

Implementace vybraných evolučních algoritmů v embedded zařízeních

The Implementation of Selected Evolutionary Algorithms
Optimized for Embedded Systems

Martin Holčík

Bakalářská práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Martin HOLČÍK**
Osobní číslo: **A08616**
Studijní program: **B 3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**

Téma práce: **Implementace vybraných evolučních algoritmů v embedded zařízeních**

Zásady pro vypracování:

1. Vypracujte literární rešerši na dané téma.
2. Popište zvolené embedded zařízení.
3. Navrhněte vlastní implementaci vybraných evolučních algoritmů.
4. Analyzujte výkon navržených implementací.
5. Demostrujte výsledky na ukázkové aplikaci.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. ZELINKA, Ivan, Zuzana OPLATKOVÁ, Miloš ŠEDA, Pavel OŠMERA a František VČELARĚ. Evoluční výpočetní techniky: principy a aplikace. 1. české vyd. Praha: BEN, 2009, 534 s. ISBN 978-052-1880-688.
2. PRESS, William H., Saul A. TEUKOLSKY, William T. VETTERLING a Brian P. FLANNERY. Numerical recipes: the art of scientific computing. 3rd ed. Cambridge: Cambridge University Press, 2007, 1235 s. ISBN 978-052-1880-688.
3. KVASNÍČKA, V.; POSPÍCHAL, J.; TIŇO, P. Evoluční algoritmy. Bratislava : STU, 2000. 215 s. ISBN 80-227-1377-5.
4. KRAMPL, Jakub. Implementace vybraných evolučních algoritmů v prostředí .NET. Zlín, 2011. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně.
5. KRÁL, Tomáš. Knihovna evolučních optimalizačních algoritmů v prostředí Java. Zlín, 2011. Diplomová práce. Univerzita Tomáše Bati ve Zlíně.

Vedoucí bakalářské práce:

Ing. Erik Král

Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce:

24. února 2012

Termín odevzdání bakalářské práce:

8. června 2012

Ve Zlíně dne 24. února 2012



prof. Ing. Vladimír Vašek, CSc.

děkan



prof. Ing. Vladimír Vašek, CSc.

ředitel ústavu

ABSTRAKT

Cílem této bakalářské práce je implementovat vybrané evoluční algoritmy – Particle Swarm Optimization (PSO), Self Organized Migration Algorithm (SOMA) a Differential Evolution (DE) na embedded zařízení CmuCam3 a porovnat jejich výkon.

Klíčová slova: Evoluční Algoritmus, Embedded, CmuCam3, Particle Swarm Optimization, Self Organized Migration Algorithm, Differential Evolution

ABSTRACT

This Bachelor thesis is focused on implementation of the selected evolutionary algorithms – Particle Swarm Optimization (PSO), Self Organized Migration Algorithm (SOMA) and Differential Evolution (DE) on embedded device CmuCam3 and compare performance.

Keywords: Evolutionary Algorithm, Embedded, CmuCam3, Particle Swarm Optimization, Self Organized Migration Algorithm, Differential Evolution

Tímto děkuji vedoucímu mé bakalářské práce panu Ing. et Ing. Eriku Královi za odborné vedení a za cenné rady při vypracovávání této práce.

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
1 TEORETICKÁ ČÁST	10
1 OPTIMALIZAČNÍ ALGORIMY	11
1.1 DĚLENÍ OPTIMALIZAČNÍCH ALGORITMŮ.....	11
1.1.1 Enumerativní	11
1.1.2 Deterministické	11
1.1.3 Stochastické.....	12
1.1.4 Smíšené	12
2 EVOLUČNÍ ALGORITMY	13
3 ROJENÍ ČÁSTIC	14
3.1 PRINCIP	14
3.1.1 Postup prohledávání prostoru tradičního algoritmu PSO.....	15
3.2 PARAMETRY	16
3.3 DALŠÍ VARIANTY PSO.....	17
4 SAMOORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS	19
4.1 PRINCIP	19
4.1.1 Postup při prohledávání prostoru	20
4.2 STRATEGIE	21
4.3 PARAMETRY	22
5 DIFERENCIÁLNÍ EVOLUCE	23
5.1 PRINCIP	23
5.1.1 Postup vyhledávání	23
5.2 PARAMETRY	24
5.3 DALŠÍ VARIANTY DE.....	25
6 UKONČOVACÍ KRITÉRIA	27
7 TESTOVACÍ FUNKCE	29
7.1 PRVNÍ DE JONGOVA FUNKCE.....	29
7.2 DRUHÁ DO JONGOVA FUNKCE	30
7.3 RASTRIGINOVA FUNKCE	31
7.4 ACKLEYHO FUNKCE.....	32
7.5 EASOMOVA FUNKCE	34
7.6 GOLDSTEINOVA-PRICEOVA FUNKCE	35
8 CMUCAM3	37
8.1 POPIS.....	37
8.1.1 Technické parametry	38

II	PRAKTICKÁ ČÁST	39
9	POPIS PROGRAMU	40
9.1	IMPLEMENTACE EVOLUČNÍCH ALGORITMŮ	41
9.1.1	Implementace algoritmu PSO	41
9.1.2	Implementace algoritmu SOMA	46
9.1.3	Implementace algoritmu DE	48
10	POROVNÁNÍ VÝSLEDKŮ	50
	ZÁVĚR	51
	ZÁVĚR V ANGLIČTINĚ.....	52
	SEZNAM POUŽITÉ LITERATURY.....	53
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	55
	SEZNAM OBRÁZKŮ	56
	SEZNAM TABULEK.....	57
	SEZNAM PŘÍLOH.....	58

ÚVOD

Tato bakalářská práce se zabývá Evolučními algoritmy, konkrétně Rojem částic, Samo-organizujícím se Migračním algoritmem a diferenciální evolucí. Dále pak jsou tyto algoritmy implementovány v jazyce C podle normy ISO/IEC 9899:1999 [1]. Jejich výkon je otestován na embedded zařízení CMUCam3.

Evoluční algoritmy jsou inspirovány Darwinovou Teorií evoluce, ze které přebírají některé rysy, jako křížení, mutace, dědičnost, vývoj v generacích, přirozený výběr atd. Díky těmto vlastnostem jsou výkonným nástrojem pro řešení složitých optimalizačních úloh.

V teoretické části jsou popsány evoluční algoritmy obecně, jejich rozdělení, vlastnosti. Dále jsou popsány vybrané algoritmy – Roj částic, Samo-organizujícím se Migrační algoritmus a diferenciální evoluce. Popsán je princip těchto algoritmů a jejich vlastnosti.

Praktická část obsahuje zdrojové kódy daných algoritmů. Je zde také uveden postup implementace a testování těchto algoritmů pro vybrané účelové funkce na zařízení CmuCam3 a vyhodnocení výkonu těchto algoritmů.

I. TEORETICKÁ ČÁST

1 OPTIMALIZAČNÍ ALGORIMY

Cílem optimalizačních algoritmů je nalezení globálního extrému dané účelové funkce omezené oborem hodnot. Algoritmus se snaží vyhledat takovou kombinaci argumentů, pro které bude mít hodnota řešené funkce minimální hodnotu. Účelová funkce se značí obecně $F(x)$, *fitness* nebo $F_{\text{cost}}(x)$ kde „cost“ je anglický výraz pro „cena“. Dříve se také říkalo účelové funkci „cenová funkce“. Prostor, na kterém se účelová funkce nachází, se nazývá hyperplocha nebo také prostor možných řešení. Dimenze D je dána počtem argumentů řešené účelové funkce. [2][3][4]

1.1 Dělení optimalizačních algoritmů

Jedno z možných rozdělení optimalizačních algoritmů je podle způsobu hledání kombinace optimálních argumentů účelové funkce: [2][3]

1.1.1 Enumerativní

Algoritmus vyčíslí všechny možné kombinace argumentů dané funkce. Tento postup je vhodný pouze pro řešení funkcí, jejíž argumenty nabývají diskrétních hodnot. Pro řešení úloh, kde argumenty funkce nabývají hodnot v oboru reálných čísel je tento algoritmus nepoužitelný, neboť doba výpočtu by se rovnala nekonečnu. [2][3]

1.1.2 Deterministické

Tyto algoritmy jsou založeny pouze na rigorózních metodách klasické matematiky. Algoritmus reaguje předvídatelně, reaguje tedy vždy stejně na stejné výchozí podmínky. Dosažení skutečně efektivních výsledků je za předpokladu že:

- Prohledávaný prostor je spojitý
- Problém je konvexní
- Účelová funkce je unimodální
- Problém je definován v analytickém tvaru [2][3]

1.1.3 Stochastické

Stochastické algoritmy využívají náhodného hledání. Algoritmus prakticky hledá náhodně hodnoty argumentů účelové funkce. Výsledkem je takové řešení, pro které byly nalezeny nejlepší hodnoty argumentů během celého hledání. Tyto algoritmy jsou pomalé a nepřesné. Slouží spíše jako orientační řešení. [2][3]

1.1.4 Smíšené

Smíšené algoritmy kombinují vlastnosti stochastických a deterministických algoritmů. Díky tomu mohou dosahovat velmi dobrých výsledků. Vlastnosti smíšených algoritmů:

- Robustnost - Velmi často naleznou globální extrém nezávisle na počátečních podmínkách
- Efektivita a výkon - mají předpoklad k nalezení optimálního řešení i během relativně malého počtu ohodnocení účelové funkce
- Bez nutnosti informací o řešené funkci

Mezi smíšené algoritmy patří také skupina evolučních algoritmů. [2][3]

2 EVOLUČNÍ ALGORITMY

Charles Darwin v 18. Století popsal Evoluční teorii o přirozeném výběru, kde je populace jedinců šlechtěna v generacích a právo na přežití mají pouze nejvhodnější jedinci.

Evoluční algoritmy (Evolutionary Algorithm, EA) jsou účinné optimalizační algoritmy, jejichž princip vychází z výše zmíněné teorie. Stejně jako u evoluční teorie, tak i zde probíhá šlechtění jedinců. Tito jedinci prohledávají prostor za účelem nalezení globálního minima. Děje se tak tomu v generacích, kde každá generace je složena z jedinců lepších nebo alespoň stejně kvalitních ve srovnání s generací předchozí. Probíhá zde selekce jedinců, kde v populaci zůstanou pouze nejlepší jedinci. Vhodnost jedince je označována anglosaským výrazem fitness. Evoluční Algoritmy typicky využívají evoluční operátory jako:

- Selekcce – jen nejlepší jedinci v populaci mohou přežít
- mutace – vlastnosti jedince jsou náhodně modifikovány
- křížení – je vytvořen nový jedinec rekombinací dvou nebo více jedinců z populace

Mezi výhody evolučních algoritmů patří především použitelnost na složité optimalizační problémy.

Nevýhodou evolučních algoritmů je fakt, že není zaručeno, že nalezené řešení odpovídá skutečně globálnímu extrému. [5][6]

3 ROJENÍ ČÁSTIC

Rojení Částic (Particle Swarm Optimization, PSO) je evoluční optimalizační metaheuristická stochastická výpočetní technika vyvinuta roku 1995 Dr. Kennedym a Dr. Eberhartem. Tato metoda je inspirována sociálním chováním ptačího, nebo rybího hejna při hledání potravy.

System je založen na populaci, jejíž jedinci jsou náhodně generováni v daném prostoru. Každý z těchto jedinců se pohybuje v prostoru a hledá optimální řešení. Podobně jako u genetických algoritmů zde probíhá hledání optimálního řešení v generacích, ale bez evolučních operátorů jako křížení, mutace a bez vytváření nových potomků. Tato metoda nezaručuje vyhledání optima pro každou funkci. Úspěšnost silně závisí na testované funkci a na vhodně zvolených parametrech. [5][7][8]

3.1 Princip

Algoritmus PSO lze přirovnat k ptačímu hejnu, jehož jedinci hledají nejvyšší bod v prohledávané oblasti. Informace o poloze nejvyššího místa není pro skupinu známa, ale každý jedinec po každé iteraci ví, který jedinec našel nejvyšší pozici. Zbytek hejna tedy následuje jedince s nejlepším výsledkem.

Každý jedinec uchovává informaci o své poloze v prohledávaném prostoru, rychlosti a aktuální nejlepší pozici značené pBest. Nejlepší pozice jedince s nejlepším výsledkem je uchovávána v proměnné gBest. Tato proměnná je známa pro všechny jedince populace. [5][7][8]

3.1.1 Postup prohledávání prostoru tradičního algoritmu PSO

1. Náhodná generace jedinců v prostoru
2. Výpočet rychlosti částice

$$v_d(t+1) = v_d(t) + c_1 \cdot rand_1 \cdot (pBest_{i,j} - x_{i,d}(t)) + c_2 \cdot rand_2 \cdot (gBest_d - x_{i,d}(t)) \quad (1)$$

kde

- i - $i = 1, 2, \dots, N$, kde N je počet částic,
- d - $d = 1, 2, \dots, D$, kde D je dimenze prohledávaného prostoru,
- t - $t = 1, 2, \dots, MAX$, kde MAX je počet iterací,
- $v_d(t+1)$ - je rychlost jedince v následující iteraci,
- $v_d(t)$ - je rychlost jedince v této iteraci,
- $pBest_{i,d}$ - je nejlepší dosavadní pozice jedince,
- $gBest_d$ - je nejlepší nalezená pozice v populaci,
- $rand$ - je náhodné číslo z intervalu $[0,1]$,
- c_1, c_2 - učící se faktory.

3. Výpočet pozice (2) za použití rovnice (1)

$$x_{i,d}(t+1) = x_{i,d}(t) + v_d(t+1) \quad (2)$$

kde

- $x_{i,d}(t+1)$ - je pozice jedince v následující iteraci,
- $x_{i,d}(t)$ - je pozice jedince v této iteraci.

4. Porovnání, jestli je hodnota účelové funkce pro nově spočtenou pozici jedince lepší než jeho doposud nejlepší řešení. Pokud je nová pozice jedince lepší, uloží souřadnice tohoto jedince do proměnné $pBest$.

5. Porovnání, pokud je hodnota účelové funkce pro pBest lepší než gBest uloží hodnotu pBest do proměnné gBest.
6. Proces pokračuje bodem 2 se stejným způsobem v dalším iteračním kole pro všechny částice až do uplynutí poslední nastavené iterace nebo nastaveným ukončovacím parametrem. [5]

3.2 Parametry

Dimenze

Je dána počtem argumentů řešené účelové funkce. Velikost dimenze se rovná právě počtu argumentů řešené účelové funkce. Každý jedinec uchovává souřadnice pro všechny dimenze. [5]

Rozsah

Jedná se o velikost prohledávaného prostoru. Pro každou dimenzi může existovat i více intervalů. Prohledávaná oblast nemusí být spojitá. [5]

Počet částic

Udává počet jedinců, tvořící populaci. S větším počtem částic roste hustota prohledávaného prostoru, ale také výpočetní náročnost. [5]

Vmax

Udává maximální možnou rychlost částice. Tento parametr slouží k tomu, aby se částice nevzdalovaly z prohledávané oblasti. Nastavení parametru na příliš vysokou hodnotu způsobí zmíněné vzdalování jedinců z prohledávané oblasti. Příliš nízká hodnota naopak způsobí, že každá částice bude prohledávat pouze malý prostor ve svém okolí a nemusí tak nikdy najít optimální řešení. [5]

Učící se faktory c1 ,c2

Tyto parametry ovlivňují směr pohybu částice. Parametr c1 způsobuje pohyb částice směrem ke svému nejlepšímu výsledku. Parametr c2 způsobuje pohyb částice ke nejlepšímu výsledku populace. Oba parametry jsou násobeny náhodným číslem z intervalu [0,1]. [5]

Setrvačnost

Tento parametr byl do algoritmu přidán později - roku 1998. Tímto parametrem se násobí rovnice (3) a upravuje se tak rychlost jedince. Pro $w > 1$ rychlost narůstá, pro $w < 1$ rychlost klesá. Ve tradiční verzi PSO je setrvačnost rovna konstantě. Lze použít i setrvačnost s dynamicky měnící se hodnotou. Díky tomu se částice na začátku pohybuje rychleji a díky tomu se rychleji přiblíží hledanému výsledku a s každou novou generací hodnota tohoto parametru klesá a oblast je tak prohledávána podrobněji. [5][9]

$$v_d(t+1) = w \cdot v_d(t) + c_1 \cdot rand \cdot (pBest_{i,j} - x_{i,d}(t)) + c_2 \cdot rand \cdot (pBest_j - x_{i,d}(t)) \quad (3)$$

$$w = w_{start} - \frac{(w_{start} - w_{end}) \cdot iteraçe}{migrace} \quad (4)$$

kde

w_{start}, w_{end} - jsou konstanty volené uživatelem,

$iteraçe$ - je aktuální iterace,

$migrace$ - je počet migračních kol celkem.

3.3 Další varianty PSO

SPSO

Standardní PSO se liší od tradičního PSO ve výpočtu rychlosti. V této verzi je navíc parametr x (5), takzvaný omezující faktor. Rychlost je zde počítána přes rovnici (7). [9]

$$x = \frac{2}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4 \cdot \varphi} \right|} \quad (5)$$

$$\varphi = c_1 + c_2 \quad (6)$$

$$v_d(t+1) = x \cdot (v_d(t) + c_1 \cdot rand_1 \cdot (pBest_{i,j} - x_{i,d}(t)) + c_2 \cdot rand_2 \cdot (pBest_j - x_{i,d}(t))) \quad (7)$$

Speciation PSO

V této verzi algoritmu jsou z jedinců v každém kole tvořeny skupiny. Uvnitř skupin platí stejná pravidla jako u tradičního PSO. [5]

Niching PSO

Kolem částice s nejlepším nalezeným řešením je vytvořena skupina z několika jedinců. Tato skupina prohledává nalezenou oblast podrobněji. Zbylé částice pokračují v prohledávání oblasti. Mohou se tvořit další podskupiny. [5]

INPSO

Independent Neighbourhoods Particle Swarm Optimization. Jak již název napovídá, tato verze pracuje na principu prohledávání oblasti v samostatných, na sobě nezávislých skupinách. [5]

DN-PSO

Dynamic Neighbourhood PSO. Částice jsou zde rozděleny do nezávislých skupin jako u INPSO. Liší se v tom, že pokud je částice blíže jiné skupině než své aktuální, je přijata touto bližší skupinou. [5]

4 SAMOORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS

SamoOrganizující se Migrační Algoritmus (Self Organized Migration Algorithm, SOMA) je evoluční optimalizační stochastický algoritmus vytvořen roku 1999 prof. Ivanem Zelinkou. Algoritmus je inspirován přírodou, konkrétně stádem inteligencích jedinců, kteří kooperují při hledání potravy.

Algoritmus SOMA simuluje chování jedinců ve stádě s vedoucím jedincem. Každý z těchto jedinců se snaží najít nejlepší zdroj potravy. Hledání řešení probíhá v evolučních cyklech, zde nazývanými migrační kolo. Probíhá zde soutěživě-komparativní strategie, kde každý jedinec hledá nejlepší řešení – lepší než ostatní (soutěžení) a zároveň mají jedinci informace o stavu ostatních jedinců a na základě toho upravují svoje chování (kooperace). Odtud také plyne název „samoorganizující se“. Při srovnání s genetickými algoritmy zde neprobíhá křížení jedinců ani tvorba nových potomků. Tento Algoritmus lze také zařadit mezi algoritmy memetické nebo hejnové. [5]

4.1 Princip

Tento algoritmus pracuje s populací, jejíž jedinci jsou náhodně generováni pro zvolený prostor, za účelem nalezení optimálního řešení. Každý jedinec se pohybuje po prohledávané oblasti a nese informace o své poloze v prostoru. Ten z jedinců, který v daném migračním kole našel nejlepší řešení je brán jako vůdce, a ostatní jedinci migrují k tomuto jedinci. [5]

4.1.1 Postup při prohledávání prostoru

1. Vytvoření populace generováním náhodných jedinců ve zvoleném prostoru
2. Výběr nejlepšího jedince ze skupiny – Leader
3. Výpočet nové polohy částice

$$x_{i,j}^{ML+1} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML+1}) \cdot t \cdot PRTVector_j \quad (8)$$

kde

- i - $i = 1, 2, \dots, N$, kde N je počet částic,
- j - $j = 1, 2, \dots, J$, kde J je dimenze prohledávaného prostoru,
- ML - $ML = 1, 2, \dots, MAX$, kde MAX je počet migračních kol,
- $x_{i,j}^{ML+1}$ - je pozice jedince v následujícím migračním kole,
- $x_{i,j,start}^{ML}$ - je pozice jedince v aktuálním migračním kole,
- $x_{L,j}^{ML+1}$ - je pozice leadera,
- t - nabývá hodnot v rozsahu $[0, PathLength]$,
- $PRTVector_j$ - perturbační vektor.

4. Porovnání účelových funkcí nových jedinců s původními. Zůstávají pouze ti s lepšími výsledky.
5. Skok na bod 2. Proces končí buď zvoleným ukončovacím kritériem nebo nastavením maximálního počtu iterací. [5]

4.2 Strategie

AllToOne – Všichni k jednomu

Tato strategie již detailněji popsána v předešlém bodě. Kde všichni jedinci z populace (kromě leadera) migrují směrem k leaderovi populace. [5]

AllToOne – Všichni ke všem

Jak již vypovídá název, tak v této strategii každý jedinec migruje ke všem ostatním jedincům ve stádě. Migruje vždy ze stejného místa, do nové pozice se přesune až po všech migracích ke všem ostatním. Tato variace je výpočetně mnohem náročnější než AllToOne, ale je zde větší pravděpodobnost, že bude nalezeno globální minimum, neboť je prohledáván větší prostor. [5]

AllToAllAdaptive – Adaptivně všichni ke všem

Strategie vychází s AllToAll strategie. Jediný rozdíl je v tom, že jedinec migruje k dalšímu jedinci z aktuální nejlepší nalezené pozice, tzn. jakmile najde lepší pozici přesune se na ni okamžitě, ne až po všech migracích jako u AllToAll. [5]

AllToOneRand – Všichni k jednomu jednotlivě

V této strategii migruje každý jedinec vždy jen k jednomu, náhodně zvolenému jedinci. [5]

4.3 Parametry

Dimenze

Dimenze se rovná počtu argumentů řešené účelové funkce. [2]

PopSize

Počet jedinců, kteří prohledávat prostor a hledají optimální řešení. [2]

PathLenght

Je vzdálenost nové pozice jedince od vedoucího jedince. Při PathLenght = 0 jedinec zůstane na původní pozici, Při PathLenght = 1 se jedinec přemístí na pozici nejlepšího jedince při PathLenght > 1 zastaví jedinec za leaderem. Pokud je PathLenght < 1 jedinec zastaví před vedoucím jedincem a stádo tak bude degenerovat a sníží se schopnost nalezení globálního extrému. [2]

Step

Parametr určuje, s jakou hrubostí bude prostor prohledáván. Parametr musí být menší než hodnota PathLenght a také by neměl být jeho celočíselným násobkem. Při nastavení vyšších hodnot Step je prostor prohledáván rychleji, naopak při nižších nastavených hodnotách je prostor prohledáván pomaleji ale důkladněji. [2]

PRT

Tento parametr představuje perturbaci, která má za účel náhodnou změnu jedince (prakticky jde o mutaci). Parametr nabývá hodnot v intervalu [0,1]. Tento parametr určuje, zdali se bude jedinec pohybovat směrem k vedoucímu jedince nebo zůstane na místě. Tímto parametrem je tvořen perturbační vektor – PRTVector. Hodnoty vektoru nabývají hodnot 0 nebo 1. Pro každý parametr PRTVectoru je generování náhodné číslo z intervalu [0,1]. Pokud je vygenerované číslo menší než PRT tak je do PRTVectoru Uložena 1 pokud je generované číslo větší než PRT tak se uloží 0. [2]

5 DIFERENCIÁLNÍ EVOLUCE

Diferenciální Evoluce (Differential Evolution, DE) je evoluční optimalizační metoda, vyvinuta Kenem Pricem a Rainerem Stormem v roce 1995. Základy této metody jsou položeny v genetickém žihání, publikované Pricem o rok dříve. DE se podobá genetickým algoritmům, se kterými má společné například hledání v generacích, vytváření potomků (v tomto případě ze čtyř jedinců) a mutace. [5]

5.1 Princip

Tento algoritmus pracuje s populací jedinců, kteří jsou náhodně vygenerováni ve zvoleném prostoru. Tito jedinci jsou šlechtěni v generacích. V každé nové generaci je pro každého jedince vytvořen potomek ze tří náhodně zvolených rodičů, takzvaný šumový vektor. Tento šumový vektor soutěží o místo v populaci. [5]

5.1.1 Postup vyhledávání

1. Náhodné generování jedinců ve zvoleném prostoru
2. Postupně pro každého jedince v generaci se vyberou tři náhodně zvolení jedinci. Dva ze tří zvolených jedinců se od sebe odečtou a vznikne tak **diferenční vektor**. Na něm proběhne mutace – vynásobením mutační konstantou F a vznikne tak **váhovaný diferenční vektor**. Ten je přičten ke zbylému třetímu náhodnému jedinci a vznikne šumový vektor (9). [5]

$$v_j = x_{r3,j}^G + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (9)$$

kde

- j - $j = 1, 2, \dots, J$, kde J je počet argumentů řešené účelové funkce,
- x - je pozice částice,
- x_{r1}, x_{r2}, x_{r3} - jsou náhodně zvolení jedinci,
- F - je mutační konstanta.

3. Dále je tvořen **zkušební vektor**. Ten je vytvořen za pomoci váhového diferenciálního vektoru a aktuálního jedince, kde se postupně pro každou souřadnici generuje náhodné číslo v intervalu $[0,1]$ a pokud je hodnota větší než parametr CR tak se do tohoto zkušební vektoru uloží hodnota z aktuálního jedince, při generování hodnoty větší než CR se ukládá hodnota z šumového vektoru.
6. Porovnání hodnot účelové funkce zkušební vektoru s původním jedincem. Pokud má zkušební vektor hodnotu účelové funkce lepší, nahradí aktuálního jedince. V opačném případě zůstává aktuální jedinec.
7. Proces se opakuje od bodu 2 až do vyčerpání možných generací nebo v závislosti na ukončovacích parametrech. [5]

5.2 Parametry

V následujícím textu budou vysvětleny parametry algoritmu. [5]

D

Parametr D Představuje dimenzi. Dimenze je rovna počtu argumentů řešené funkce.

NP

Parametr udávající počet jedinců v populaci. Minimální možná hodnota je 4.

CR

Tento parametr představuje práh křížení. Nabývá hodnot v intervalu $[0,1]$.

F

Mutační konstanta. Pokud je $F = 0$, neprobíhá žádná mutace. Čím více je F různé od 1, tím více je částice mutována.

Generation

Jedná se o počet generací, ve kterých bude probíhat hledání globálního minima. Hodnota musí být větší než 0.

5.3 Další varianty DE

Různé varianty Diferenciální Evoluce se od sebe liší především ve výpočtu šumového vektoru. Výše uvedený postup popisuje variantu DE/rand/1exp. Kde DE je zkratka pro Diferenciální Evoluci, rand je metoda výběru rodiče (rand = náhodně, best = nejlepší), další parametr uvádí počet vektorů, jež perturbují základní vektor. Poslední parametr představuje typ použitého křížení (exp = exponenciální, bin = binomiální). Jak je patrné z rovnic (10) až (19) binomiální a exponenciální varianty používají stejné rovnice pro výpočet šumového vektoru. [5]

DE/best/1/exp

$$v = x_{best,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (10)$$

DE/rand/1/exp

$$v = x_{r1,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (11)$$

DE/rand-to-best/1/exp

$$v_j = x_{i,j}^G + \lambda \cdot (x_{best,j}^G - x_{i,j}^G) + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (12)$$

DE/best/2/exp

$$v = x_{best,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (13)$$

DE/rand/2/exp

$$v = x_{r5,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (14)$$

DE/best/1/bin

$$v = x_{best,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (15)$$

DE/rand/1/bin

$$v = x_{r1,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (16)$$

DE/rand-to-best/1/bin

$$v_j = x_{i,j}^G + \lambda \cdot (x_{best,j}^G - x_{i,j}^G) + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (17)$$

DE/best/2/bin

$$v = x_{best,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (18)$$

DE/rand/2/bin

$$v = x_{r5,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (19)$$

6 UKONČOVACÍ KRITÉRIA

Ukončovací kritéria slouží k zastavení běhu evolučních algoritmů. Pro zrychlení chodu evolučních algoritmů je rozumné využít některých z níže uvedených ukončovacích kritérií, které zastaví algoritmus na základě jejich naplnění. Bez použití ukončovacích kritérií je algoritmus ukončen až po uplynutí maximálního počtu iterací. [10]

ImpBest

Je zde sledována nejlepší hodnota účelové funkce. Pokud je změna hodnoty účelové funkce menší než prahové t pro generace g , optimalizace je ukončena.

ImpAv

Princip podobný ImpBest s tím rozdílem, že je zde sledována průměrná hodnota účelové funkce pro celou populaci.

MovObj

Pokud je pohyb částice, s ohledem na hodnotu funkce, menší než prahové t pro generace g , je algoritmus ukončen.

MovPar

Pokud je pohyb částice, s ohledem na polohu částice, menší než prahové t pro generace g , je algoritmus ukončen.

StdDev

Pokud je směrodatná odchylka účelových funkcí pro všechny jedince populace menší než prahová hodnota, algoritmus končí.

MaxDist

V tomto případě je sledována maximální vzdálenost všech jedinců od jedince s nejlepším výsledkem. Pokud je tato maximální vzdálenost menší než zvolená hodnota, je běh algoritmu ukončen.

MaxDistQuick

Toto kritérium vychází z principu MaxDist s tím rozdílem, že zde nejsou sledovány vzdálenosti všech jedinců od jedince s nejlepším výsledkem, ale pouze určité procento jedinců z populace.

Diff

Pokud je rozdíl mezi nejlepší a nejhorší hodnotou účelové funkce menší než zvolená hodnota d , je algoritmus ukončen.

ComCrit

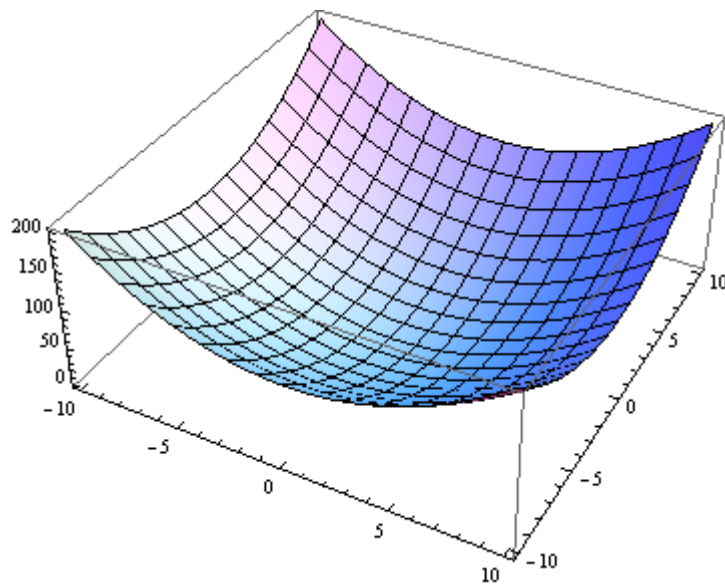
Toto kritérium je kombinací ImpAv a MaxDist. Pokud je splněna podmínka ImpAv je zkontrolováno kritérium MaxDist. Pokud jsou splněny obě kritéria, je běh algoritmu ukončen.

7 TESTOVACÍ FUNKCE

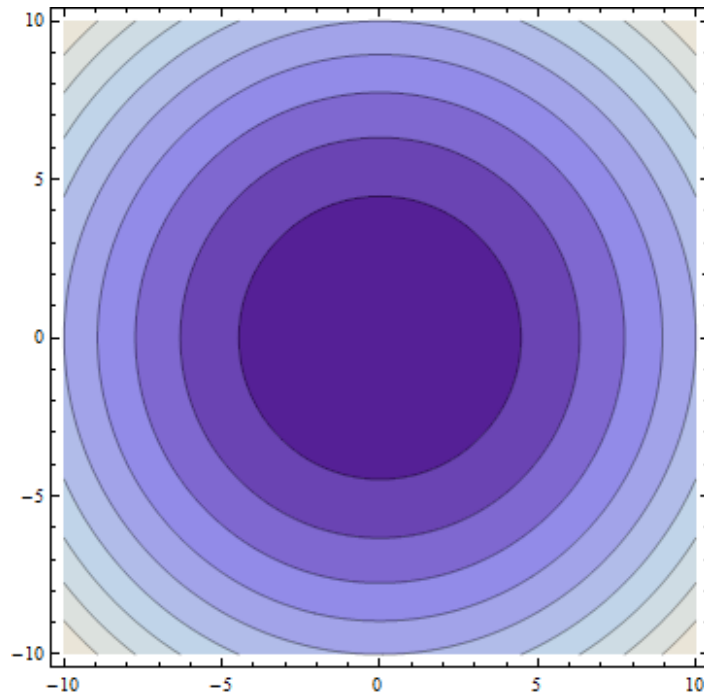
Pro otestování výše zmíněných algoritmů byly vybrány následující funkce [11]. Hodnoty globálních extrémů získaných evolučními algoritmy budou porovnány se skutečnými hodnotami globálních extrémů.

7.1 První de Jongova funkce

$$f(x_1, x_2) = x_1^2 + x_2^2 \quad (20)$$



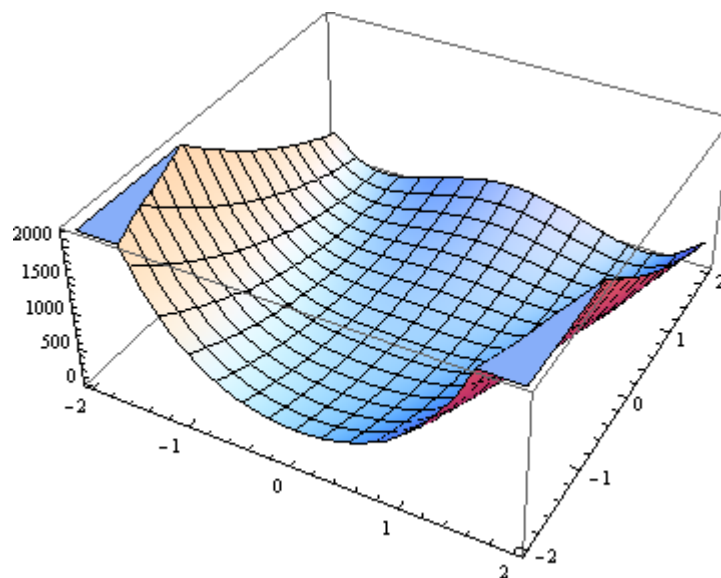
Obrázek 1. První de Jongova funkce ve 3D



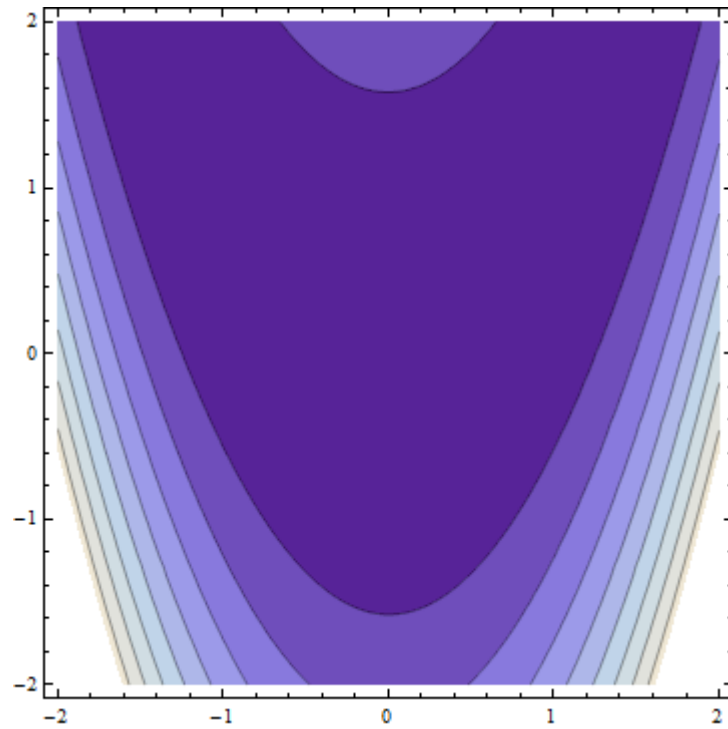
Obrázek 2. první de Jongova funkce

7.2 Druhá do Jongova funkce

$$f(x_1, x_2) = 100 \cdot (x_1^2 - x_2)^2 + (x_1 - x_1)^2 \quad (21)$$



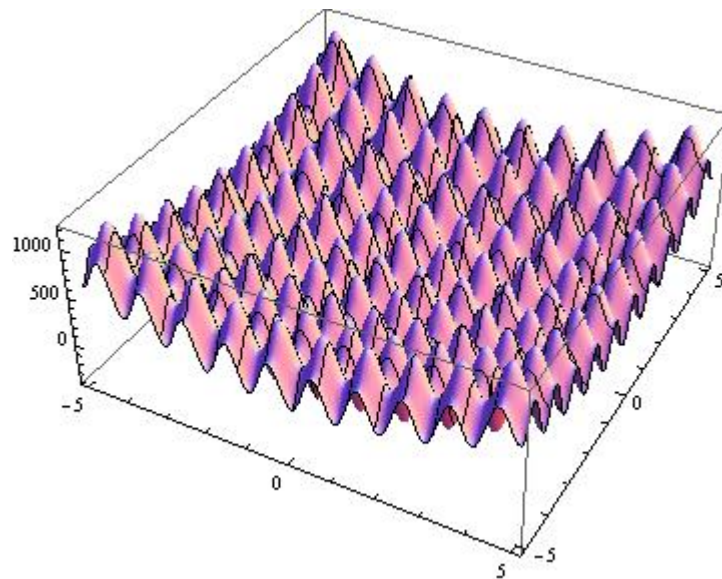
Obrázek 3. druhá de Jongova funkce ve 3D



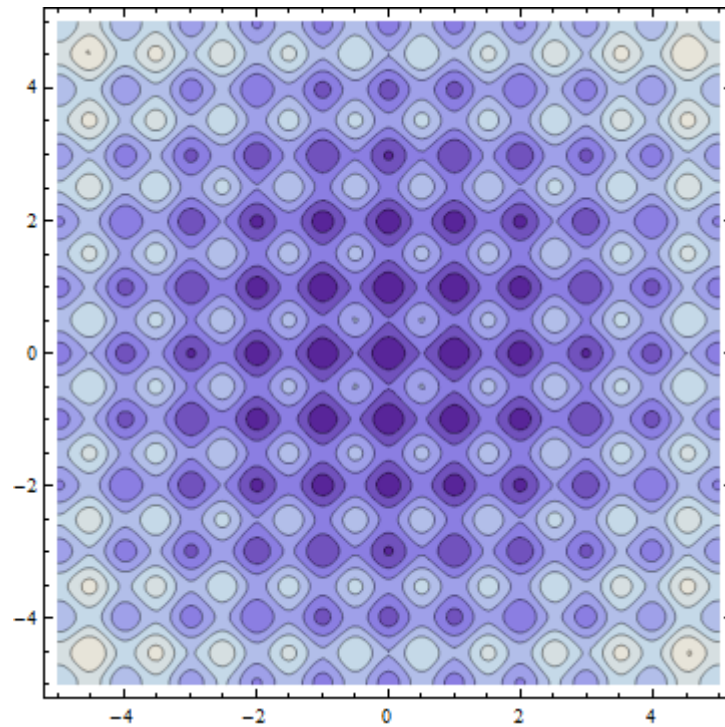
Obrázek 4. druhá de Jongova funkce

7.3 Rastriginova funkce

$$f(x_1, x_2) = 20 \cdot (x_1^2 + x_2^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_1) - 10 \cdot \cos(2 \cdot \pi \cdot x_2)) \quad (22)$$



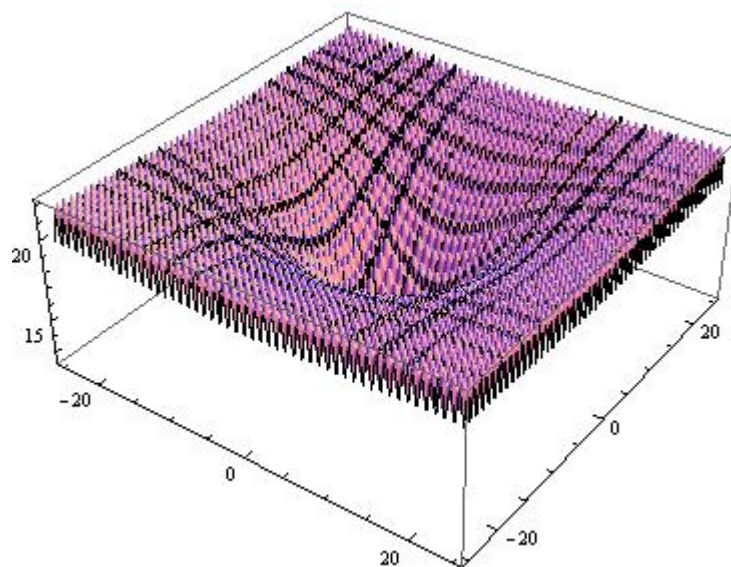
Obrázek 5. Rastriginova funkce ve 3D



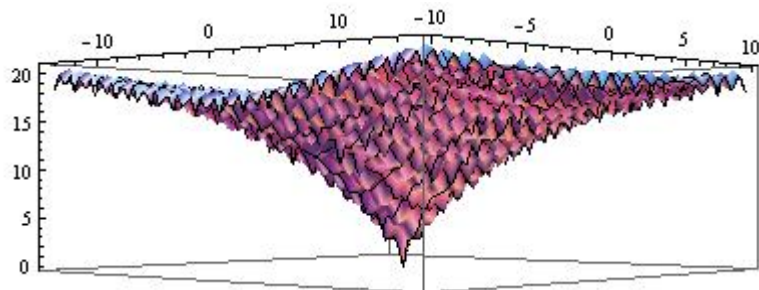
Obrázek 6. Rastriginova funkce

7.4 Ackleyho funkce

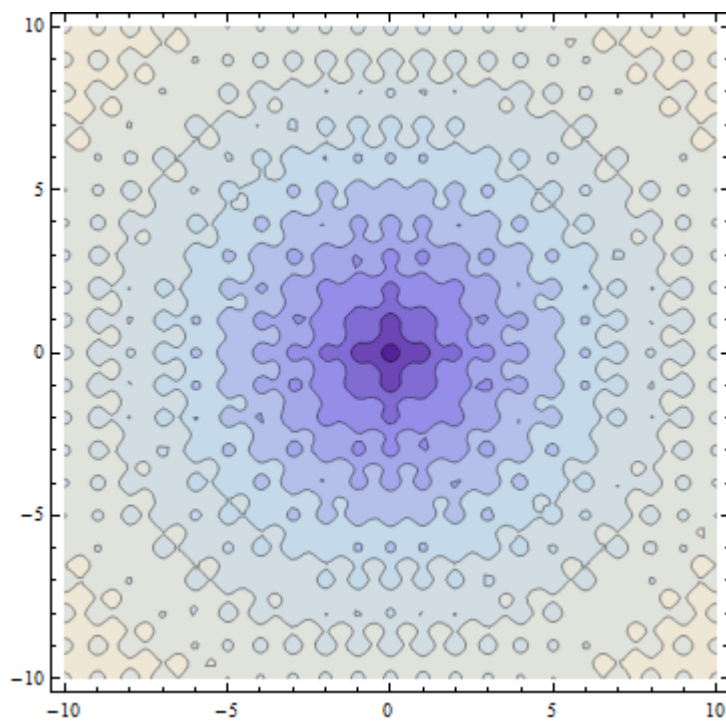
$$f(x_1, x_2) = 20 + e^{-\frac{20}{e^{0.2 \sqrt{\frac{x_1^2 + x_2^2}{2}}}} - e^{0.5(\cos(2\pi \cdot x_1) + \cos(2\pi \cdot x_2))}} \quad (23)$$



Obrázek 7. Ackleyho funkce ve 3D



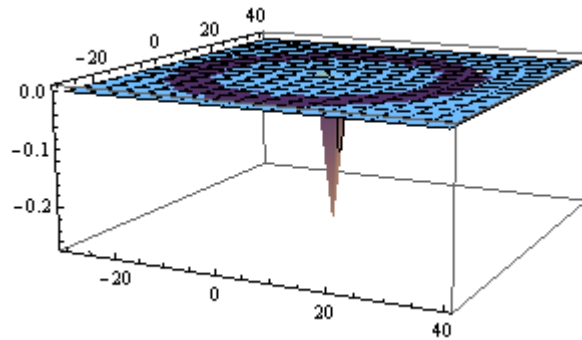
Obrázek 8. Ackleyho funkce ve 3D



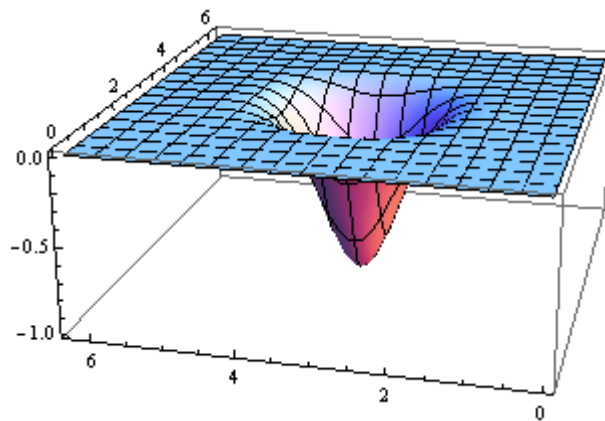
Obrázek 9. Ackleyho funkce

7.5 Easomova funkce

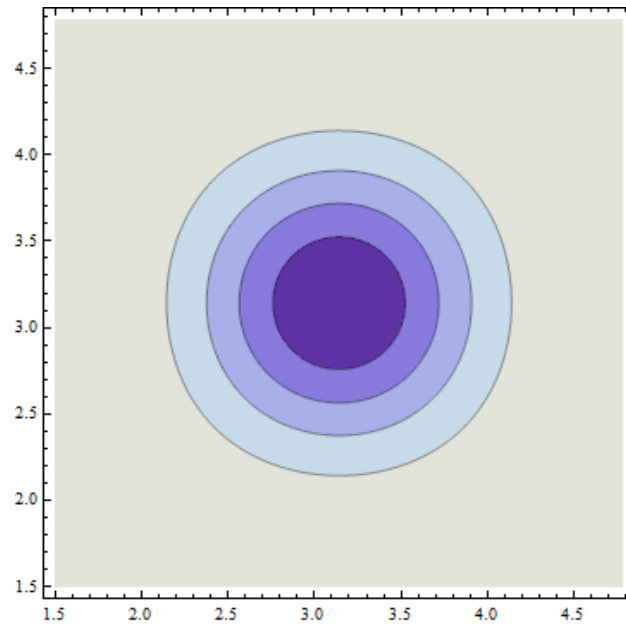
$$f(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-((x_1-\pi)^2 + (x_2-\pi)^2)} \quad (24)$$



Obrázek 10. Easomova funkce ve 3D



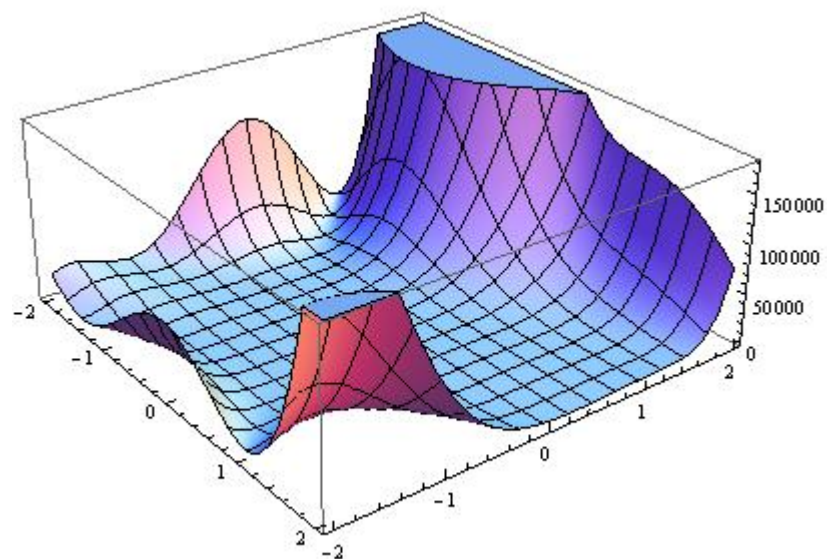
Obrázek 11. Easomova funkce ve 3D



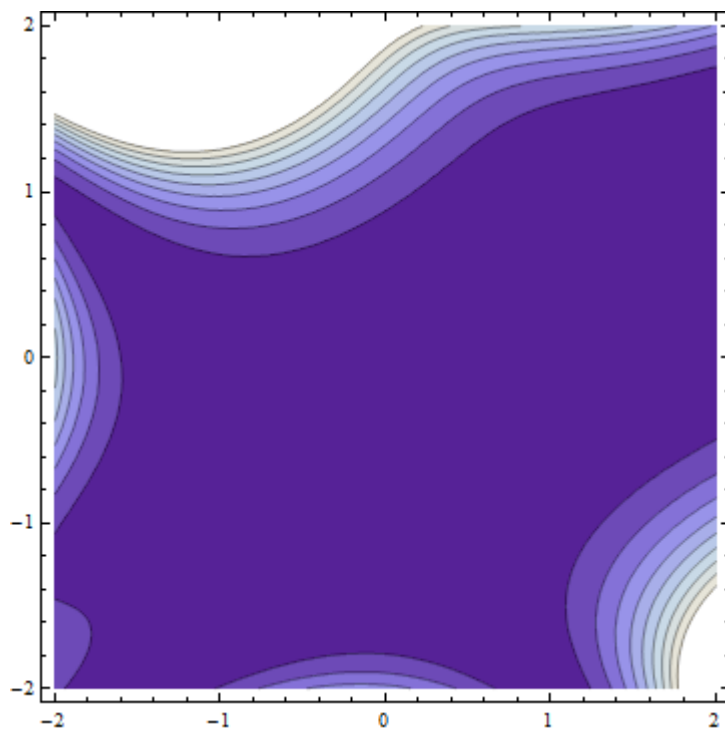
Obrázek 12. Easomova funkce

7.6 Goldsteinova-Priceova funkce

$$f(x_1, x_2) = \left(1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14 \cdot x_1 + 3 \cdot x_1^2 - 14 \cdot x_2 + 6 \cdot x_1 \cdot x_2 + 3 \cdot x_2^2)\right) \cdot \left(30 + (2 \cdot x_1 - 3 \cdot x_2)^2 \cdot (18 - 32 \cdot x_1 + 12 \cdot x_1^2 + 48 \cdot x_2 - 36 \cdot x_1 \cdot x_2 + 27 \cdot x_2^2)\right) \quad (25)$$



Obrázek 13. Goldsteinova-Priceova funkce vs 3D



Obrázek 14. Goldsteinova-Priceova funkce

8 CMUCAM3

Pro otestování výkonu evolučních algoritmů bylo jako embedded zařízení zvolena kamera CmuCam3. CMUcam3 je embedded zařízení, založené na ARM7 architektuře a je plně programovatelná v jazyce C. Kamera byla představena roku 2007 robotickým institutem Carnegie Mellon University. Pro účely této práce bude využit pouze procesor tohoto zařízení, a to na běh evolučních algoritmů. Snímací část zde nebude využita vůbec. [12]



Obrázek 15. Zařízení CMUCam3 [12]

8.1 Popis

Jedná se o barevnou inteligentní kameru. Je složena z mikroprocesoru Philips NXP LPC2106 s jádrem ARM7TDMI. Je zde použit GCC kompilátor s podporou normy ISO/IEC 9899:1999 jazyka C [1]. Jako obrazový snímač zde slouží CMOS senzor Omnicision OV6620/OV7630. Zařízení disponuje slotem na paměťové karty MMC/SD s podporou souborového systému FAT16. Jako komunikační porty slouží SPI a I²C. Napájení je možné čtyřmi AA bateriemi zapojenými v sérii, nebo stejnosměrným zdrojem v napěťovém rozmezí 6-15V a schopným dodávat proud alespoň 150 mA. [12]

8.1.1 Technické parametry

Následující tabulka obsahuje technické parametry kamery CMUCam3. [12]

Tabulka 1. Parametry kamery CMUCam3

Součástka		Hodnota
CPU		
	RAM	64KB
	ROM	128KB
	frekvence	(14-16 MHz)
Zásobník FIFO		
	Kapacita	1MB
	Max Rychlost	50 FPS
CMOS snímač		
	Max. rozlišení	352x288
	Barevná hloubka	8 bitů na pixel
Ostatní		
	Max přenos. Rychlost	115,200 b/s

II. PRAKTICKÁ ČÁST

9 POPIS PROGRAMU

Zdrojový kód je vytvořen v jazyce C a je optimalizován pro běh na embedded zařízení CMUCam3. Kód je proto tvořen podle normy ISO/IEC 9899:1999 [1]. Při tvorbě programu byl kladen důraz na efektivnost kódu. [13]

Import knihoven

Díky těmto použitým knihovnám je možný nejen běh programu vůbec, ale také funkce jako `rand()`, `sin()`, `cos()`, atd.

```
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <float.h>
#include <stdbool.h>
#include <stdio.h>
#include "soma.h"
#include "pso.h"
#include "de.h"
#include "cc3.h"
```

Definice konstant

Definice konstant `PI` a `E` které představují přibližné hodnoty čísla π a Eulerova čísla.

```
#define E 2,7182818284
#define PI 3.1415926535
```

Implementace účelových funkcí

Všechny testované účelové funkce, pro které jsou řešeny globální extrémy, jsou zapsány do funkcí s návratovou hodnotou typu `double`.

```
double fce1(double* params, int D)
{
    double rovnice = 0;
    for(int i = 0; i < D; i++)
    {
        rovnice += params[i] * params[i];
    }
    return rovnice;
};
```

Podobným způsobem jsou definovány i všechny ostatní účelové funkce.

9.1 Implementace evolučních algoritmů

Všechny řešené evoluční algoritmy jsou implementovány do hlavičkových souborů. Tyto evoluční algoritmy jsou volány jako funkce. Všechny evoluční algoritmy mají společné tyto vstupní parametry:

1. Účelová funkce, pro kterou je hledán globální extrém
2. Počet jedinců, kteří budou prohledávat prostor
3. Maximální počet iterací
4. Rozsah prohledávané oblasti

9.1.1 Implementace algoritmu PSO

V následující tabulce jsou uvedeny názvy parametrů a názvy proměnných algoritmu PSO včetně jejich datových typů.

Tabulka 2. Parametry vs. Proměnné algoritmu PSO

Parametr	Proměnná	Typ
i	i	int
n	N	int
d	d	int
D	D	int
t	iterace	int
$v_d(t)$	V	double[]
$pBest$	pBest	double
$gBest$	gBest	double
c_1	c1	double
c_2	c2	double
x	X	double[]
$rand_1$	r1	double
$rand_2$	r2	double

Následující část programu představuje implementaci struktury, ve které jsou definovány proměnné a matice algoritmu PSO.

```
typedef struct StructPSO_data
{
    int D;           // dimenze problemu
    int N;           // pocet jedincu
    double c1;       // socialni konstanta
    double c2;       // kognitivni konstanta
    double w;        // setrvacnost
    int t;           // index iterace
    int max_t;       // maximalni pocet iteraci
    double* X;       // matice N x D, pozice jedincu
    double* B;       // matice N x D, nejlepsi dosazene pozice jedincu
    double* V;       // matice N x D, rychlost jedincu
    double* gB;      // vektor o rozmeru D, pozice nejlepsiho jedince ze vsech
    double* FB;      // vektor o rozmeru N, hodnoty fitness function pro
                    // nejlepsi dosazene pozice jedincu
    double* F;       // vektor o rozmeru N, nove (aktualni) hodnoty fitness
                    // function vsech jedincu
    int b;           // skalar, index nejlepsiho jedince
    int error;       // status chyb
} PSO_data;
```

V následující části kódu jsou inicializovány řídicí parametry PSO, vypočítány parametry setrvačnosti w podle rovnice (5). Je zde také alokována paměť pro matice uchovávající pozice a rychlosti jedinců.

```
PSO_data InitPSO(int D, int N, double c1, double c2, int max_t, double* min,
double* max)
{
    //srand(time(0));

    PSO_data dataPSO;
    dataPSO.error = false;

    // kontrola vstupnich parametru
    if(D <= 0)
    {
        perror("D musi byt kladne cele cislo. \n");
        dataPSO.error = true;
    };
    if(N <= 0)
    {
        perror("N musi byt kladne cele cislo. \n");
        dataPSO.error = true;
    };
    if(max_t <= 0)
    {
        perror("max_t musi byt kladne cele cislo. \n");
        dataPSO.error = true;
    };
    if( min == NULL || max == NULL )
    {
        perror("min a max nesmi byt NULL. \n");
        dataPSO.error = true;
    };
};
```

```
if(dataPS0.error == true)
{
    exit(0);
};
// inicializace
dataPS0.D = D;
dataPS0.N = N;
dataPS0.c1 = c1;
dataPS0.c2 = c2;
dataPS0.w = 0;
dataPS0.t = 0;
dataPS0.max_t = max_t;
dataPS0.X = NULL;
dataPS0.B = NULL;
dataPS0.V = NULL;
dataPS0.gB = NULL;
dataPS0.b = 0;

// vycet fi
double fi = dataPS0.c1 + dataPS0.c2;
dataPS0.w = 2.0 / abs(2.0 - fi - sqrt( (fi * fi) - (4 * fi) ));

// alokace pameti
dataPS0.X = (double*)malloc( dataPS0.N * dataPS0.D * sizeof(double) );
if(dataPS0.X == NULL)
{
    perror("Nedostatek pameti. \n");
    dataPS0.error = true;
    exit(0);
};
dataPS0.B = (double*)malloc( dataPS0.N * dataPS0.D * sizeof(double) );
if(dataPS0.B == NULL)
{
    perror("Nedostatek pameti. \n");
    dataPS0.error = true;
    exit(0);
};
dataPS0.V = (double*)malloc( dataPS0.N * dataPS0.D * sizeof(double) );
if(dataPS0.V == NULL)
{
    perror("Nedostatek pameti. \n");
    dataPS0.error = true;
    exit(0);
};
dataPS0.gB = (double*)malloc( dataPS0.D * sizeof(double) );
if(dataPS0.gB == NULL)
{
    perror("Nedostatek pameti. \n");
    dataPS0.error = true;
    exit(0);
};
dataPS0.FB = (double*)malloc( dataPS0.N * sizeof(double) );
if(dataPS0.FB == NULL)
{
    perror("Nedostatek pameti. \n");
    dataPS0.error = true;
    exit(0);
};
dataPS0.F = (double*)malloc( dataPS0.N * sizeof(double) );
if(dataPS0.F == NULL)
{
    perror("Nedostatek pameti. \n");
```

```

        dataPSO.error = true;
        exit(0);
    };

    // inicializace jedincu
    for(int i = 0 ; i < N; i++) /// fill up the matrix
    {
        for(int d = 0; d < D; d++)
        {
            int pos = d + (i * dataPSO.D); // [i][d] -> d + (i * data.D);
            double r = (double)rand()/(double)RAND_MAX; // nahodne cislo <0,1>
            dataPSO.X[pos] = min[d] + ( r * ( max[d] - min[d] ) );
            dataPSO.B[pos] = dataPSO.X[pos];
            dataPSO.V[pos] = 0.0;
        };

        dataPSO.F[i] = DBL_MAX;
        dataPSO.FB[i] = DBL_MAX;
    };

    // inicializuji pozici nejlepsiho jedince jako pozici prvnioho jedince
    for(int d = 0; d < D; d++)
    {
        dataPSO.gB[d] = dataPSO.X[d];
    }

    return dataPSO;
}

```

Následující funkce představuje samotný algoritmus PSO. Každé volání této funkce představuje jednu generaci algoritmu. Pokud je dosaženo maximálního počtu generací, funkce vrátí hodnotu „false“.

```

bool UpdatePSO(PSO_data* dataPSO)
{
    for(int i = 0 ; i < dataPSO->N; i++)
    {
        // porovnání hodnot ucelove funkce pro nove a puvodni souradnice jedince
        if( dataPSO->F[i] < dataPSO->FB[i] )
        {
            for(int d = 0; d < dataPSO->D; d++)
            {
                //pokud je nova pozice jedince vhodnejsi nez predchozi,
                //ulozi se lepsi hodnota
                int pos = d + (i * dataPSO->D);
                dataPSO->B[pos] = dataPSO->X[pos];
            }

            dataPSO->FB[i] = dataPSO->F[i];
            //pokud je nova pozice jedince vhodnejsi nez celkova nejlepsi, ulozi
            //se lepsi hodnota
            if( dataPSO->FB[i] < dataPSO->FB[dataPSO->b] )
            {
                for(int d = 0; d < dataPSO->D; d++)
                {
                    int pos = d + (dataPSO->b * dataPSO->D);
                    dataPSO->gB[d] = dataPSO->B[pos];
                }
            }
        }
    }
}

```

```

        dataPSO->b = i;
    }
}

for(int i = 0 ; i < dataPSO->N; i++)
{
    for(int d = 0; d < dataPSO->D; d++)
    {
        // nahodne cislo [0,1]
        double r1 = (double)rand()/((double)RAND_MAX);
        // nahodne cislo [0,1]
        double r2 = (double)rand()/((double)RAND_MAX);

        int pos = d + (i * dataPSO->D);

        //vypocet rychlosti jedince
        dataPSO->V[pos] = dataPSO->w * dataPSO->V[pos];
        dataPSO->V[pos] += r1 * dataPSO->c1 * ( dataPSO->B[pos] -
dataPSO->X[pos]);
        dataPSO->V[pos] += r2 * dataPSO->c2 * ( dataPSO->gB[d] -
dataPSO->X[pos]);

        //vypocet nove pozice jedince
        dataPSO->X[pos] += dataPSO->V[pos];
    }

    // zvysi index iterace
    ++dataPSO->t;

    // ukoncovaci podminka na maximalni pocet iteraci
    if(dataPSO->t < dataPSO->max_t)
    {
        return false;
    }
    return true;
}

```

Tato funkce uvolní alokovanou paměť pro všechny alokované matice a pole.

```

void destroyPSO(PSO_data* dataPSO)
{
    if(dataPSO == NULL)
        return;

    if(dataPSO->X != NULL)
    {
        free(dataPSO->X);
        dataPSO->X = NULL;
    }

    if(dataPSO->B != NULL)
    {
        free(dataPSO->B);
        dataPSO->B = NULL;
    }

    if(dataPSO->V != NULL)
    {

```

```

        free(dataPSO->V);
        dataPSO->V = NULL;
    }

    if(dataPSO->gB != NULL)
    {
        free(dataPSO->gB);
        dataPSO->gB = NULL;
    }

    if(dataPSO->F != NULL)
    {
        free(dataPSO->F);
        dataPSO->F = NULL;
    }

    if(dataPSO->FB != NULL)
    {
        free(dataPSO->FB);
        dataPSO->FB = NULL;
    }
}

```

9.1.2 Implementace algoritmu SOMA

Následující tabulka obsahuje seznam parametrů algoritmu SOMA uvedených v textu, názvy jejich proměnných v programu a také jejich datové typy.

Tabulka 3. Parametry vs. proměnné algoritmu SOMA

Parametr	Proměnná	Typ
i	i	int
N	N	int
j	j	int
J	J	int
x	X	double[]
x_L	XL	double[]
ML	ML	double
$PRTVector_j$	PRTv	double[]
t	t	double

Postup při implementaci algoritmu SOMA je shodný s implementací algoritmu PSO. Zásadní rozdíl je ve výpočtu nových souřadnic pro jedince. Děje se tomu tak v následující funkci.

```

bool UpdateSOMA(SOMA_data* dataSOMA)
{
    for(int i = 0 ; i < dataSOMA->N; i++)
    {
        // porovnání hodnot ucelove funkce pro nove a puvodni souradnice
        // jedince
        if( dataSOMA->F[i] < dataSOMA->FB[i] )
        {
            // pokud je nova pozice jedince vhodnejsi nez predchozi, ulozi se
            // lepsi hodnota
            for(int j = 0; j < dataSOMA->J; j++)
            {
                int pos = j + (i * dataSOMA->J); // [i][j] -> j + (i * data.D);
                dataSOMA->B[pos] = dataSOMA->X[pos];
            }
            //ulozi hodnotu ucelove funkce pro aktualniho jedince jako
            // nejlepsi
            dataSOMA->FB[i] = dataSOMA->F[i];

            // pokud je nova pozice jedince vhodnejsi nez celkova nejlepsi, ulozi
            //se lepsi hodnota
            if( dataSOMA->FB[i] < dataSOMA->FB[dataSOMA->b] )
            {
                for(int j = 0; j < dataSOMA->J; j++)
                {
                    int pos = j + (dataSOMA->b * dataSOMA->J);
                    dataSOMA->XL[j] = dataSOMA->B[pos];
                }
                dataSOMA->b = i;
            }
        }
    }

    for(int i = 0 ; i < dataSOMA->N; i++)
    {
        for(int j = 0; j < dataSOMA->J; j++)
        {
            double rnd = (double)rand()/((double)RAND_MAX); // nahodne cislo [0,1]

            int pos = j + (i * dataSOMA->J);

            if (rnd < dataSOMA->prt)
            {
                dataSOMA->PRTv[j] = 0;
            }
            else
            {
                dataSOMA->PRTv[j] = 1;
            };
            //vypocet nove pozice jedince
            dataSOMA->X[pos] = dataSOMA->X[pos] + ( dataSOMA->XL[j] -
dataSOMA->X[pos] ) * dataSOMA->t * dataSOMA->PRTv[j];
        }
    }
}

```

```

// zvysi index iterace
++dataSOMA->ML;

// ukoncovaci podminka na maximalni pocet iteraci
if(dataSOMA->ML < dataSOMA->max_t)
{
    return false;
}

return true;
}

```

9.1.3 Implementace algoritmu DE

V níže uvedené tabulce je uveden přehled názvů parametrů algoritmu DE v textu a název proměnných v kódu.

Tabulka 4. Parametry vs. proměnné algoritmu DE

Parametr	Proměnná	Typ
j	j	int
J	J	int
D	D	int
v	V	double[]
F	F	double
CR	CR	double
Generation	Generation	int

Výpočet nových souřadnic pro jedince.

```

bool UpdateDE(DE_data* dataDE)
{
    for(int i = 0 ; i < dataDE->NP; i++)
    {
        // porovnání hodnot ucelove funkce pro nove a puvodni souradnice
        //jedince
        if( dataDE->F[i] > dataDE->FT[i] )
        {
            //pokud je nova pozice jedince vhodnejsi nez predchozi, ulozi se
            //lepsi hodnota
            for(int d = 0; d < dataDE->D; d++)
            {
                // [i][d] -> d + (i * data.D);
                int pos = d + (i * dataDE->D);
                dataDE->X[pos] = dataDE->V[pos];
                dataDE->F[i] = dataDE->FT[i];
            }
        }
    }
}

```

```
for(int i = 0 ; i < dataDE->NP; i++)
{
    for(int d = 0; d < dataDE->D; d++)
    {
        int x0, x1, x2;
        double rnd = (double)rand()/(double)RAND_MAX; // nahodne cislo <0,1>

        //vyber 3 nahodnych jeincu z populace
        do {x0 = rand()%dataDE->NP;} while (x0 == i);
        do {x1 = rand()%dataDE->NP;} while (x1 == i || x1 == x0);
        do {x2 = rand()%dataDE->NP;} while (x2 == i || x2 == x1 || x2 == x0);

        int pos = d + (i * dataDE->D); // [i][d] -> d + (i * data.D);
        // pokud je PARAMETR CR mensi nez nahodne cislo je spocitan zkusebni
        // vektor
        if (dataDE->CR < rnd)
            dataDE->V[pos] = dataDE->X[x2+ (i * dataDE->D)] + dataDE->f *
(dataDE->X[x0 + (i * dataDE->D)] - dataDE->X[x1 + (i * dataDE->D)]);
        else
            dataDE->V[pos] = dataDE->X[pos];
    }
}

// zvysi index iterace
++dataDE->t;

// ukoncovaci podminka na maximalni pocet iteraci
if(dataDE->t < dataDE->Generation)
{
    return false;
}

return true;
}
```

10 POROVNÁNÍ VÝSLEDKŮ

Výkon zde popsáných a naprogramovaných algoritmů je porovnán přes počet ohodnocení účelové funkce. Pro každou testovací funkci proběhlo měření dvacetkrát pro každý evoluční algoritmus. Vyhledávání bylo vždy ukončeno tehdy, když rozdíl mezi nalezeným extrémem a skutečným globálním extrémem byl menší než 0,0001. Hodnoty uvedené níže v tabulce *Tabulka 5* jsou aritmetickým průměrem naměřených výsledků. Nastavení parametrů algoritmů bylo voleno s ohledem na testované funkce. Rozsah hledané oblasti byl pro všechny dimenze v rozsahu [-2,2] pro Easomovu funkci, a pro všechny ostatní testované funkce byl zvolen rozsah [-10,10] pro každou dimenzi.

Tabulka 5. Počet ohodnocení účelové funkce pro vybrané funkce a algoritmy

	dimenze	počet jedinců	Počet ohodnocení účelové funkce		
			PSO	SOMA	DE
1 de jong	1	30	120	300	150
1 de jong	5	100	8200	5960	4640
1 de Jong	10	200	21800	18800	13400
2 de Jong	2	30	7400	7100	10800
Rastrign	2	30	5900	2550	3540
Ackley	2	30	2800	2400	2280
Easom	2	30	1230	930	1170
Golden-price	2	30	1260	920	1090

Jak je patrné z tabulky *Tabulka 5* tak výsledky tohoto testování ukazují, že zde testované evoluční algoritmy dosáhly podobných výsledků pro vybrané účelové funkce. Celkový počet ohodnocení účelové funkce závisí na počtu provedených iterací a počtu jedinců. Je také důležité vhodně zvolit parametry evolučních algoritmů.

ZÁVĚR

Cílem této bakalářské práce bylo popsat vybrané evoluční algoritmy PSO, SOMA a DE. Dále pak tyto algoritmy realizovat v jazyce C a otestovat jejich výkon na embedded zařízení.

V teoretické části této práce je popsána funkce evolučních algoritmů obecně. Dále jsou podrobněji popsány vybrané evoluční algoritmy. Je zde popsán princip těchto algoritmů, a způsob, jakým tyto evoluční algoritmy prohledávají prostor při hledání globálního extrému účelové funkce.

V praktické části práce jsou implementovány tyto algoritmy v jazyce C. Kód je vytvořen podle normy ISO/IEC 9899:1999 a to z důvodu kompatibility s embedded CMUcam3. Každý evoluční algoritmus byl implementován do hlavičkových souborů, takže je lze použít i v jiných projektech. Výkon těchto evolučních algoritmů byl otestován na embedded zařízení CMUCam3 a výsledky porovnány přes počet ohodnocení účelové funkce.

Jak je patrné z výsledků testování, tak všechny zde testované evoluční algoritmy dosáhly podobných výsledků na jednotlivých testovaných účelových funkcích. Evoluční algoritmy našli v během tohoto testování globální minimum pro každou zde testovanou účelovou funkci. Může se ale stát, že algoritmy během prohledávání nenaleznou globální minimum, ale pouze lokální minimum. Toto riziko lze potlačit zvolením většího počtu jedinců, kteří budou prohledávat oblast, a také vhodně zvoleným nastavením parametrů daného evolučního algoritmu.

ZÁVĚR V ANGLIČTINĚ

The goal of this thesis was to describe the evolutionary algorithms PSO, DE and SOMA. Furthermore, to implement these algorithms in C language and to test the performance of EA on the embedded device.

The theoretical part of this thesis described the function of evolutionary algorithms in general and the detail principles of the selected evolutionary algorithms. It also described the methods of searching global extremes of objective functions for particular evolutionary algorithm.

In the practical part of the thesis, the selected algorithms were implemented in C language. Code is designed in accordance with the standard ISO/IEC 9899:1999 because of the compatibility with embedded CMUCam3. Each evolutionary algorithm was implemented as a header file. This header files can be used in other projects. Performance of these EA was tested on embedded device CMUCam3 and the results were compared over the number of objective function evaluations.

According to the results of the test was evident that all the tested evolutionary algorithms achieved similar results for each tested objective function. Evolutionary algorithms found the global minimum for every tested objective function. It could happen that the algorithms didn't find the global minimum for the objective function. It can be eliminated by selecting larger number of individuals searching for the global minimum. Suitable chosen input parameters for EA increase chance to find global minimum as well.

SEZNAM POUŽITÉ LITERATURY

- [1] International Standard ISO/IEC 9899:1999. Programming languages - C. Geneva: International Organization for Standardization, 1999.
- [2] WEISE, Thomas. Global Optimization Algorithms – Theory and Application [online]. 2009[cit. 2012-05-02]. Dostupné z: <http://www.it-weise.de/projects/book.pdf>
- [3] PANUŠ, Jan. Evoluční algoritmy v optimalizačních problémech veřejné správy. Pardubice, 2008. Disertační. Univerzita Pardubice.
- [4] KRÁL, Tomáš. Knihovna evolučních optimalizačních algoritmů v prostředí Java. Zlín, 2011. Diplomová práce. Univerzita Tomáše Bati ve Zlíně.
- [5] ZELINKA, Ivan, Zuzana OPLATKOVÁ, Miloš ŠEDA, Pavel OŠMERA a František VČELARĚ. Evoluční výpočetní techniky: principy a aplikace. 1. české vyd. Praha: BEN, 2009, 534 s. ISBN 978-052-1880-688.
- [6] KVASNIČKA, V.; POSPÍCHAL, J.; TIŇO, P. Evoluční algoritmy. Bratislava : STU, 2000. 215 s. ISBN 80-227-1377-5.
- [7] AUSTRALIA, The University of Western. Proceedings / 1995 IEEE International Conference on Neural Networks Perth, Western Australia, 27 November - 1 December 1995. Piscataway, NJ: Order from IEEE Service Center. ISBN 0780327683.
- [8] KRAMPL, Jakub. Implementace vybraných evolučních algoritmů v prostředí .NET. Zlín, 2011. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně.
- [9] D. Bratton, J. Kennedy, “Defining a Standard for Particle Swarm Optimization” IEEE Swarm Intelligence Symposium, April 2007, pp.120-127.
- [10] *Informatica: an international journal of computing and informatics*. Ljubljana: The Slovene Society Informatika, 2006. ISSN 0350-5596. Dostupné z: http://www.informatica.si/PDF/31-1/16_Zielinski-stopping%20Criteria%20for%20a%20Constrained.pdf

- [11] POHLHEIM, Hartmut. Examples of Objective Functions. In: *Examples of Objective Functions* [online]. 2006 [cit. 2012-04-25]. Dostupné z: http://www.geatbx.com/download/GEATbx_ObjFunExpl_v38.pdf
- [12] CARNEGIE MELLON UNIVERSITY. CMUcam3 Datasheet. Pittsburgh, 2007. Dostupné z: http://www.superrobotica.com/download/cmucam3/CMUcam3_datasheet.pdf
- [13] PRESS, William H., Saul A. TEUKOLSKY, William T. VETTERLING a Brian P. FLANNERY. Numerical recipes: the art of scientific computing. 3rd ed. Cambridge: Cambridge University Press, 2007, 1235 s. ISBN 978-052-1880-688.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

EA	Evoluční Algoritmy.
PSO	Particle Swarm Optimalization.
SOMA	Self Organized migration Alforithm.
DE	Diferential Algorithm
D	Dimenze
N	Počet částic
pBest	Nejlepší pozice daného jedince
gBest	Pozice nejlepšího jedince.
Vd	Rychlost jedince
Xd	Rychlost jedince
C1,C2	Učící se faktory
w	Setrvačnost
rand	Náhodné číslo
PRT	Perturbace
CR	Práh křížení
F	Mutační konstanta
exp	Exponenciální
bin	Binomiální
C99	Standard ISO/IEC 9899:1999
CMOS	Complementary Metal Oxide Semiconductor
SPI	Serial Peripheral Interface
I ² C	Inter-Integrated Circuit
CPU	Central Processor Unit
FIFO	Zásobník typu First In First Out

SEZNAM OBRÁZKŮ

<i>Obrázek 1. První de Jongova funkce ve 3D</i>	29
<i>Obrázek 2. první de Jongova funkce</i>	30
<i>Obrázek 3. druhá de Jongova funkce ve 3D</i>	30
<i>Obrázek 4. druhá de Jongova funkce</i>	31
<i>Obrázek 5. Rastriginova funkce ve 3D</i>	31
<i>Obrázek 6. Rastriginova funkce</i>	32
<i>Obrázek 7. Ackleyho funkce ve 3D</i>	32
<i>Obrázek 8. Ackleyho funkce ve 3D</i>	33
<i>Obrázek 9. Ackleyho funkce</i>	33
<i>Obrázek 10. Easomova funkce ve 3D</i>	34
<i>Obrázek 11. Easomova funkce ve 3D</i>	34
<i>Obrázek 12. Easomova funkce</i>	35
<i>Obrázek 13. Goldsteinova-Priceova funkce vs 3D</i>	35
<i>Obrázek 14. Goldsteinova-Priceova funkce</i>	36
<i>Obrázek 15. Zařízení CMUCam3 [12]</i>	37

SEZNAM TABULEK

<i>Tabulka 1. Parametry kamery CMUCam3</i>	<i>38</i>
<i>Tabulka 2. Parametry vs. Proměnné algoritmu PSO</i>	<i>41</i>
<i>Tabulka 3. Parametry vs. proměnné algoritmu SOMA.....</i>	<i>46</i>
<i>Tabulka 4. Parametry vs. proměnné algoritmu DE.....</i>	<i>48</i>
<i>Tabulka 5. Počet ohodnocení účelové funkce pro vybrané funkce a algoritmy.....</i>	<i>50</i>

SEZNAM PŘÍLOH

PI CD s textem práce, zdrojovými kódy a výsledky testování.