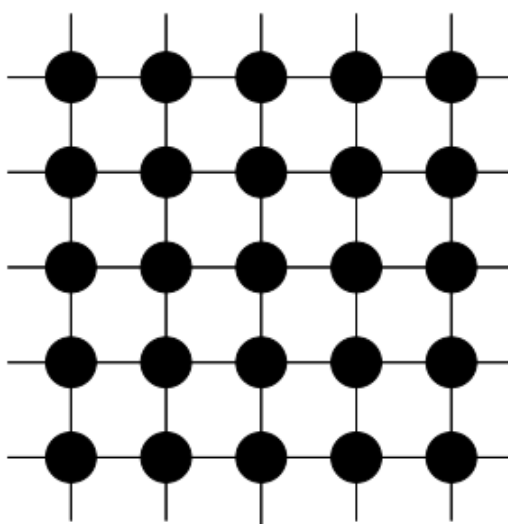
Doslovný preklad tohto výrazu je hrubo „zrnný model“. Ako veľkosť „zrna“ je pri paralelných aplikáciách označovaný pomer medzi časom, ktorý algoritmus spotrebuje na samotný výpočet a časom, ktorý spotrebuje na komunikáciu. Ak je tento pomer vysoký, tak je algoritmus označovaný ako coarse grained. Jedinci nie sú schopní kríženia s ľubovoľným ďalším jedincom, ale evolúcia prebieha paralelne v niekoľkých subpopuláciách medzi ktorými prebieha migrácia jedincov.



Obr. 2 : Schéma coarse grained algoritmu. [3]

1.2.3 Fine grained

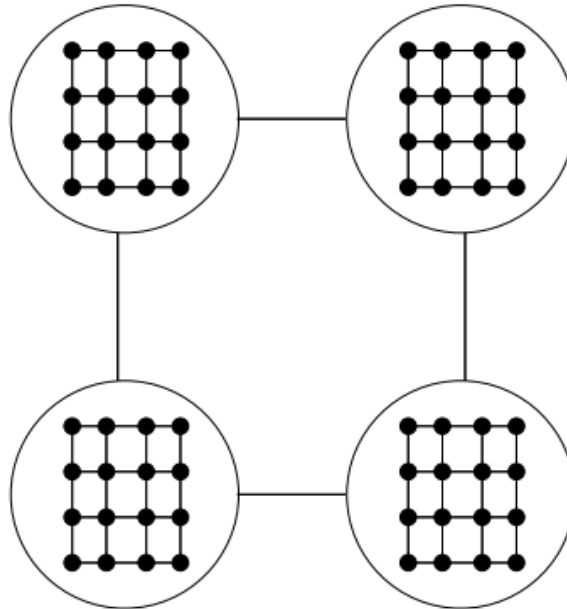
Základem tohto modelu je rozdelenie celej populácie na veľké množstvo malých subpopulácií, pričom ideálnym stavom je aby každý jedinec mal pridelenú jednu výpočtovú jednotku. Tento model je vhodný na masívne paralelné počítače.



Obr. 3 : Schéma coarse grained algoritmu. [3]

1.2.4 Hybridná metóda

Používa určitú kombináciu predchádzajúcich troch modelov.



Obr. 4 : Coarse grained - fine grained hybrid. [3]

1.3 Genetické algoritmy

Genetické algoritmy sú jedny z najznámejších evolučných techník. Často sa stáva, že slová genetické algoritmy sú zamenené za evolučné algoritmy, čo nie je správne keďže sú triedou algoritmov vychádzajúcich z klasického genetického algoritmu. Ten bol po prvý krát predstavený Johnom Hollandom v roku 1975. Sú založené na princípoch evolúcie v prírode popísané Charlesom Darwinom. Z toho dôvodu je aj väčšina terminológie prevzatá z biológie.

1.3.1 Princíp algoritmu

Po počiatočnom vygenerovaní populácie sa postupne prechádza celá populácia. Pre každého jedinca sa vyberá ďalší, tí spolu tvoria rodičov nového jedinca. Na výber druhého rodiča existuje viacero spôsobov, napr.: náhodné, ruletové, elitistické...

Nový jedinec vzniká skrížením génov jeho rodičov, kríženia existuje taktiež viacero druhov. Jednobodové kríženie rozdeľuje genotypy rodičov v jednom bode a ich časti sa spoja do

nového genotypu, čím vytvoria nového jedinca. Podobne funguje dvojbodové kríženie s dvomi deliacimi bodmi. Pri uniformnom krížení sa gény vyberajú náhodne.

Ďalším krokom je mutácia. Pri nej jeden alebo viacero génov mení svoju hodnotu. Postupným vytváraním nových jedincov vzniká nová generácia a cyklus pokračuje až do ukončenia.

1.3.2 Parametre genetických algoritmov

- Prah mutácie (M) $\in (0, 1)$:

Určuje pravdepodobnosť mutácie.

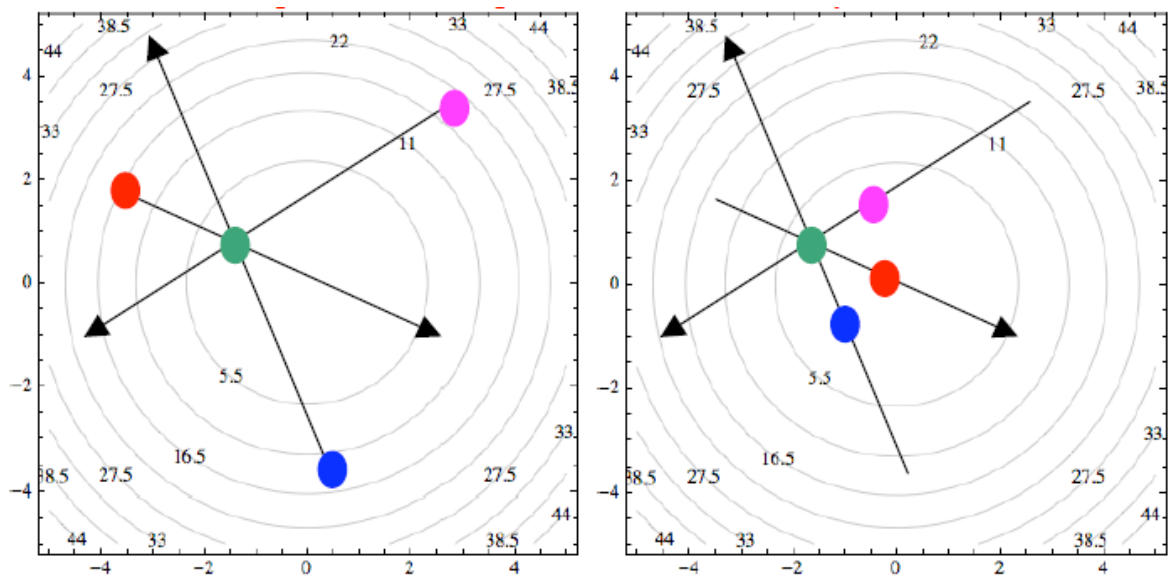
1.4 SOMA : SamoOrganizující se Migrační Algoritmus

Existuje od roku 1999, vzhľadom na to, že pracuje s populácia podobne ako napr. genetické algoritmy a výsledok po jednom evolučnom cykle je taktiež rovnaký, dá sa radiť medzi evolučné algoritmy, aj napriek tomu že počas behu nedodržiava filozofiu vytvárania nových potomkov ako je tomu u klasických evolučných algoritmov. Biologická analógia tohto algoritmu by bola skôr háremová tvorba potomkov než klasický výber rodičov z populácie. Presnejšie je však radenie medzi algoritmy memetické alebo hejnové (swarm intelligence). [2]

1.4.1 Princíp algoritmu

SOMA pracuje v cykloch zvaných migračné kolá (tie však nemajú nič spoločného s migráciami medzi ostrovmi). Tie majú rovnakú úlohu ako generácie v genetických algoritmoch. Počas migračných kôl sa na rozdiel od generácií netvoria noví jedinci, ale pôvodný sú premiestňovaný do novej lepšej pozície pomocou sekvencie pozícií. Smer premiestňovania je určovaný pomocou pozície najlepšieho jedinca populácie.

Smer premiestňovania môže určovať aj perturbancia (tzv. rušenie pohybu). Tá by sa dala prirovnať k mutácii pri genetických algoritmoch. Náhodne sa generuje perturbančný vektor, podľa ktorého sa určuje v ktorých rozmeroch sa jedinec bude pohybovať.



Obr. 5 : Princíp SOMA – a) pred migráciou, b) po migrácii na nových pozíciách.[2]

1.4.2 Parametre SOMA

- Dĺžka cesty (pathLength) $\in (1, 5)$:

Určuje ako ďaleko sa aktívny jedinec zastaví od vedúceho jedinca

- Krok (step) $\in (0.11, \text{pathLength})$:

Určuje hustotu, s akou bude prehľadávaný priestor

- PRT $\in (0, 1)$:

Určuje pravdepodobnosť perturbancie (narušenia) smeru migrácie jedinca.

1.5 Diferenciálna evolúcia

Je veľmi podobná genetickému algoritmu (vytváranie nového jedinca, opakovanie generácií, ...). Pôvod má v tzv. genetickom žíhaní vyvinutým Kenethom Priceom. Jej princíp spočíva v pričítaní rozdielu dvoch jedincov k tretiemu. Tým vzniká tzv. šumový „noisy“ jedinec, ktorý po skrížení s aktívnym jedincom vytvára nového jedinca. V prípade, že má lepšiu hodnotu účelovej funkcie, nahrádza aktívneho jedinca v populácii.

1.5.1 Princíp algoritmu

Diferenciálna evolúcia pracuje s populáciami podobne ako genetické algoritmy, rozdiel je vo vytváraní nových jedincov. Po počiatocnom vygenerovaní sa postupne prechádza populáciou a u každého jedinca prebieha evolučný cyklus. Jeho prvým krokom je výber ďalších troch (alebo podľa varianty iného počtu) jedincov. Prvý dva sa od seba odčítajú a tým vzniká **diferenčný vektor**. Ten sa vynásobí mutačnou konštantou F , čím vzniká **váňovaný diferenčný vektor**. Pričíta sa k tretiemu jedincovi, tým dostaneme **šumový vektor**. Podľa náhodne vygenerovaného čísla a jeho porovnania s prahom kríženia CR sa vyberajú jednotlivé hodnoty genotypu a tým sa vytvorí **skúšobný vektor**. Na pozíciu v populácii sa umiestni lepší z jedincov, skúšobný alebo aktívny (pôvodný).

1.5.2 Varianty diferenciálnej evolúcie

Algoritmy diferenciálnej evolúcie sa môžu líšiť hlavne vo výpočte šumového (noisy) jedinca. Uvedené sú príklady niektorých z mnohých variant, tieto budú použité aj na testovanie frameworku.

DE/rand/1/bin :

$$v = x_{r1,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (1.1)$$

DE/rand/2/bin :

$$v = x_{r5,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (1.2)$$

DE/best/1/bin :

$$v = x_{best,j}^G + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (1.3)$$

DE/best/2/bin :

$$v = x_{best,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (1.4)$$

1.5.3 Parametre diferenciálnej evolúcie

- Mutačná konštanta (F) $\in (0, 2)$:
Určuje pravdepodobnosť mutácie.
- Prah kríženia (CR) $\in (0.11, \text{pathLength})$:

Určuje pravdepodobnosť s akou budú gény dedené z rodiča aktívneho alebo vygenerovaného šumového (noisy) jedinca.

2 MAP/REDUCE

MapReduce je programovací model vyvinutý a patentovaný spoločnosťou Google. Jeho účelom je spracovanie veľkého množstva dát (rádovo terabyty) na výpočtových clusteroch s vysokou mierou paralelizmu. MapReduce bol inšpirovaný funkciami *map* a *reduce* používanými vo funkcionálnych jazykoch, napríklad v jazyku LISP . [4]

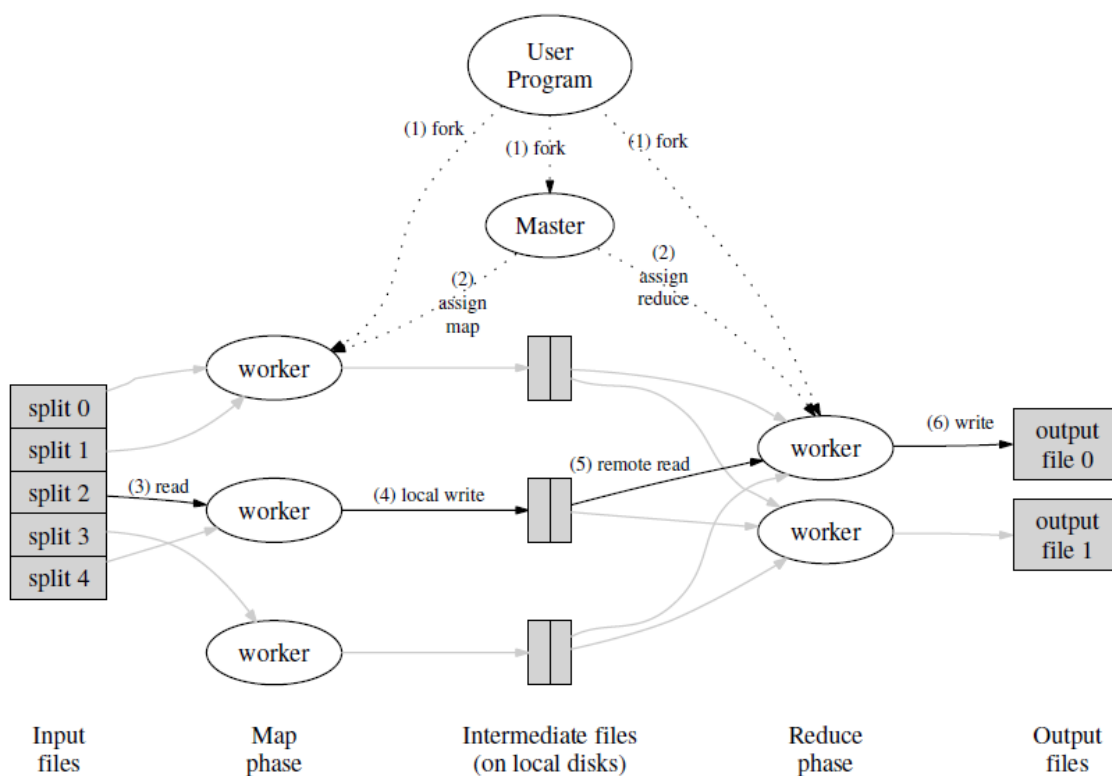
Volanie funkcie *map* je distribuované na niekoľko samostatne bežiacich počítačov. Distribúcia je realizovaná pomocou rozdelenia vstupných dát na M častí. Takto rozdelené časti môžu byť spracované paralelne na rôznych strojoch. Volania funkcie *Reduce* sú distribuované pomocou rozdelenia priestoru kľúčov na R častí, najčastejšie pomocou hashovacej funkcie: hash (kľúč mod R). Rozdeľovacia funkcia, a teda aj počet častí, môžu byť definované používateľom.

Na Obr. 6 je zobrazený celkový priebeh jednej MapReduce úlohy v implementácii firmy Google. Pri volaní MapReduce úlohy prebieha nasledujúci sled činností [5]:

- Framework, tvoriaci základnú funkčnosť MapReduce, rozdelí vstupné súbory na M častí. Veľkosť jednotlivých častí je kontrolovateľná užívateľom. Framework potom spustí kópie programu na oddelených počítačoch.
- Jedna kópia programu má zvláštne určenie, nazýva sa master a stará sa o pridelovanie úloh pre ostatné uzly a koordináciu práce celku. Ostatné kópie sú pracovné uzly, ktorým master priradzuje úlohy. Master musí postupne nečinným uzlom priradiť M mapovacích úloh a R redukovacích úloh.
- Pracovný uzol, ktorý má vykonávať mapovaciu funkciu na začiatku vykonávania prečíta obsah príslušnej časti vstupu. Analyzuje dvojice <kľúč, hodnota> zo vstupných dát a každú takúto dvojicu posunie na spracovanie príslušnej map funkcií (definovaná užívateľom). Prechodné dvojice, ktoré sú výsledkom map funkcie sú uložené do vyrovnávacej pamäte.
- Dvojice uložené vo vyrovnávacej pamäti sú v pravidelných intervaloch zapísané na lokálne disky na jednotlivých pracovných uzloch. Umiestnenie týchto dvojíc je poslané mastrovi, ktorý zodpovedá za preposlanie umiestnenia na uzly vykonávajúce redukovacie úlohy.
- Keď redukovací uzol dostane od mastra správu o umiestnení dát, použije vzdialené volanie procedúr na načítanie uložených dát. Po získaní všetkých prechodných dát

ich zoradí podľa prechodných kľúčov tak, aby všetky dáta s rovnakým kľúčom boli zoskupené spolu. Zoradenie je dôležité, pretože pri typickom použití je na vstupe jedného redukovacieho uzla množstvo dvojíc s rozdielnymi kľúčmi.

- Rovnako ako pri mapovaní, redukovací uzol prechádza cez zoradené prechodné dáta a pre každý jedinečný kľúč volá reduce funkciu (definovaná užívateľom), ktorej pošle kľúč a súbor zodpovedajúcich hodnôt. Hodnoty sú poskytované pomocou iterátorov, takže je možné pracovať aj s dátami, ktoré sú príliš veľké, aby sa vošli do pamäte.
- Výsledok Reduce funkcie sa pripojí do konečného výstupného súboru pre túto časť prechodných dát. Bežnou praxou je, že výstupom je len jedna výsledná hodnota z každého volania funkcie Reduce, nie je to však podmienkou.
- Po skončení všetkých úloh je program ukončený, respektíve je ďalej vykonávaný program, z ktorého bolo MapReduce volané.



Obr. 6 : Diagram vykonávania MapReduce [6]

2.1 Hadoop

Apache Hadoop je framework pre aplikácie bežiacie na veľkých clusteroch postavených z bežného hardware. Poskytuje aplikáciám ako spoľahlivosť, tak aj prenos dát. Hadoop implementuje výpočtový model Map/Reduce, v ktorom je aplikácia rozdelená na čiastkové práce, z ktorých každá môže byť vykonaná na ktoromkoľvek uzly clusteru. Okrem toho poskytuje distribuovaný súborový systém (HDFS) ktorý uchováva dáta v clustery, čím dosahuje veľkú celkovú dátovú priepustnosť v rámci celého clusteru. Map/Reduce a distribuovaný súborový systém sú stavané tak, aby automaticky zvládali zlyhanie jednotlivých uzlov. [7]

2.1.1 Architektúra

Hadoop Map/Reduce framework implementuje master/slave architektúru. Pozostáva z jedného master serveru (plánovač úloh, jobtracker) a viacerých slave serverov (tasktrackers). Interakcia medzi užívateľom a frameworkom nastáva v plánovači úloh, užívateľ mu zadá map/reduce úlohy a tie sú zaradené do fronty a postupne sa vykonávajú. Jobtracker zaobstaráva priradovanie map a reduce úloh tasktrackerom.

2.1.2 Hadoop Core MapReduce

Framework Hadoop poskytuje užívateľovi prostredie pre ovládanie a vykonávanie *map* a *reduce* úloh na clustery počítačov. Užívateľ musí definovať: [8]

- Umiestnenie vstupu v HDFS
- Umiestnenie výstupov v HDFS
- Vstupný formát
- Výstupný formát
- Triedu obsahujúcu *Map* funkciu

Nepovinná je trieda s *Reduce* funkciou, ktorá v tejto práci nie je využitá.

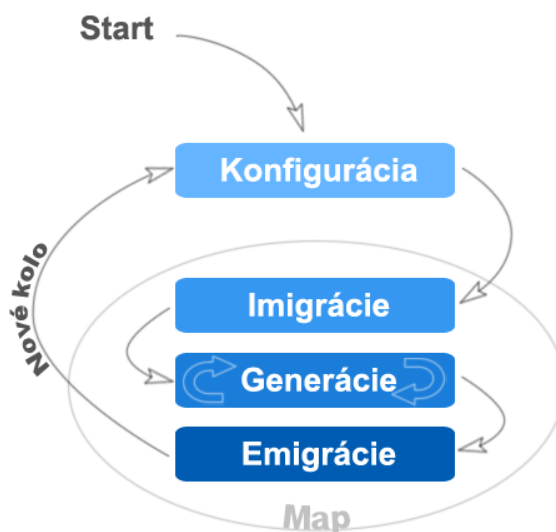
II. PRAKTICKÁ ČÁST

3 IMPLEMENTÁCIA OSTROVNÝCH ALGORITMOV DO PROGRAMOVACIEHO MODELU MAP/REDUCE

Vzhľadom na to, že použitý framework Hadoop využíva programovací model Map/Reduce, bolo nutné upraviť model ostrovných algoritmov tak, aby bol spustiteľný ako Map/Reduce program. Tým sa vytvorí vyššia vrstva abstrakcie, skrývajúca samotné vnútorné prevedenie populácií, jedincov, ich migrácie medzi ostrovmi a hlavne samotnú distribúciu výpočtov.

Priebeh celého programu prebieha v kolách, po každom z nich medzi populáciami podľa nastavení migrujú jedinci alebo celé časti populácie. V každom kole prebieha niekoľko generácií. Jedna generácia je jeden beh daného algoritmu.

Celý tento proces prebieha iba v rámci časti Map. Reduce sa napriek počiatočnému plánu použiť ho pre migráciu nepoužíva. Migrácie je uskutočňovaná pomocou HDFS, zápis a načítanie migrujúcich populácií prebieha taktiež v časti Map.



Obr. 7 : Priebeh programu

- Po nastavení a spustení sa konfiguruje Hadoop framework, ten následne inicializuje a spúšťa triedu Map na podradených počítačoch (tasktrackers).
- S výnimkou prvého kola, kde neexistujú žiadne migrujúce populácie, sa do populácie konkrétneho ostrova prijímajú migrujúci jedinci. Spôsob ich prijímania určuje enumeratívny dátový typ `Population.acceptImmigrantsMethod`. Populácie sa načítavajú z HDFS. Počet imigrujúcich populácií nie je obmedzený.

- Dalším krokom je samotné šľachtenie populácie, to prebieha cyklicky, počet cyklov je nastavovaný v konštruktore frameworku. To programovo odpovedá volaniu metódy `generation()` z triedy užívateľom definovaného algoritmu.
- Posledným krokom v rámci funkcie `Map` je emigrácia jedincov. Spôsob emigrácie určuje enumeratívny dátový typ `Population.expelEmigrantsMethod`. Migrujúca populácia je aj s číslom cieľového ostrova zapísaná do HDFS. Počet cieľových ostrovov nie je obmedzený, určuje ho topológia ostrovov. Emigruje vždy jedna populácia, ktorá sa klonuje pre každý cieľový ostrov.
- Až po dosiahnutie žiadaného počtu kôl sa celý cyklus opakuje znova.

Všetky výsledky sa postupne zapisujú na HDFS v binárnom aj textovom (pokiaľ bola táto možnosť povolená) formáte. Binárne súbory je možné použiť ako prvotné populácie, je však potrebné vypnúť generovanie prvej generácie pomocou funkcie `setRepeatedTest(true)`.

4 UŽÍVATELSKÉ TRIEDY

Framework z definície predstavuje abstrakciu určitého systému. V tomto prípade užívateľ a zbavuje od starostí s distribúciou výpočtov (pomocou Hadoop) a práce s populáciami, jedincami a inými aspektmi evolučných algoritmov.

Užívateľ definuje iba jednotlivé triedy dediace zodpovedajúce triedy algoritmu. Konkrétne sú to :

- Evolučný algoritmus – dedí triedu *Algorithm*
- Účelovú funkciu – dedí triedu *FitnessFunction*

4.1 Algoritmus `public class userClass extends Algorithm`

Dedí triedu *Algorithm*. Jej funkciou je samotné hľadanie extrému funkcie. V mojej práci sú ako ukážky uvedené algoritmy SOMA, Diferenciálna evolúcia a Genetický algoritmus.

4.1.1 Dedené premenné:

Tieto premenné sú potrebné vo všetkých algoritmoch, preto sú vopred definované v rodičovskej triede.

```
int dimensions;    : Počet dimenzií účelovej funkcie
int popSize;       : Veľkosť populácie
FitnessFunction function; : Účelová funkcia
protected Population population; : Populácia
Individual bestIndividual; : Najlepší jedinec
double bestFitness = Double.MAX_VALUE; : Hodnota účelovej funkcie najlepšieho jedinca
```

4.1.2 Povinné funkcie:

4.1.2.1 `public Individual generation() :`

Funkcia v ktorej prebiehajú samotné operácie s jedincami za účelom nájdenia lepšej hodnoty účelovej funkcie. Volá sa v každom kole toľko krát, koľko užívateľ zadal generácií na kolo.

4.1.2.2 *public void newRound();*

Je volaná pred každou migráciou. Slúži na prenastavenie parametrov algoritmu. V prípade nemenných parametrov je možné ponechať túto funkciu prázdnu.

4.1.2.3 *public void setParameter(String configName, float value) throws IOException;*

Slúži na nastavovanie parametrov algoritmu definovaných konkrétne pre jeden ostrov. Nastavuje hodnotu parametru `value` s názvom `configName`.

Napr.:

```
if(configName.equals("name")) {  
    name = value;  
} else if ....
```

4.1.2.4 *public Population getPopulation();*

Vracia populáciu používanú pri behu algoritmu.

4.1.2.5 *public void setPopulation(Population population_);*

Nastavuje populáciu pre použite pri behu algoritmu.

4.2 **Fitness Funkcia** *public class Function extends FitnessFunction*

Predstavuje optimalizovanú funkciu.

4.2.1 **Dedené premenné:**

protected int specimenLength;

Určuje veľkosť jedinca a tým aj počet rozmerov definovanej funkcie.

protected double minRestrictions[],maxRestrictions[];

Ohraničujú definičný obor funkcie. N-tý prvok pola určuje ohraničenie v n-tom rozmere.

4.2.2 Povinné funkcie:

4.2.2.1 *public double evaluate(DoubleWritable[] s);*

public double evaluate(double[] genotype);

V tejto funkcii bude obsiahnutá už samotná optimalizovaná účelová funkcia.

Napr. Schwefelova funkcia :

```
double val = 0;
for (int i = 0; i < specimenLength; i++) {
    val += -1 * s.get(i * Math.sin(Math.sqrt(Math.abs(s.get(i)))));
}
return val;
```

5 FRAMEWORK

V tejto časti sú popísané jednotlivé súbory frameworku, ich funkcie a parametre.

5.1 Verejné funkcie

5.1.1 Deaf()

Súbor : Deaf.java

```
public Deaf (Class<? extends FitnessFunction> fitFunction, int D, Class<? extends Algorithm> algorithm, int generationsPerRound, int islandCount, int populationSize, int migratingIndividuals)
```

Konštruktor frameworku, nastavuje väčšinu obecných premenných. Tie platia pre všetky ostrovy. Pre nastavenia vlastností jednotlivých ostrovov slúži trieda `addSubpopulationsConfig()`.

Parametre:

Class<? extends FitnessFunction> fitFunction : Implementácia triedy `FitnessFunction` obsahujúca optimalizovanú účelovú funkciu.

int D : Počet dimenzií funkcie z `fitFunction`

Class<? extends Algorithm> algorithm : Implementácia triedy `Algorithm`. Obsahuje výpočtový algoritmus pre optimalizáciu. Vid': **4.1 Algoritmus**

int generationsPerRound : Počet generácií (cyklov algoritmu)

int islandCount : Počet ostrovov algoritmu, zároveň nastavuje počet `MapTasks` pre Hadoop. V prípade potreby sa dá ich počet zmeniť pomocou funkcie `setMapTasks()`.

int populationSize : Veľkosť jednotlivých populácií.

int migratingIndividuals : Počet migrujúcich jedincov v každom kole.

5.1.2 addSubpopulationsConfig()

```
public void addSubpopulationsConfig(Class<? extends Algorithm> alg, int popSize, Map<String, Double> subpopParameters)
```

```
public void addSubpopulationsConfig(Class<? extends Algorithm> alg, Map<String, Double> subpopParameters)
```

Slúži na definície parametrov jednotlivých ostrovov. Pre ostrovy s nedefinovanými špecifickými parametrami sa použijú všeobecné parametre zadané v konštruktore.

Parametre:

Class<? extends Algorithm> alg : Implementácia triedy Algorithm.

int popSize : Veľkosť populácie na ostrove.

Map<String, Double> subpopParameters : Pole (pomocou Map collection) obsahujúce parametre špecifické pre použitý algoritmus. Pri inicializácii budú predané funkcii setParameter() ktorá ich spracuje. Vid' 4.1.2.3 setParameter

Napr.:

```
Map<String, Double> subpopParameters = new HashMap<String, Double>();  
subpopParameters.put("Param1", 10);  
subpopParameters.put("Param2", 0.2);  
deaf.addSubpopulationsConfig(DERandlbin.class, subpopParameters);
```

5.1.3 setTopology()

```
public void setTopology(Class<? extends Topology> top)
```

Nastavenie topológie ostrovov tj. spôsob výberu cieľového ostrova pre emigrujúcich jedincov, vid': 6.2 Topológia.

5.1.4 setImmigrationMethod()

```
public void setImmigrationMethod(Population.acceptImmigrantsMethod met)
```

Určuje akým spôsobom budú prijímaný imigrujúci jedinci do populácie.

Parametre:

Population.acceptImmigrantsMethod met: Jedna z preddefinovaných metod. Vid' 5.3 *Metódy migrácie*.

5.1.5 setEmigrationMethod()

```
public void setEmigrationMethod(Population.expelEmigrantsMethod met)
```

Určuje spôsob výberu jedincov migrujúcich na iný ostrov. Cieľový ostrov určuje topológia ostrovov, vid' : 5.2 *Topológia*.

Parametre:

Population. expelEmigrantsMethod met: Jedna z preddefinovaných metod. Vid' 5.3 *Metódy migrácie*.

5.1.6 setVerbose

```
public void setVerbose(boolean verb)
```

Pri zapnutom verbose móde sa vypisujú jedinci populácií pred a po prejdení nastaveného počtu generácií v každom kole.

5.1.7 setRepeatedTest

```
public void setRepeatedTest (boolean b)
```

Pri zapnutí tejto voľby sa prvá generácia negeneruje, ale načítava zo súborov vo vstupnej zložke. Prvá generácia sa nevymazáva ale necháva pre použitie v ďalších kolách opakovaného testu.

5.1.8 setWriteHistory

```
public void setWriteHistory(boolean b)
```

Nastavuje konverziu populácií do textovej formy na konci programu.

5.1.9 setHistoryPath

```
public void setHistoryPath(String r)
```

Nastavenie zložky pre ukladanie textových reprezentácií populácií.

5.1.10 setMapTasks

```
public void setMapTasks(int m)
```

Počet Map funkcií Hadoop-u, mal byť menší alebo rovný počtu uzlov v clustery. Nie je nutné nastavovať, implicitne je nastavený podľa počtu ostrovov.

5.1.11 run

```
public Individual run(int rounds)
```

Spúšťa samotnú evolúciu. Jediným parametrom je počet kôl/migrácií, všetky ostatné parametre sa určujú v konštruktore frameworku alebo pomocou jeho verejných funkcií. Výsledkom je celkovo najlepší jedinec (s najmenšou hodnotou účelovej funkcie).

5.2 Topológia

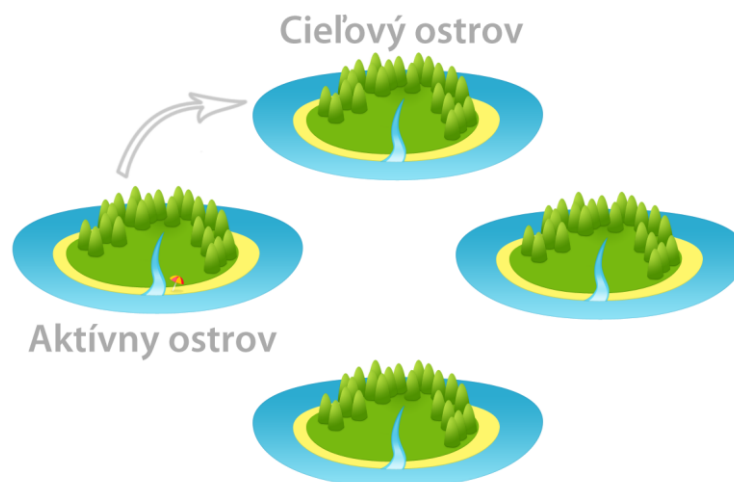
Súbor : Topology.java

Topológia ostrovov určuje ich imaginárne rozloženie a tým spôsob akým migrujúce populácie vyberajú cieľ svojej cesty. Framework umožňuje užívateľovi vytvárať vlastné topológie, alebo vybrať z 3 jednoduchých predom definovaných topológií. Pokiaľ užívateľ neurčí inak, používa sa topológia *Ring*.

5.2.1 Predefinované topológie

Ring :

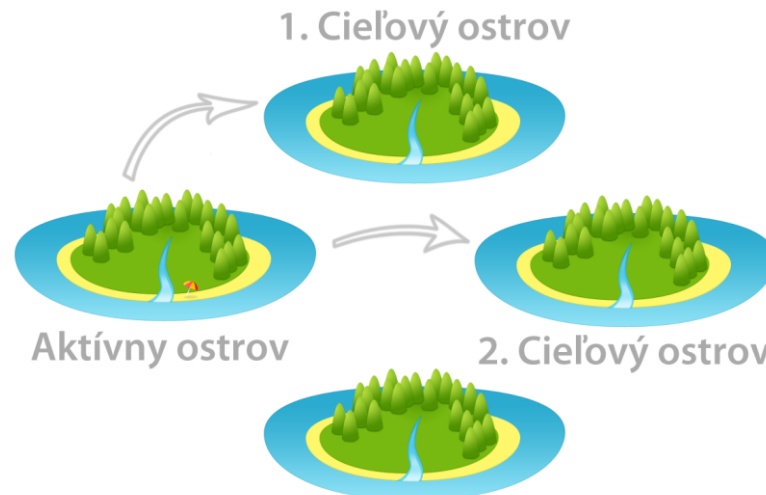
V kruhovej topológii migrujú jedinci iba na najbližší susedný ostrov.



Obr. 8 : Znázornenie kruhovej topológie

Ring_1_2:

Rozloženie umožňuje migráciu jednej populácie na viacero ostrovov, súčasná implementácia migrácie umožňuje klonovanie populácie pre každý cieľový ostrov.



Obr. 9 : Znázornenie topológie v ktorej jedinci migrujú na dva následné ostrovy

Random :

V topológii *Random* migruje populácia na náhodne vybraný ostrov.

5.2.2 Vytváranie nových topológií

Užívateľom definovaná trieda topológie musí dediť triedu *Topology*.

```
public static class Ring extends Topology
```

V prípade potreby vlastného konštruktoru musí daný konštruktor definovaný ako `public` a prijímať jeden parameter `int`, ktorý predstavuje celkový počet ostrovov a musí volať nadradený konštruktor s tým istým parametrom.

```
public Ring(int islands) {
    super(islands);
    ....
}
```

Hlavnou funkciou triedy je funkcia `get`, prijímajúca jeden celočíselný parameter predstavujúci číslo aktuálneho ostrova.

Návratová hodnota je pole celých čísel označujúcich cieľové ostrovy. V prípade migrácie na jediný ostrov to je pole iba o jednom prvku. Nasledujúci príklad by smeroval všetky migrujúce populácie na ostrov 1.

```
public int[] get(int n) {  
    int[] arr = {1};  
    return arr;  
}
```

5.3 Metódy migrácie

Tak ako topológia určuje cieľ migrujúcej populácie, tak metódy migrácie určujú spôsob akým budú migrujúci jedinci prijatý alebo vybraný z populácie ostrovu. Tento spôsob určujú enumeratívne dátové typy z triedy *Population* *acceptImmigrantsMethod* a *expelEmigrantsMethod*.

acceptImmigrantsMethod :

REPLACE_WORST – nahradí najhorších jedincov z cieľovej populácie.

REPLACE_BEST – nahradí najlepších jedincov z cieľovej populácie.

REPLACE_RANDOM – nahradí náhodných jedincov z cieľovej populácie.

NO_REPLACE – jedinci sa nenahrádzajú, prichádzajúci jedinci zväčšujú populáciu,

expelEmigrantsMethod :

EXPEL_BEST – emigrujú najlepší jedinci

EXPEL_RANDOM – emigrujú náhodný jedinci

6 VZOROVÁ APLIKÁCIA

V nasledujúcej vzorovej aplikácii je popísaný jednoduchý program využívajúci Deaf framework na optimalizáciu 100 dimenzionálnej Rastrigin funkcie. Evolúcia prebieha na troch ostrovoch, defaultná veľkosť populácie je 50 jedincov, dva z ostrovom majú však nastavenú špecifickú veľkosť populácie na 60 a 40 jedincov.

Celkový počet migrácií bude 20, medzi ktorými prebehne 50 generácií. Migruje vždy 10 jedincov.

```
int dimension = 100;
int rounds = 20;
int islandCount = 3;
int generationsPerRound = 50;
int defaultPopulationSize = 50;
int migratingIndividuals = 10;
```

Map kolekcia používaná pre nastavovanie parametrov jednotlivých ostrovov.

```
Map<String, Double> subpopParameters = new HashMap<String, Double>();
```

Vytvorenie inštancie hlavnej triedy frameworku, v konštruktoe sú definované nastavenia frameworku ako celku, ale aj obecné nastavenia ostrovov pre prípad nedefinovania nastavení pre konkrétny ostrov.

Postupne : optimalizovaná funkcia, počet dimenzií, optimalizačný algoritmus, počet generácií medzi migráciami, počet ostrovov, veľkosť populácie, počet migrujúcich jedincov.

```
Deaf deaf = new Deaf(FitnessFunctions.Rastrigin.class,
                    dimension,
                    SOMA.class,
                    generationsPerRound,
                    islandCount,
                    defaultPopulationSize,
                    migratingIndividuals);
```

Nastavenie pre jednotlivé ostrovy. Do hore definovanej Map kolekcie sa vkladajú nastavenia v tvare *map.put(„názovParametru“, hodnota)*. Pomocou metódy *addSubpopulationConfig(algoritmus, velkostPopulacie, mapaParametrov)*, kde veľkosť populácie nie je povinná. Po každom nastavení vyčistíme Map kolekciu pre ďalšie použitie.

```
subpopParameters.put("F", 0.5);
subpopParameters.put("CR", 0.2);
deaf.addSubpopulationsConfig(DEBest1bin.class, 60, subpopParameters);
subpopParameters.clear();
```

```
subpopParameters.put("PathLength", 2.0);
subpopParameters.put("Step", 0.11);
deaf.addSubpopulationsConfig(SOMA.class, subpopParameters);
subpopParameters.clear();
```

```
subpopParameters.put("M", 0.11);
deaf.addSubpopulationsConfig(GeneticAlgorithm_Best.class, 40,
subpopParameters);
subpopParameters.clear();
```

Nepovinné nastavenia frameworku sa nastavujú pomocou jeho funkcií vid': *5.1 Verejné funkcie*.

„Verbose“ mód je zapnutý, takže všetci jedinci sa budú vypisovať do systémového výstupu.

```
deaf.setVerbose(true);
```

Náhodne sa generuje prvá generácia.

```
deaf.setRepeatedTest(false);
```

Topológia ostrovov je Ring_1_2, tj. migrácia na dva nasledujúce ostrovy.

```
deaf.setTopology(Topologies.Ring_1_2.class);
```

Imigrujúci jedinci budú nahradzovať najhorších v cieľovej populácii.

```
deaf.setImmigrationMethod(Population.acceptImmigrantsMethod.REPLACE_WORST);
```

Emigrovať budú najlepší jedinci z konkrétnej populácie.

```
deaf.setEmigrationMethod(Population.expelEmigrantsMethod.EXPEL_BEST);
```

Nakoniec sa všetci jedinci zo všetkých kôl prepíšu do textového formátu.

```
deaf.setWriteHistory(true);
```

Po nastavení požadovaných parametrov sa samotná evolúcia spúšťa pomocou metódy *run(početKôl)*. Jej návratovou hodnotou je jedinec s najlepšou (najmenšou) hodnotou účelovej funkcie. Pri spúšťaní je potrebné odchytať výnimky *IOException*

a *ConfigurationException*. *IOException* výnimky sú takmer vždy dôsledkom chyby v Hadoop frameworku.

```
try {  
    Individual winner = deaf.run(rounds);  
    System.out.println("Best individual : \n" + winner);  
} catch (ConfigurationException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
} catch (IOException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
}
```

6.1 Spúšťanie programu

Celý program je nutné spúšťať pod Hadoop frameworkom, nespúšťa sa teda priamo, ale skompilovaný .jar súbor sa spúšťa pomocou programu *hadoop-0.20/bin/hadoop*. Jeho parametre budú :

- -jar : spúšťame skompilovaný .jar súbor.
- Cesta k .jar : napr. /home/username/java/deaf/dist/Main.jar
- Hlavná trieda : napr. myPackage.Main

Výsledný príkaz, za predpokladom že aktívnou zložkou je *bin* zložka Hadoop frameworku, bude teda:

```
./hadoop -jar /home/username/java/deaf/dist/Main.jar myPackage.Main
```

Framework používa na ukladanie dát HDFS, pre kopírovanie dát na lokálny súborový systém slúži príkaz *hadoop dfs -get filename.txt* [9]. Pre prehliadanie a zisťovanie stavu HDFS sa využíva webové rozhranie Hadoop frameworku prístupné cez webový browser na adrese Namenode-u s portom 50070 [10]. Napr.:

http://localhost:50070

7 VÝSTUP VÝSLEDKOV

Výpis jedince:

n) Value: *fitness - gene1 | gene2 | geneD*

n – index jedince.

fitness – vhodnosť jedince podľa účelovej funkcie.

gene1-geneD – výpis genómu jedince.

7.1 “Verbose“ mód

`deaf.setVerbose(true);`

Pri nastavení „verbose“ módu sú počas behu programu vypisované populácie pred a po migrácii. Tento mód slúži hlavne pre ukážku a kontrolu funkčnosti algoritmov, keďže pri väčšom počte dimenzií a rozsiahlejších populáciách sú ich výpisy príliš veľké, nečitateľné a výrazne spomaľujú beh programu.

V prípade presmerovania systémového výstupu do súboru je možné použiť tento mód aj na zber dát. Pre lepšie štruktúrovaný zber dát je vhodnejšie použiť textový výstup.

7.2 Textový výstup

`deaf.setWriteHistory(true);`

`deaf.setHistoryPath("history_path");`

Po skončení výpočtov sú pri povolenom nastavení „setWriteHistory“ všetky populácie vytvorené počas behu programu prekonvertované do textovej podoby, každá do vlastného textového súboru do zložky nastavenej pomocou funkcie „setHistoryPath“.

```
Population key: 3
Population size: 50
Best fitness: -4778.752929740184

Individual: 0 -2450.670154501361 439.718197 -278.996562 221.231659 407.368161
Individual: 1 1117.4034006697113 241.388109 -103.387850 -366.540358 305.672111
Individual: 2 1213.946842191739 -232.013239 165.307425 18.305874 24.348244 -131
Individual: 3 2402.554114973645 31.401978 456.946464 -391.007250 -78.400506 21
Individual: 4 -948.6356251882639 183.536746 -69.119518 357.399662 -94.280391
Individual: 5 -1493.9843828043547 -18.818025 416.659796 -245.030051 506.674691
Individual: 6 -4778.752929740184 61.308747 -302.325888 -298.553878 210.148920
Individual: 7 -890.0987592679384 -281.239080 416.409396 206.785709 243.001251
Individual: 8 539.7827849784019 -473.209562 -93.996307 -370.787316 84.847574 -1
Individual: 9 -2789.0383442776256 502.621561 -426.684202 -29.257077 -426.72041
Individual: 10 -2693.3024627962113 -234.746165 -251.214908 -479.312420 -279.931
Individual: 11 5942.486347571545 211.020061 -172.045134 182.584389 362.505491
Individual: 12 2532.513013965378 -405.998153 83.657851 165.734117 -161.880016
Individual: 13 -2947.728287802671 18.273809 151.519645 -380.971065 -258.45651
Individual: 14 3856.0385705239823 452.096887 -365.375677 -67.951509 350.7371
Individual: 15 -482.36728810255215 -204.388509 -164.093756 51.472008 -30.610421
Individual: 16 1700.8773032158055 373.704048 265.016853 184.705874 174.5041
```

Obr. 10 : Ukážka časti textového výpisu populácie

8 UKÁŽKA VPLYVU NASTAVENÍ NA PRIEBEH VÝPOČTOV

Pre zistenie funkčnosti a priebehu výpočtov som spriemerované hodnoty najlepších jedincov vynesol do grafov. Prvá časť dokazuje schopnosť programu nájsť globálne minimum daných funkcií, druhá znázorňuje vplyv nastavení na priebeh výpočtov

Všetky výpočty mali rovnakú počiatočnú populáciu, tým sa eliminoval možný vplyv rôzne vygenerovaných populácií na ich priebeh a výsledok.

Parametre :

Počet ostrovov : 10

Migrujúcich jedincov : 10

Kôl : 30

Topológia : Ring_1_2 (migrácia na najbližšie dva ďalšie ostrovy)

Hľadané minimum : -83796,58 (Schwefel)

Algoritmy :

8 * DERand1bin :

F = Rand(0, 2), náhodné pre každý ostrov

CR = Rand(0, 1) , náhodné pre každý ostrov

2 * SOMA :

pathLength = 3

step = 0.11

8.1 Priebeh vyhľadávania globálneho minima

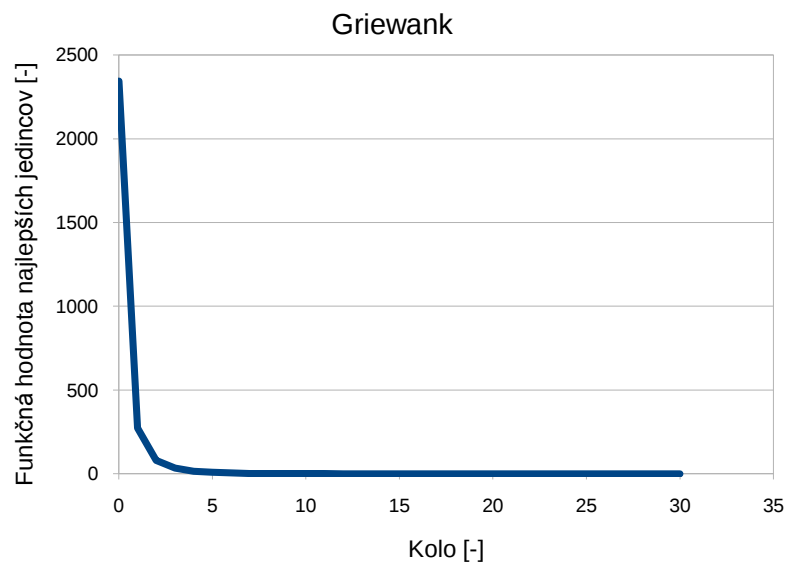
Všetky grafy sú normované, tj. s minimom v nule. Výsledky sú priemerom troch výpočtov.

Aj napriek veľkému počtu dimenzií (100) bolo minimum s veľkou presnosťou nájdené veľmi rýchlo.

Jedincov na ostrov : 50

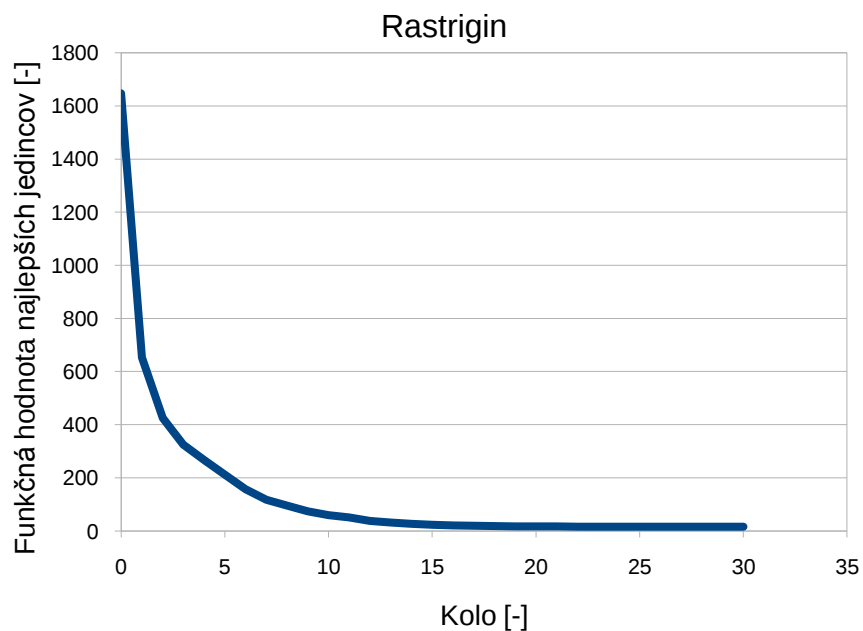
Počet dimenzií : 100

8.1.1 Griewank



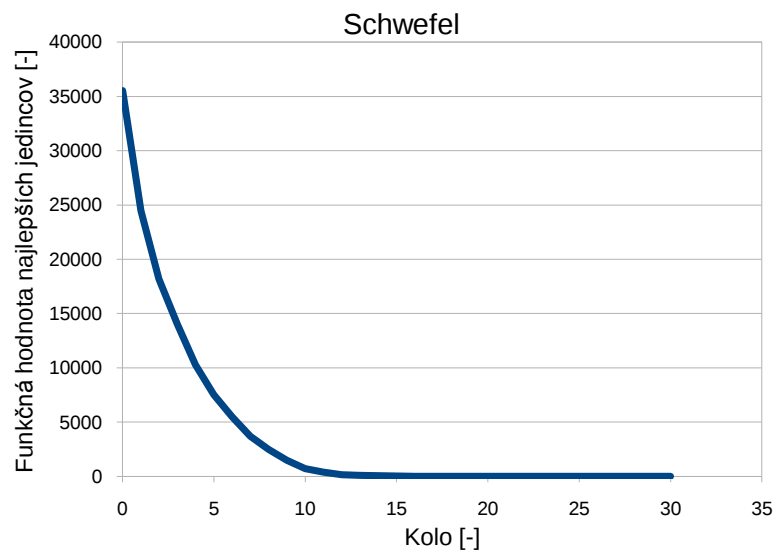
Graf 1 : Priebeh vyhľadávania globálneho minima na funkcii Griewank

8.1.2 Rastrigin



Graf 2 : Priebeh vyhľadávania globálneho minima na funkcii Rastrigin

8.1.3 Schwefel



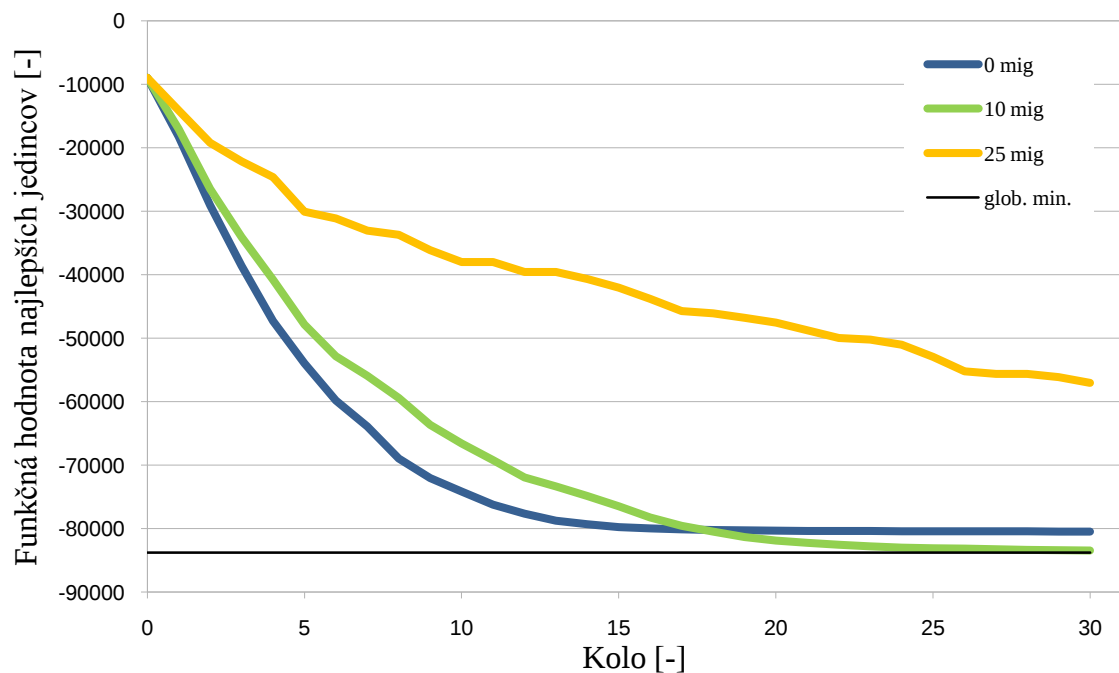
Graf 3 : Priebek vyhľadávania globálneho minima na funkcii Schwefel

8.2 Ukážka vplyvu nastavení na priebeh výpočtov

Jedincov na ostrov : 65

Počet dimenzií : 200

8.2.1 Vplyv počtu migrujúcich jedincov



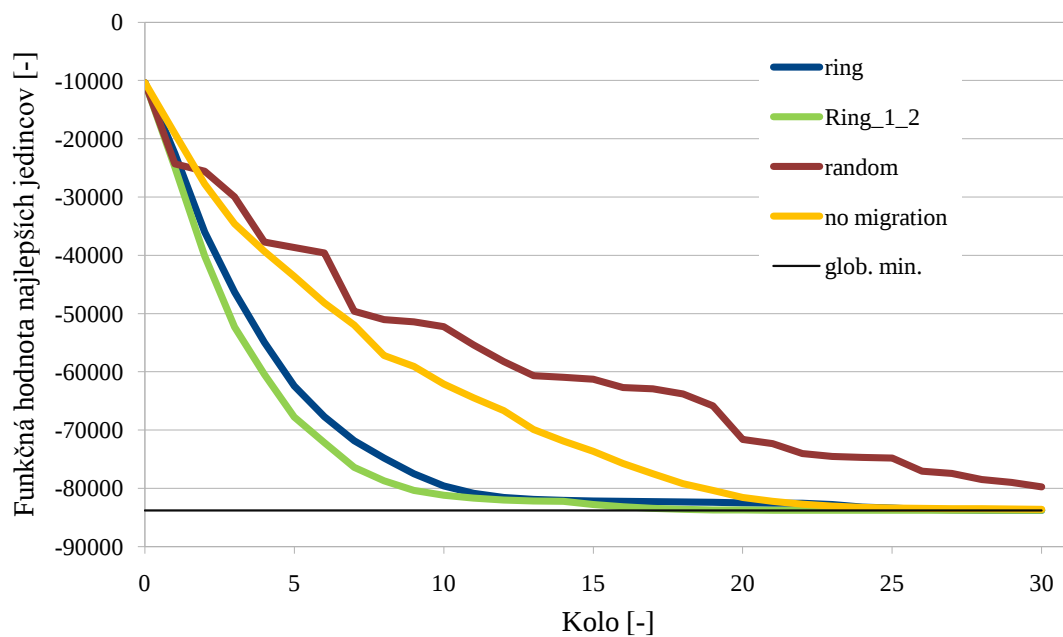
Graf 4 : Závislosť priebehu vyhľadávania extrémů na počte migrujúcich jedincov

Z grafu 1 jednoznačne vyplýva, že veľkosť migrujúcej populácie je veľmi dôležitá nastaviť na primeranú hodnotu (v tom to prípade 10 zo 65 jedincov). Bez migrácie sa síce hodnota ustálila veľmi rýchlo, ale aj najlepšia populácia zapadla do lokálneho minima z ktorého sa bez pomoci migrujúcich jedincov už nedokázala dostať.

Pri príliš veľkom počte migrujúcich jedincov sa stráca diverzita populácií a tým sa evolúcia výrazne spomalila a v prípade že by jedinci z jedného lokálneho minima premigrovali na všetky ostrovy a stiahli jej ostatok k sebe, všetky populácie by rýchlo zapadli do lokálneho minima a výsledok by bol neoptimálny.

V prípade správneho nastavenia migrácie je síce evolúcia o niečo pomalšia, o to sú ale jej výsledky lepšie. Pri hore uvedených nastaveniach sa výpočty dostali k hodnote globálneho minima len relatívne s malou odchýlkou.

8.2.2 Vplyv topológie ostrovov

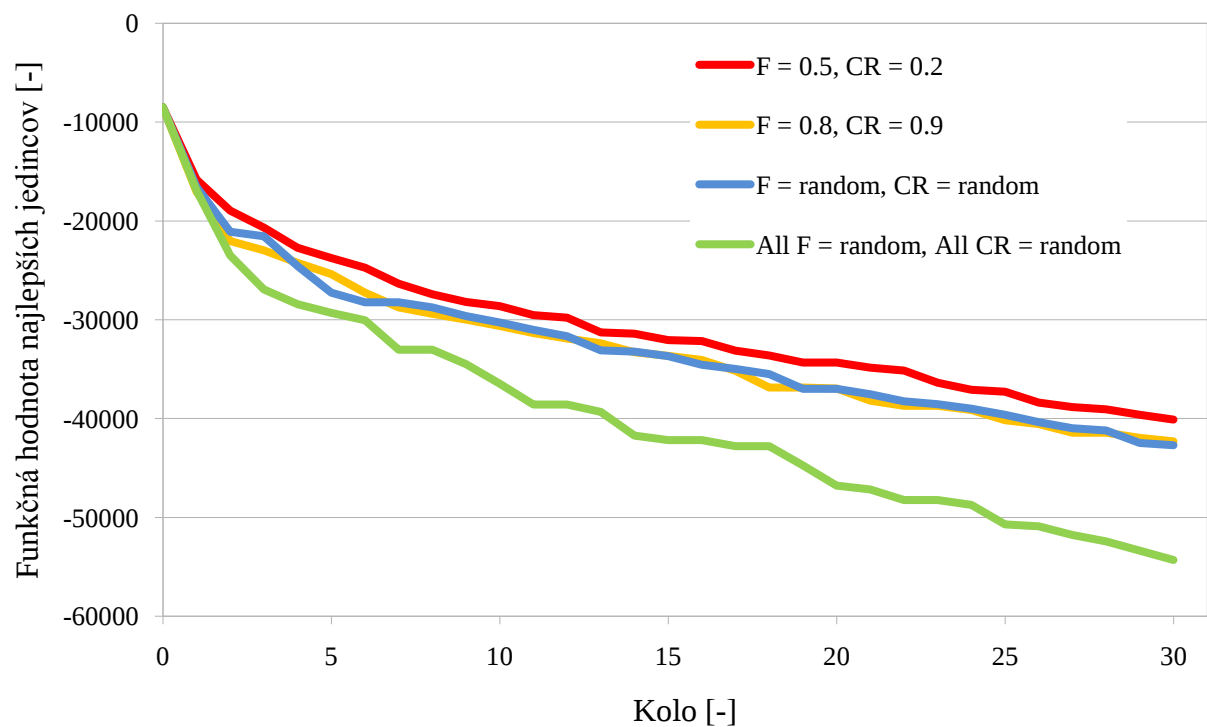


Graf 5 : Závislosť priebehu vyhľadávania extrémum na topológii ostrovov

Pri meniacich sa topológiách je ľahko vidieť výhody migrácie, evolúcia prebiehala výrazne rýchlejšie. Prekvapením bola veľmi pomalá a nepresná evolúcia pri migrácii na náhodný ostrov.

8.2.3 Vplyv parametrov

Pre testovanie vplyvu nastavených parametrov na výsledky bol vybraný algoritmus DE/Rand/1/bin.



Graf 6: Závislosť priebehu vyhľadávania extrémů na nastavení parametrov algoritmu

Ako najlepšia možnosť sa ukázal náhodný výber parametrov pre jednotlivé ostrovy. Rôznorodé parametre dovoľovali aspoň jednému z ostrovov efektívnejšie hľadať extrém.

ZÁVĚR

Cieľom tejto práce bolo vytvoriť framework pre distribuované evolučné algoritmy v jazyku Java.

Teoretická časť popisuje základy a priebeh evolučných algoritmov a rozdielne typy ich paralelizácie. Zahrnutý je aj popis konkrétnych algoritmov použitých vo vzorovej aplikácii.

Ďalej je popísaný programový model MapReduce a framework Hadoop použitý v tejto práci.

Prvá časť praktickej časti popisuje implementáciu ostrovných algoritmov do nadradeného frameworku a tým vytvorenie vyššej vrstvy abstrakcie pre užívateľa. V tejto sekcii je taktiež popísaný priebeh programu.

Nasleduje popis užívateľských tried, tj. tried ktoré bude programovať užívateľ podľa svojich potrieb. Pre každú triedu sú uvedené a popísané povinné funkcie a dedené premenné.

Ďalšou časťou je popis verejných funkcií samotného frameworku a ich parametrov. Pomocou týchto funkcií sa riadi chovanie frameworku. V rámci tejto sekcie sú popísané preddefinované topológie, vytváranie nových topológií a metódy emigrácie a imigrácie jedincov.

V nadväzujúcej časti je uvedená a popísaná vzorová aplikácia s krátkym návodom na jej spustenie a spôsoby výstupu informácií.

Posledná časť obsahuje testovanie aplikácie s rôznymi nastaveniami a vyhodnocuje ich vplyv na priebeh výpočtov.

Súčasťou práce sú zdrojové kódy frameworku a vzorovej aplikácie. Priložená je aj JavaDoc dokumentácia v ktorej sú popísané všetky verejné funkcie.

ZÁVĚR V ANGLIČTINĚ

The objective of this work was to create a framework for distributed algorithms in Java.

The theoretical part describes basics and processes of evolutionary algorithm and different types of their parallelization. Actual algorithms used in the exemplary application are also described.

Further, MapReduce programming paradigm and Hadoop framework used in this work are described.

The first part of the practical part describes implementation of island algorithms into superior framework and by this creating a higher level of abstraction for user. Flow of program's operations is also described in this section.

User classes are described in the next section. These are classes that the user will write as he needs. Each class has its mandatory functions and inherited variables described.

The next part documents public functions of the framework itself with their respective parameters. With these function user controls the behavior of the framework. This section also characterizes predefined topologies, creation of new topologies and methods of emigration and immigration of individuals.

The exemplary application is described in the following part, with a short manual for its launching and form of output information.

Last part contains tests of application with various settings and evaluates their effect on the computations.

Part of this work also consists of frameworks source codes with JavaDoc documentation describing all its public functions and an exemplary application.

SEZNAM POUŽITÉ LITERATURY

- [1] Z. Skolicki, K. De Jong. The influence of migration sizes and intervals on island models. In Proceedings of the 2005 Genetic and Evolutionary Computation Conference, pages 1295-1302. ACM Press, 2005.
- [2] ZELINKA, Ivan; OPLATKOVÁ, Zuzana; ŠENKEŘÍK, Roman. Aplikace umělé inteligence. Zlín : Ústav informatiky a umělé inteligence (Univerzita Tomáše Bati, Fakulta Aplikované Informatiky), 2010. 150 s. ISBN 978-80-7328-898-6.
- [3] CANTÚ-PAZ, Erick. A Survey of Parallel Genetic Algorithms. 1998. 30s.
- [4] PAVLECH, Michal. SOMA a paralelné výpočty pomocou modelu MapReduce. Trilobit [online]. 31.5.2010, 2010, [cit. 2011-01-10]. Dostupný z WWW: <<http://trilobit.fai.utb.cz/soma-a-paralelne-vypocty-pomocou-modelu-mapreduce>>.
- [5] Apache.org [online]. 2009-09-20 [cit. 2011-01-10]. ProjectDescription. Dostupné z WWW: <<http://wiki.apache.org/hadoop/ProjectDescription>>.
- [6] DEAN Jeffrey, GHEMAWAT Sanjay. MapReduce: Simplified Data Processing on Large Clusters. 2004, 13 s.
- [7] Hadoop [online]. 2011 [cit. 2011-04-21]. Wiki. Dostupné z WWW: <<http://wiki.apache.org/hadoop/>>.
- [8] VENNER, Jason. Pro Hadoop. Berkeley : Apress, 2009. 409 s.
- [9] Apache Hadoop [online]. 2008 [cit. 2011-05-29]. Hadoop Shell Commands. Dostupné z WWW: <http://hadoop.apache.org/common/docs/r0.17.2/hdfs_shell.html>.
- [10] ZEYLIGER, Philip. Cloudera.com [online]. 2009 [cit. 2011-05-30]. Hadoop Default Ports Quick Reference. Dostupné z WWW: <<http://www.cloudera.com/blog/2009/08/hadoop-default-ports-quick-reference/>>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HDFS Hadoop Distributed File System.

SOMA Samo Organizující sa Migračný Algoritmus

DE Diferenciálna evolúcia

M Prah mutácie

D Mutačná konštanta

CR Prah kríženia

SEZNAM OBRÁZKŮ

Obr. 1 : Obecný cyklus evolučného algoritmu (bez ukončenia).	12
Obr. 2 : Schéma coarse grained algoritmu. [3]	14
Obr. 3 : Schéma coarse grained algoritmu. [3]	14
Obr. 4 : Coarse grained - fine grained hybrid. [3].....	15
Obr. 5 : Princíp SOMA – a) pred migráciou, b) po migrácii na nových pozíciách.[2].....	17
Obr. 6 : Diagram vykonávania MapReduce [6]	21
Obr. 7 : Priebeh programu	24
Obr. 8 : Znázornenie kruhovej topológie	32
Obr. 9 : Znázornenie topológie v ktorej jedinci migrujú na dva následné ostrovy	33
Obr. 10 : Ukážka časti textového výpisu populácie.....	38

SEZNAM GRAFŮ

Graf 1 : Priebeh vyhľadávania globálneho minima na funkcii Griewank	40
Graf 2 : Priebeh vyhľadávania globálneho minima na funkcii Rastrigin	40
Graf 3 : Priebeh vyhľadávania globálneho minima na funkcii Schwefel	41
Graf 4 : Závislosť priebehu vyhľadávania extrému na počte migrujúcich jedincov.....	42
Graf 5 : Závislosť priebehu vyhľadávania extrému na topológii ostrovov	43
Graf 6: Závislosť priebehu vyhľadávania extrému na nastavení parametrov algoritmu.....	44