Dokumentácia k projektu do predmetu Soft computing: Praktická úloha riešená algoritmom PSO

Autor: Juraj Ondrej Dúbrava (xdubra03)

1. Úvod

Cieľom tohto projektu bola implementácia algoritmu Particle Swarm Optimization (PSO) inšpirovaného prírodou a jeho použitie na riešenie praktickej úlohy. Tento algoritmus nachádza uplatnenie najmä pri riešení optimalizačných problémov. Samotný program je implementovaný v jazyku C++ spolu s využitím frameworku Qt [1] na tvorbu grafického užívateľského rozhrania.

Riešenou úlohou pomocou algoritmu PSO je optimalizácia funkcie, konkrétne nájdenie minima zvolenej funkcie. Na optimalizovanie som si zvolil 3 rôzne funkcie, konkrétne Ackleyho funkciu, Schwefelovu funkciu a Schafferovu funkciu z ktorých je v užívateľskom rozhraní možné voliť.

2. Popis problému

Algoritmus Particle Swarm Optimization [2] patrí do kategórie tzv. swarm intelligence a jeho primárne využitie je na riešenie optimalizačných problémov ako napríklad optimalizácia funkcie. Pracuje s populáciou častíc pohybujúcich sa v stavovom priestore problému, častice reprezentujú potenciálne riešenia skúmaného problému. Kvalita riešenia jednotlivých častíc je ohodnotená pomocou tzv. fitnes funkcie.

Každá častica je v priestore reprezentovaná polohou a rýchlosťou, ktorou sa pohybuje. V priebehu algoritmu sa mení rýchlosť a poloha častíc v závislosti na dosiaľ najlepšej pozícií danej častice a najlepšej pozícií v populácií a skúma sa kvalita riešenia daná aktuálnou pozíciou. Ak je kvalita častice lepšia ako globálne najlepšia hodnota populácie, aktualizuje sa táto najlepšia hodnota a rovnako ak je lepšia ako najlepšia hodnota častice, častica si aktualizuje svoju najlepšiu hodnotu a pozíciu. V prípade hľadania minima funkcie bude za lepšiu fitness považovaná menšia hodnota. Rovnica udávajúca zmenu rýchlosti a polohy je nasledovná:

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

Model využívajúci tieto vzťahy sa nazýva globálny, riešením je hodnota na najlepšej pozícií x_best nájdenej celou populáciou. Zmena polohy častice je ovplyvnená tzv. sociálnym a kognitívnym koeficientom, ktoré určujú akú mieru vplyvu na zmenu bude mať globálne riešenie respektíve riešenie častice.

3. Implementácia

Aplikácia je implementovaná v jazyku C++ s použitím frameworku Qt na tvorbu grafického rozhrania. Aplikácia je rozdelená do tried tvoriacich jadro aplikácie a triedu grafického rozhrania.

Triedami jadra sú *Particle* a *PSO* nachádzajúce sa v súboroch Particle.cpp a PSO.cpp, slúžia na realizáciu samotného algoritmu PSO, ktorý je implementovaný v podobe globálneho modelu tohto algoritmu. Trieda *Particle* reprezentuje časticu v populácií. Jej atribútmi sú poloha, rýchlosť, najlepšia nájdená pozícia, najlepšia hodnota častice a hranice pre maximálnu a minimálnu povolenú pozíciu v priestore. Poloha, rýchlosť a najlepšia pozícia sú definované ako dvojrozmerné riadkové vektory typu *RowVector2d*, na použitie tohto typu je využitá matematická knižnica Eigen [3] dostupná na referenčnom serveri Merlin. Využitie tohto typu uľahčuje operácie s vektormi potrebné v realizácií algoritmu, taktiež zápis je týmto spôsobom elegantnejší.

Trieda *PSO* slúži na realizovanie samotného algoritmu. Obsahuje nasledovné atribúty:

- koeficient zotrvačnosti - *inertia*
- Kognitívny koeficient - *cp*
- Sociálny koeficient - *cg*
- Maximálnu povolenú rýchlosť pre častice - *particleSpeed*
- Počet častíc - *n_particles*
- Dimenziu problému - *n_dims*
- Počet iterácií - *n_iter*
- Ukazateľ na optimalizovanú funkciu - *obj_fun*
- Vektor populácie častíc - *population*
- Definičný obor zvolenej funkcie - *boundaries*
- Vektor s najlepšou pozíciou populácie - *globalBestPos*
- Najlepšiu hodnotu nájdenú populáciou - *globalBestVal*

Hodnota dimenzie je pevne nastavená na 2, pretože všetky funkcie zvolené na optimalizovanie v rámci úlohy majú vstup s daným počtom dimenzií. Hlavnými metódami tejto triedy sú *init, initParticles a updatePopulation.* Metóda *init* realizuje inicializáciu potrebných parametrov algoritmu na základe hodnôt zvolených v užívateľskom rozhraní, metóda *initParticles* slúži na vytvorenie počiatočnej populácie častíc s počiatočnými pozíciami a rýchlosťami, nájdenie globálne najlepšej pozície a hodnoty a inicializáciu grafickej reprezentácie častíc. Metóda *updatePopulation* je najdôležitejšou metódou, realizuje rovnicu pre výpočet nových rýchlostí častíc a zmenu pozície všetkých častíc.

Grafické rozhranie

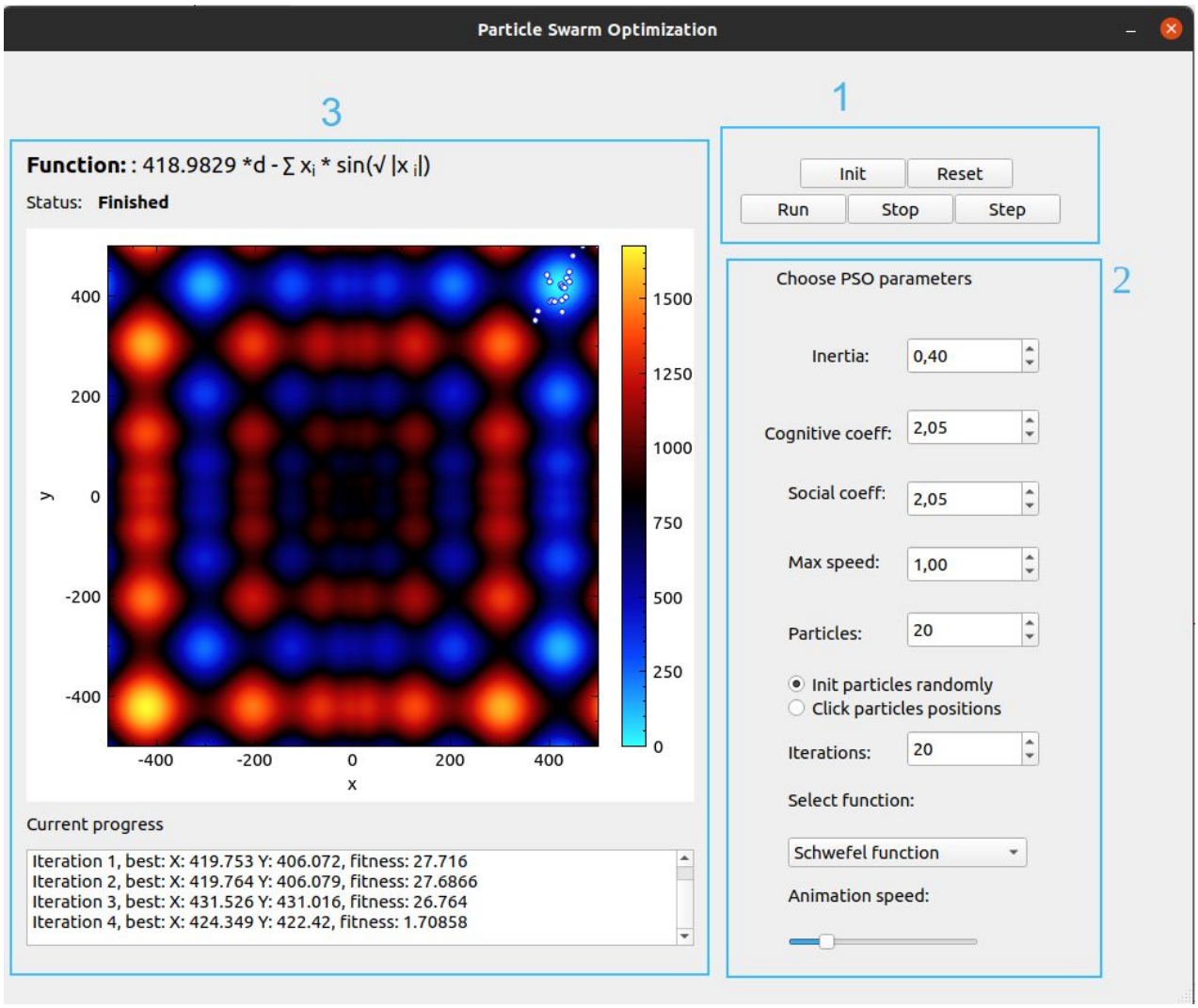
Prvky grafického rozhrania sú definované v súbore PsoMain.ui, hlavnou triedou rozhrania je trieda PsoMain implementovaná v súbore PsoMain.cpp Táto trieda obsahuje inštanciu triedy PSO na realizáciu algoritmu, atribút grafu a tzv. heat mapy na zobrazenie častíc a samotnej funkcie. Jednotlivé funkcie sú vždy zobrazené pre pevne zvolený definičný obor, ktorý je odporúčaný pre skúmanie danej funkcie. Metódy triedy sú *runClicked*, *stepClicked*,

initClicked,*stopCilcked,resetClicked* a reprezentujú signály, ktoré reagujú na stlačenie príslušných tlačidiel rozhrania.

Grafické rozhranie je rozdelené do 3 logických častí:

1. Ovládanie aplikácie
2. Voľba parametrov
3. Zobrazenie funkcie a priebehu algoritmu

Nasledujúci obrázok znázorňuje grafické rozhranie s vyznačenými časťami:



Aplikácia umožňuje zvoliť parametre koeficientu zotrvačnosti, kognitívny a sociálny koeficient, maximálnu možnú rýchlosť častice, počet častíc, počet iterácií, rýchlosť animácie pohybu častíc a optimalizovaní funkciu. Pri časticiach je možné voliť či sa budú inicializovať náhodne alebo je možné zvoliť ich pozície ručne.
Parametre majú prednastavené hodnoty, takže po spustení je možné prejsť priamo na inicializáciu. Na výber sú 3 funkcie, prednastavenou je Ackleyho funkcia. Na ľavej strane sa nachádza zobrazovacia časť, na jej implementáciu som využil QCPColorMap a QCPGraph

pomocou knižnice QCustomPlot [4], ktorej zdrojové kódy sú voľne šíriteľné a sú priložené v projekte kvôli prekladu. Pri mape sa nachádza farebná škála hodnôt funkcie. Nad grafom sa vždy nachádza rovnica vyjadrujúca funkciu a aktuálny status aplikácie, pod grafom sa nachádza textové pole kde sa priebežne zobrazujú výsledky jednotlivých iterácií textovo.

Ovládanie

Ovládanie aplikácie je realizované tlačidlami v časti 1 a ďalšími ovládacími prvkami v časti 2. Ako prvé je potrebné nastaviť parametre algoritmu, zvoliť spôsob inicializácie častíc, funkciu a rýchlosť animácie. V prípade, že je zvolený spôsob zvolenia počiatočných pozícií častíc ručným naklikaním, je potrebné naklikať body ešte pred samotnou inicializáciou, pri spôsobe náhodného generovania sa pozície určia pri inicializácií. Po spustení aplikácie sú všetky parametre nastavené na predvolené hodnoty, čo umožňuje ihneď po spustení prejsť na inicializáciu.

Na inicializáciu aplikácie je potrebné stlačiť tlačidlo *Init***,**inicializácie parametrov a zvolenej funkcie, a prípadne aj pozícií častíc pri voľbe náhodného generovania. V prípade prepínania zvolenej funkcie sa aktuálne zvolená funkcia automaticky zobrazí v heat mape. Po inicializácií sú na grafe zobrazené častice na počiatočných pozíciách a priebeh algoritmu je potrebné spustiť buď použitím tlačidla *Run* alebo pomocou *Step*.

V prípade použitia *Run* sa vykoná metóda *runClicked*, kde sa postupne vykonávajú jednotlivé iterácie algoritmu s časovým oneskorením, aby bolo možné sledovať ich priebeh v podobe pohybu častíc v grafe. Rýchlosť tohto oneskorenia sa volí cez hodnotu slidera v dolnej časti. Priebeh v režime *Run* je možné zastaviť tlačidlom *Stop*, po zastavení je možné pokračovať v priebehu algoritmu opätovným stlačením tlačidla *Run* alebo *Step*. Ďalšou možnosťou je voľba *Step***,** slúžiaca na krokovanie algoritmu a po stlačení tohto tlačidla sa vykoná metóda *stepCicked*, ktorá realizuje 1 iteráciu algoritmu PSO. Je možné použiť aj kombináciu prístupu *Step* a *Run*. Ako prvé sa musí použiť *Step*, ktorým môžeme odkrokovať niekoľko iterácií algoritmu a po stlačení *Run* sa zvyšné iterácie uskutočnia automaticky.

Po skončení zvoleného počtu iterácií je na opätovné spustenie behu potrebné stlačiť tlačidlo *Reset*, čím sa zmažú častice zobrazené v grafe a textový výstup. Následne je nutné opäť zvoliť parametre, inicializovať a spustiť priebeh.

4. Spustenie aplikácie

Na spustenie aplikácie je potrebné uskutočniť 3 kroky: preklad aplikácie pomocou qmake, čím sa vytvorí Makefile súbor. Ďalej samotný preklad pomocou vytvoreného Makefilu a následne je možné spustiť vytvorený program. Tieto kroky sú automatizované v skripte *run_pso.sh*, po ktorého spustení sa aplikácia preloží a vznikne spustiteľný program *PSO*, ktorý sa automaticky spustí. Skript je potrebné spúšťať z adresára kde sa nachádzajú zdrojové súbory. Skript obsahuje aj definíciu premenných prostredia potrebných pre cesty k použitej verzii frameworku Qt na serveri Merlin. Na zobrazenie aplikácie na referenčnom serveri Merlin je potrebné prihlásiť sa pomocou ssh s prepínačom -X.

5. Pozorovania algoritmu na zvolených funkciách

Ackleyho funkcia - táto funkcia má len 1 globálne minimum, čo vedie na dobré riešenie pomocou prednastavených parametrov sociálneho a kognitívneho parametra a rovnako aj váhového koeficientu, minimum sa nájde aj s použitím nie veľkého počtu častí a počtu iterácií.
Schwefelova funkcia - daná funkcia je zaujímavejšia,pretože sa tu nachádza množstvo lokálnych miním, kde môže funkcia skončiť. Pri prednastavených koeficientoch môžeme skončiť v lokálnom minime, ale aj globálnom, pri tejto funkcií bude vhodnejšie voliť sociálny koeficient s vyššou hodnotou ako kognitívny, čo by malo viesť k lepším výsledkom. Vyšší počet častíc by taktiež mohol prispieť k lepšiemu riešeniu. Pozoroval som však, že aj pri prednastavených hodnotách algoritmus vie prísť k dobrému riešeniu.
Schafferova funkcia - v zvolenej variante má globálne minimum v bode 0,0, v ktorého blízkosti sa nachádzajú hodnoty blízke 0 oddelené vyššími hodnotami, kde môže tento algoritmus skončiť, nemusí teda vždy nájsť globálne minimum. Pri vyššom počte iterácií by sa výsledok mohol zlepšiť, pri prednastavených 20 iteráciách však algoritmus produkuje vcelku dobré výsledky.

6. Referencie

[1] https://www.qt.io/product/framework
[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.
[3] http://eigen.tuxfamily.org/index.php?title=Main_Page
[4] https://www.qcustomplot.com/