Vysoké učení technické v Brně
Fakulta informačních technologií

SFC - Soft Computing – 1. projekt

Praktická úloha řešená algoritmem PSO

26. listopadu 2020 Jakub Trubka (xtrubk00@vutbr.cz)

1 Úvod

Tato technická zpráva popisuje projekt, který vznik v rámci předmětu SFC. Projekt se zabývá implementací aplikace demonstrující využití algoritmu optimalizace hejnem částic (dále jen PSO) na praktické úloze. PSO algoritmus je vhodný například na hledání globálního minima funkcí o dvou neznámých, tato optimalizace je v tomto projektu také implementována. Jednotlivé sekce se věnují teorii algoritmu, návrhu, implementaci a experimenty s výslednou aplikací.

2 Teoretický úvod

Zdrojem teoretických informací o algoritmu PSO je [3] a [1]. Tento typ algoritmů vznikl v roce 1995 na základě práce Jamese Kennedyho a Russela Eberharta [2], která se věnovala chování hejna ptáků či ryb a využití tohoto chování v optimalizačních algoritmech. Algoritmus který vzešel ze zmíněné práce je založený na stádě či hejnu jedinců. Každý jedinec v hejnu má vlastní pozici a rychlost a snaží se dosáhnout následujících cílů:

- jedinec se snaží vyrovnávat rychlost se zbytkem hejna,
- jedinci mají tendenci směřovat ke středu hejna,
- jedinci se snaží udržovat určitou vzdálenost od ostatních jedinců, tato vzdálenost je ale dostatečně malá, aby nedošlo k separaci jedince.

Implementace algoritmu se tímto chováním inspiruje. Každý jedinec, neboli částice, má na začátku určenou náhodnou pozici a náhodnou rychlost a v každém kroku provádí pohyb na základě vlastních poznatků a poznatků hejna.

2.1 Algoritmus

Jak již bylo zmíněno, každá částice má polohu a rychlost. Zároveň má částice znalost nejlepší doposud dosažené pozice celého hejna a nejlepší pozice které dosáhla částice, či její sousedství. V každé iteraci je počítána nová rychlost a poloha částice. Rovnice rychlosti je zobrazena v 1 a polohy v 2, kde k představuje index částice a t krok hejna, a vektory x a v představují rychlost, ω je koeficient setrvačnosti a $c_p c_g$ jsou váhové koeficienty kognitivní a sociální (udávají důležitost vlastní paměti a vlivu hejna).

$$\vec{v}_k(t+1) = \omega \vec{v}_k(t) + c_p r_p(\vec{x}_{kbest} - \vec{x}_k(t)) + c_g r_g(\vec{x}_{best} - \vec{x}_k(t)) \tag{1}$$

$$\vec{x}_k(t+1) = \vec{x}_k(t) + \vec{v}_k(t) \tag{2}$$

Algoritmus v jednotlivých krocích vypadá následovně:

1. Nastavte hodnoty koeficientů, počet částic, kroků a maximální chybu výsledku.
2. Pro každou částici v hejnu:
   (a) nastavte náhodnou polohu částice,
   (b) nastavte náhodnou rychlost částice,
   (c) ohodnoťte pozici částice.
3. Nastavte nejlepší doposud dosaženou polohu částice či sousedství.
4. Nastavte nejlepší doposud nalezenou polohu hejna.
5. Pro každou částici v hejnu:

(a) vypočítejte novou rychlost částice,

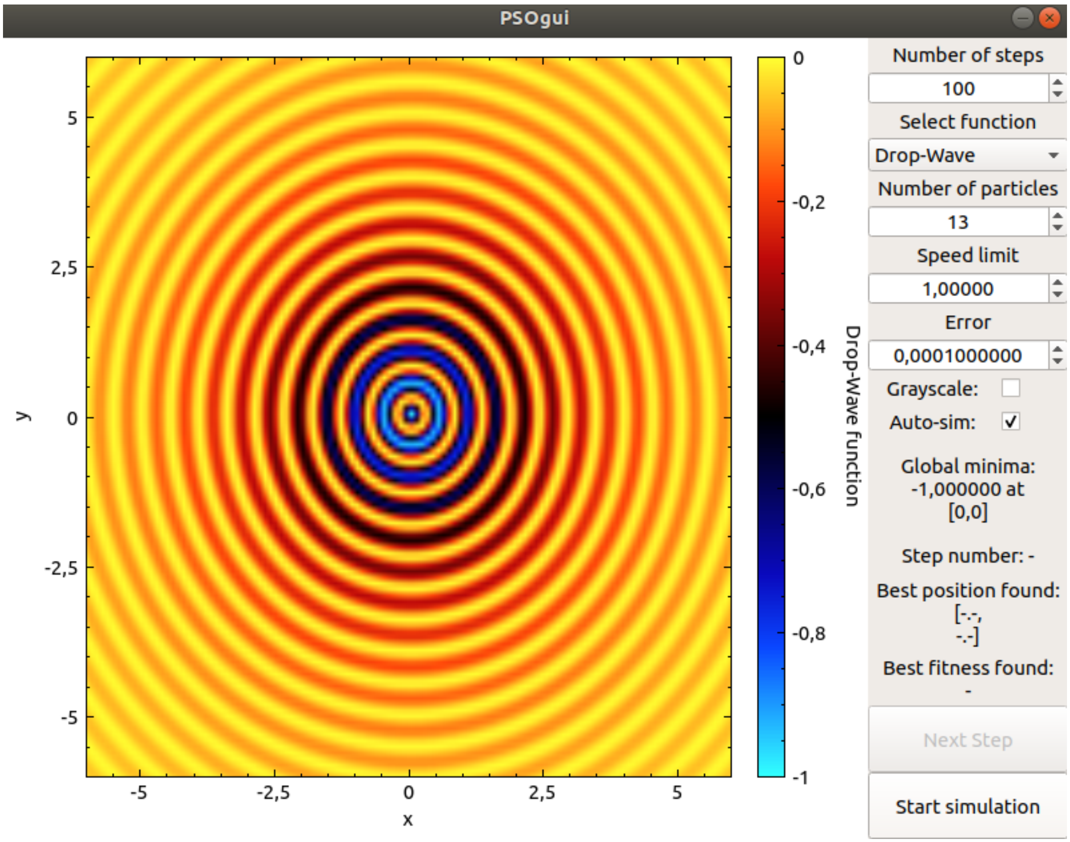
(b) vypočítejte novou polohu částice,

(c) ohodnoťte pozici částice a případně aktualizujte doposud nejlepší nalezenou hodnotu částice či hejna.

6. Inkrementujte počítadlo kroků a pokud se nejednalo o konečný krok opakujte bod 5.

3 Návrh a implementace

Aplikace byla implementována v jazyce C++ a uživatelské grafické rozhraní bylo vytvořeno pomocí nástroje QT5[1]. Grafy jsou vykreslovány pomocí knihovny QCustomPlot[2]. Samotná implementace hejna částic se skládá ze dvou tříd: hejna (Swarm) a částice (Particle). Třída hejna umožňuje provést kompletní simulaci nastaveného počtu kroků či krokování. Je umožněno provádění nastavení počtu kroků, funkce, počtu částic, limitu rychlosti, akceptovatelné chyby. Jednotlivé funkce pro optimalizaci byly implementovány pomocí dědičnosti třídy Function a optimalizačnímu algoritmu se předává přímo instance která počítá zvolenou funkci. Pro počítání polohy a rychlosti byly využity vzorce 1 a 2 s drobnou obměnou, místo nejlepší doposud nalezené hodnoty částice je využito doposud nejlepší poloha nalezená v sousedství. Sousedství se skládá vždy ze tří částic - částice samotné, částice s indexem o jedna vyšším a částice s indexem o jedna nižším.



Obrázek 1: Grafické uživatelské prostředí výsledné aplikace.

[1]https://www.qt.io/
[2]https://www.qcustomplot.com/

3.1 Výsledná aplikace

Grafické uživatelské prostředí je možné vidět na obrázku 1. Aplikace uživateli umožňuje zapnutí krokování a volbu černobílého či barevného zobrazení funkce. Hodnoty zobrazené v grafu funkce jsou počítány a vykresleny pokaždé při zvolení nějaké funkce. Rozsah os je zvolen podle doporučení prohledávaného prostoru pro jednotlivé funkce. Funkční hodnota je zobrazena pomocí teplotní mapy, jednotlivé částice jsou zobrazeny uvnitř tepelné mapy jako bílé, případně červené body (u černobílé tepelné mapy). Rozhraní umožňuje vyplnění počtu kroků, částic, rychlostního limitu, odchylky od výsledku a funkce. Aplikace obsahuje přepínač s názvem *Auto-sim*. Pokud je tento přepínač potvrzen, aplikace provede kompletní simulaci a zobrazeny jsou pouze výsledky a ne mezikroky. Pokud přepínač není potvrzen, aplikace umožňuje spouštět jednotlivé kroky, zobrazovat mezivýsledky a také automatické dokončení simulace. Výsledky jsou zobrazovány po pravé straně a obsahují: globální minimum a bod funkce (global minima), číslo aktuálního kroku, nejlepší doposud nalezená pozice a funkční hodnota. Simulace je ukončena po dosažení počtu kroků, či po nalezené výsledku s rozdílem menším než zvolený rozptyl.

4 Funkce pro optimalizaci

Pro prezentaci využití optimalizace hejnem částic bylo zvoleno patnáct funkcí u kterých je možné tuto optimalizaci provést a sledovat její průběh. Funkce byly vybrány z[3], kde lze nalézt i přepisy funkcí. Zvolené funkce lze rozdělit do několika kategorií podle tvaru - více lokálních minim, tvar misky, tvar údolí a prudké útesy. Informace ke všem funkcím jsou v následující tabulce Přepisy funkcí jsou v příloze.

Název funkce	Globální minimum	Body globálního minima	Rozsah x	Rozsah y
Ackley	0	$[0, 0]$	$< -40, 40 >$	$< -40, 40 >$
Bukin N.6	0	$[-10, 1]$	$< -51, 51 >$	$< -3, 3 >$
Cross In Tray	-2.06261	$[\pm 1.3491, \pm 1.3491]$	$< -10, 10 >$	$< -10, 10 >$
Drop Wave	-1	$[0, 0]$	$< -6, 6 >$	$< -6, 6 >$
Easom	-1	$[\pi, \pi]$	$< -80, 80 >$	$< -80, 80 >$
Griewank	0	$[0, 0]$	$< -100, 100 >$	$< -100, 100 >$
Holder Table	-19.2085	$[\pm 8.05502, \pm 9.66459]$	$< -10, 10 >$	$< -10, 10 >$
Levy	0	$[1, 1]$	$< -15, 15 >$	$< -15, 15 >$
Levy N.13	0.0	$[1, 1]$	$< -10, 10 >$	$< -10, 10 >$
Michalewicz	-1.8013	$[2.20, 1.57]$	$< -4, 4 >$	$< -4, 4 >$
Schaffer N.2	0.0	$[0, 0]$	$< -10, 10 >$	$< -10, 10 >$
Schaffer N.4	0.292579	$[0, 0]$	$< -50, 50 >$	$< -50, 50 >$
Six-Hump Camel	-1.0316	$[\pm 0.0989, \mp 0.7126]$	$< -2, 2 >$	$< -1, 1 >$
Sphere	0.0	$[0, 0]$	$< -6, 6 >$	$< -6, 6 >$
Styblinsky-Tang	-78.3319	$[-2.9035, -2.9035]$	$< -5, 5 >$	$< -5, 5 >$

5 Experimenty

Byla provedena sada experimentů u každé implementované funkce s různými vstupními parametry. Algoritmus optimalizace hejnem částic neměl problém při jednotlivých experimentech rychle najít výsledek do sta kroků u funkcí ve tvaru údolí či misky, kde se nenacházela lokální minima (například funkce Sphere). V případech funkcí obsahující lokální minima byla úspěšnost menší a částice často našli právě jedno z lokálních minim (například funkce Cross In Tray). Algoritmus byl neúspěšný v případě funkcí ve tvaru prudkých útesů (například funkce Schaffer).

[3] https://www.sfu.ca/ ssurjano/optimization.html

6 Překlad a spuštění

Pro úspěšný překlad jsou nutné nástroje cmake3, gcc a QT5. Nástroje lze na linuxové debian distribuci stáhnout následujícími příkazy:

```
$ sudo apt update
$ sudo apt install cmake3 gcc sudo apt-get install qt5-default
```

Projekt lze přeložit následujícími příkazy:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Následně lze aplikaci PSUgui spustit příkazem:

```
$ ./PSUgui
```

7 Závěr

V tomto projektu byla naimplementována aplikace umožňující na jedné z patnácti funkcích o dvou neznámých demonstrovat hledání globálního minima pomocí algoritmu optimalizace hejnem částic. Postup algoritmu v jednotlivých krocích je díky možnosti krokování v aplikaci možno snadno vizualizovat a díky tomu je snadné demonstrovat chování algoritmu při různých vstupních parametrech.

Reference

[1] ALMEIDA, B. S. G. de a LEITE, V. C. *Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems*. prosinec 2019. Dostupné na: <https://doi.org/10.5772/intechopen.89633>.

[2] KENNEDY, J. a EBERHART, R. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*. [b.m.]: IEEE. Dostupné na: <https://doi.org/10.1109/icnn.1995.488968>.

[3] LINDFIELD, G. a PENNY, J. *Introduction to Nature-Inspired Optimization*. 1. vyd. Amsterdam, Boston: Academic Press, 2017. ISBN 978-0-128-03666-2.

Příloha

Ackley

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + a + \exp(1) \tag{3}$$

Bukin N. 6

$$f(x) = 100\sqrt{\left|x_2 - 0.01x_1^2\right|} + 0.01\left|x_1 + 10\right| \tag{4}$$

Cross-In-Tray

$$f(x) = -0.0001\left(\left|\sin(x_1)\sin(x_2)\exp\left(\left|100 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right)\right| + 1\right)^{0.1} \tag{5}$$

Drop-Wave

$$f(x) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5(x_1^2 + x_2^2) + 2} \tag{6}$$

Easom

$$f(x) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right) \tag{7}$$

Griewank

$$f(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_1}{\sqrt{i}}\right) + 1 \tag{8}$$

Holder Table

$$f(x) = -\left|\sin(x_1)cos(x_2)\exp\left(\left|1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right)\right| \tag{9}$$

Levy

$$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2\left(1 + 10\sin^2(\pi w_i + 1)\right) + (w_d - 1)^2\left(1 + \sin^2(2\pi w_d)\right) \tag{10}$$

$$w_i = 1 + \frac{x_i - 1}{4} \tag{11}$$

Levy N. 13

$$f(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2\left(1 + \sin^2(3\pi x_2)\right) + (x_2 - 1)^2\left(1 + \sin^2(2\pi x_2)\right) \tag{12}$$

Michalewicz

$$f(x) = -\sum_{i=1}^{d} \sin(x_i) sin^{20} \left(\frac{ix_i^2}{\pi} \right) \tag{13}$$

Schaffer N. 2

$$f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2} \tag{14}$$

Schaffer N. 4

$$f(x) = 0.5 + \frac{\cos^2\left(\sin(|x_1^2 - x_2^2|)\right) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)} \tag{15}$$

Six-Hump Camel

$$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3} \right) x_1^2 + x_1 x_2 + (-4 + 4x_2^2)x_2^2 \tag{16}$$

Sphere

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{17}$$

Styblinski-Tang

$$f(x) = \frac{1}{2} \sum_{i=1}^{d} (x_i^4 - 16x_i^2 + 5x_i) \tag{18}$$