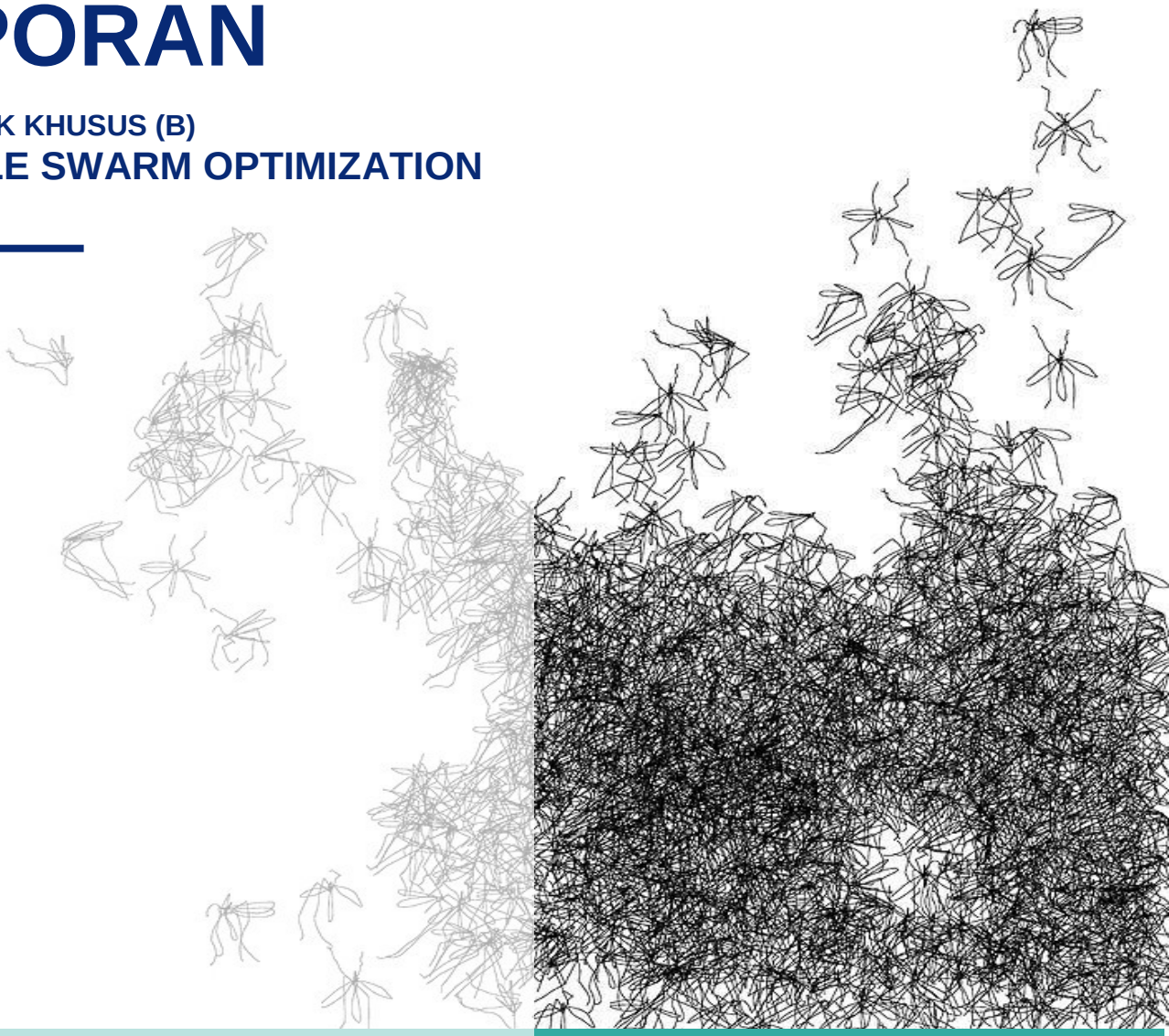
LAPORAN

UTS – TOPIK KHUSUS (B)
PARTICLE SWARM OPTIMIZATION



1 NOVEMBER 2018

TEKNIK INFORMATIKA - ITS
Oleh: Son Ardhynata Sukarno Mudha
NRP: 05111340000107

ITS
Institut
Teknologi
Sepuluh Nopember

Pernyataan

"Demi Allah Yang Maha Esa, maka dengan ini, saya bersumpah dan menyatakan dengan sebenar-benarnya bahwa saya mengerjakan jawaban soal Ujian Tengah Semester (UTS) ini secara sendiri dan mandiri, tidak melakukan kecurangan dalam bentuk apa pun, tidak menyalin/menjiplak/melakukan plagiat pekerjaan/karya orang lain, serta tidak menerima bantuan pengerjaan dalam bentuk apa pun dari orang lain. Saya bersedia menerima semua konsekuensi dalam bentuk apa pun, apabila saya ternyata terbukti melakukan kecurangan dan/atau penyalinan/penjiplakan/plagiat pekerjaan/karya orang lain."

Surabaya, 1 November 2018

Son Ardhynata Sukarno Mudha
05111340000107

Latar Belakang

Particle Swarm Optimization (PSO) adalah metode optimisasi stokastik berdasar populasi (*population-based*) yang diinspirasi dari tingkah laku sosial makhluk hidup yang sering hidup secara berkelompok seperti burung dan ikan laut. Metode ini diperkenalkan oleh Dr. Eberhart dan Dr. Kennedy pada tahun 1995.

PSO memiliki kesamaan dengan teknik komputasi evolusi (*evolutionary computation techniques*) seperti Genetic Algorithm (GA) seperti menginisialisasi populasi dari solusi secara acak dan melakukan pencarian solusi optimum (*optima*) dengan melakukan pembaruan pada generasi selanjutnya secara iteratif. Namun dalam faktor kompleksitas pengimplementasian, PSO dianggap sangat *powerful* karena dengan hanya membutuhkan sedikit parameter – lebih sedikit dari GA – dan tidak diperlukannya *evolution operator* seperti *crossover* dan *mutation*, PSO mampu memberikan hasil yang konsisten dan memuaskan seperti GA. PSO tidak menjamin ditemukannya *optima* yang diinginkan, namun PSO memiliki kinerja yang baik pada problem dengan domain yang memiliki dimensi tinggi, *non-convex*, dan *non-continuous* atau diskrit.

Pada laporan ini akan dijelaskan bagaimana mengimplementasikan PSO sederhana untuk menyelesaikan kasus matematis memaksimalkan fungsi trigonometri dua variabel.

Problem

Seperti pada laporan sebelumnya (IF184981_TK_05111340000107_Son-Ardhynata.pdf), penulis akan menggunakan problem yang sama yakni memaksimalkan fungsi trigonometri:

$$f(x, y) = 21.5 + x \sin(4\pi x) + y \sin(20\pi y)$$
$$-3.0 \leq x \leq 12.1 \ , \ 4.1 \leq y \leq 5.8$$

Degan presisi 4 angka di belakang koma.

Ide

Pengimplementasian PSO dalam laporan ini akan menggunakan pendekatan *object-oriented programming*. Pendekatan OOP dianggap sangat tepat dalam pengimplementasian PSO karena sifat dari PSO yang menirukan objek dunia nyata seperti kumpulan burung atau sekelompok ikan. Objek-objek tersebut memiliki karakteristik dan *behavior* yang dapat diformulasikan menjadi *class* dalam *object-oriented programming*.

Spesifikasi Perangkat

Software:

Programming language	:	Python 3
IDE	:	Jupyter Notebook
Library	:	numpy (Python)

Hardware:

Processor	:	Intel(R) Core(TM) i7-7700HQ
RAM	:	16 GB

Link Code

Code pada laporan ini dapat di-download/pull pada alamat https://github.com/Ardhynata/ParticleSwarmOptimization.git dan dapat di-run langsung menggunakan Jupyter Notebook.

Metode Penyelesaian

1. Mempersiapkan Library dan Class

Library yang digunakan dalam implementasi PSO ini adalah ***numpy***, sebuah library paling umum digunakan untuk menyelesaikan kasus pemrograman yang berhubungan dengan permasalahan matematika. Numpy akan digunakan untuk keperluan sebagai berikut:

- Mencari nilai sinus dari persamaan yang disebutkan di atas
- Melakukan *randomization* bilangan *float*
- Menggunakan nilai π dengan presisi hingga 16 angka di belakang koma

```
1 import numpy as np #numpy for mathematical purpose
2 pi = np.pi
3 sin = np.sin
4 power = np.power
5 randfloat = np.random.uniform
```

Gambar 1. Libary Numpy

Class yang akan digunakan adalah sebagai berikut:

- Class **Problem** untuk menampung domain dari problem dari persamaan yang dijelaskan di atas dan *fitness function* yang dalam hal ini adalah persamaan itu sendiri.

```python
class Problem:

    # x --- lower and upper bound
    x_low = -3
    x_up = 12.1

    # y --- lower and upper bound
    y_low = 4.1
    y_up = 5.8


    @staticmethod
    def fitness_func(x,y):
        #return power(x,0.5) - 1/y
        return 21.5 + x*sin(4*pi*x) + y*sin(20*pi*y)
```

Gambar 2. Class Problem

- Class **Particle** yang merupakan representasi dari solusi-solusi yang mungkin dari persamaan di atas. Class ini bertanggungjawab dalam menghitung *fitness* sebuah partikel yang direpresentasikan oleh sebuah koordinat (x,y) dan menghitung kecepatan (*velocity*) dari partikel tersebut yang direpresentasikan berupa vektor. Partikel-partikel ini memiliki juga memiliki atribut bernama *pBest* untuk menyimpan nilai *fitness* terbaik dari sebuah individu partikel yang nantinya digunakan untuk mencari *gBest* (dijelaskan pada class **Swarm**) yang berpengaruh pada pergerakan individu menuju solusi yang dicari.

- Class **Swarm** memiliki *class attribute* yakni *gBest* yang berguna untuk menyimpan nilai *fitness* terbaik yang pernah dicapai oleh populasi. Nilai *gBest* pada iterasi terakhir adalah solusi dari persamaan di atas.

2. Alur Penyelesaian

i. Initial Population

Metode PSO dimulai dengan men-*generate* populasi awal dengan menentukan jumlah individu partikel yang akan digunakan untuk mencari solusi. Proses ini dilakukan dengan melakukan *randomization* untuk men-*generate* koordinat dalam *vector space* sebanyak jumlah yang ditentukan.

```
:
iteration = 100
n = 20
swarm = Swarm(n)

```

Gambar 3. Instantiation Objek Swarm

Proses *instantiation* di atas akan menginisialisasi variable *n* (jumlah partikel) dalam class *Swarm*

```
class Swarm:

    #gBest
    g_best = None
    g_best_locx = None
    g_best_locy = None

    #instantiating swarm with parameters:
    # n = number of particle(s)
    def __init__(self, n):

        #reset gBest in initial instantiation
        self.reset()

        self.particles = []
        for i in range (0,n):
            self.particles.append(Particle())

    @classmethod
    def reset(cls):
        cls.g_best = None
        cls.g_best_locx = None
        cls.g_best_locy = None
```

Gambar 4. Inisialisasi attribute objek Swarm

Proses di atas akan men-generate partikel-partikel sejumlah *n* dari class *Particle* berikut:

```
class Particle:

    def __init__(self):

        #generate random location (coordinate)
        self.x = randfloat(Problem.x_low, Problem.x_up)
        self.y = randfloat(Problem.y_low, Problem.y_up)

        #generate random velocity
        self.x_vel = randfloat(-1,1)
        self.y_vel = randfloat(-1,1)

        #fitness
        self.fitness = None

        #pBest
        self.p_best = None
        self.p_best_locx = None
        self.p_best_locy = None
```

Gambar 5. Inisialisasi variabel-variabel Particle

ii. Evaluation

Proses evaluasi adalah proses menghitung *fitness* dari sebuah partikel. Proses ini dilakukan dengan memanggil *method* ***calc_fitness*** pada *class Swarm*.

```
    def calc_fitness(self):
        for i in range (0,len(self.particles)):
            self.particles[i].calc_fitness()

```

Gambar 6. Method calc_fitness pada class Swarm

Method di atas akan memanggil *method* ***calc_fitness*** pada seluruh objek-objek **Particle.**

```
21      def calc_fitness(self):
22          self.fitness = Problem.fitness_func(self.x,self.y)
23
```

Gambar 7. Method calc_fitness pada Particle

Sampai pada langkah ini, kita sudah bisa mengetahui nilai *fitness* semua partikel.

```
Partikel 8
x8              = 1.6091925835369638
y8              = 4.443552080012938
x8_vel          = 0.1440856529744965
y8_vel          = -0.33715938258430866
Fitness         = 24.828941851269633
```

Gambar 8. contoh output Fitness pada Partikel 8 dari 20 partikel

iii. Menentukan *pBest* dan *gBest*

Langkah selanjutnya adalah menentukan *local best* (*pBest*) dan *global best* (*gBest*). Pengertian dari *local best* adalah nilai *fitness* terbaik yang pernah dialamai oleh sebuah partikel, sedangkan *global best* adalah nilai *fitness* terbaik yang pernah dialami oleh populasi.

Untuk menentukan nilai *pBest* dan *gBest*, kita harus memanggil *method* ***set_best*** pada *class* ***Swarm***.

```
29      def set_best(self):
30          for i in range (0,len(self.particles)):
31              self.particles[i].set_best()
32              if(self.g_best == None or (self.particles[i].p_best > self.g_best)):
33                  self.g_best = self.particles[i].p_best
34                  self.g_best_locx = self.particles[i].p_best_locx
35                  self.g_best_locy = self.particles[i].p_best_locy
36
```

Gambar 9. Method set_best pada Swarm

Method di atas akan memanggil *method* ***set_best*** pada semua objek-objek **Particle** dan juga memberikan nilai pada *class attribute* ***g_best, g_best_locx,*** dan ***g_best_locy*****.**

```
24      def set_best(self):
25
26          #set pBest
27          if((self.p_best == None) or (self.fitness > self.p_best)):
28              self.p_best = self.fitness
29              self.p_best_locx = self.x
30              self.p_best_locy = self.y
31
```

Gambar 10. Method set_best pada Particle

Langkah-langkah di atas akan menghasilkan output sebagai berikut:

```
Partikel 20
x20             = 4.428682435398629
y20             = 5.654951283066548
x20_vel                 = 0.02640269268601325
y20_vel                 = -0.3584586260413416
Fitness         = 16.310366393550485
pBest           = 16.310366393550485
pBestLoc x      = 4.428682435398629
pBestLoc y      = 5.654951283066548

gBest   = 29.694510203752767
```

Gambar 11. Contoh Output pada partikel 20 dan nilai gBest iterasi pertama

iv. Menghitung *velocity* baru

Sebelum menuju ke iterasi selanjutnya terlebih dahulu kita harus meng-*update* lokasi dari partikel-partikel yang ada. Perpindahan partikel-partikel tersebut dipengaruhi oleh nilai *velocity* yang dimiliki partikel dan juga lokasi dari *pBest* dan *gBest.*

Penghitungan *velocity* baru mengikuti rumus berikut ini:

$$\vec{v}_i(t+1) = w \times \vec{v}_i(t) + r_1 c_1 (\vec{x}_{pBest} - \vec{x}_i(t)) + r_2 c_2 (\vec{x}_{gBest} - \vec{x}_i(t))$$

Implementasi untuk menghitung *velocity* baru adalah dengan memanggil *method* ***calc_velocities*** pada *class* ***Swarm***

```
37      def calc_velocities(self):
38          for i in range (0,len(self.particles)):
39              self.particles[i].calc_velocities(self.g_best_locx, self.g_best_locy)
```

Gambar 12. Method calc_velocities pada Swarm

Method di atas akan memanggil *method* ***calc_velocities*** pada semua objek-objek **Particle**

```
32      def calc_velocities(self, g_best_locx, g_best_locy):
33          w=0.5 # constant inertia weight
34          c1=1  # cognitive constant
35          c2=2  # social constant
36
37          r1 = randfloat(0,1)
38          r2 = randfloat(0,1)
39
40          #cognitive and social
41          x_vel_cognitive = c1*r1*(self.p_best_locx - self.x)
42          x_vel_social = c2*r2*(g_best_locx - self.x)
43
44          y_vel_cognitive = c1*r1*(self.p_best_locy - self.y)
45          y_vel_social = c2*r2*(g_best_locy - self.y)
46
47          self.x_vel = w*self.x_vel+x_vel_cognitive+x_vel_social
48          #dampen velocity for x
49          if(self.x_vel > 5):
50              self.x_vel = 5
51
52          self.y_vel = w*self.y_vel+y_vel_cognitive+y_vel_social
53          #dampen velocity for y
54          if(self.y_vel > 5):
55              self.y_vel = 5
56
```

Gambar 13. . Method calc_velocities pada Particle

Pada *method* di atas telah ditambahkan *if-function* untuk membatasi penambahan kecepatan yang terlalu besar (*velocity dampening*). Sehingga untuk setiap *velocity* baru yang melebihi *threshold* yang ditentukan (dalam hal ini di-set pada angka 5) akan di-set tepat pada nilai *threshold*.

v. Meng-*update* lokasi partikel-partikel

Langkah selanjutnya adalah meng-update lokasi dari partikel-partikel yang ada dalam *vector space.*

Untuk melakukan *update* lokasi dilakukan dengan memanggil *method* ***update_location*** pada *class* ***Swarm***

```
41      def update_location(self):
42          for i in range (0,len(self.particles)):
43              self.particles[i].update_location()
44
```

Gambar 14. Method update_location pada Swarm

Method di atas akan memanggil *method* ***update_location*** pada setiap objek-objek ***Particle***

```
57      def update_location(self):
58          self.x = self.x + self.x_vel
59          #maintain bound
60          if(self.x > Problem.x_up):
61              self.x = Problem.x_up
62          self.y = self.y + self.y_vel
63          #maintain bound
64          if(self.y > Problem.y_up):
65              self.y = Problem.y_up
```

Gambar 15. Method update_location pada Particle

Method di atas ditambahkan *if-function* untuk membatasi agar partikel tidak bergerak keluar batas yang ditentukan.

Lakukan secara berulang-ulang ke lima langkah di atas hingga beberapa kali perulangan (dalam implementasi ini diulang hingga 100 kali perulangan).

Hasil Implementasi

Setelah dilakukan perulangan hingga 100 kali didapatkan *gBest* = 38.73280596954382. Hasil ini adalah hasil yang sangat tepat hingga 4 angka di belakang koma.

Hasil ini dapat di-*benchmark* di WolframAlpha di alamat https://www.wolframalpha.com/input/?i=maximize+21.5+%2B+x*sin(4*pi*x)+%2B+y*sin(20*pi*y)+with+-3.0+%3C%3D+x+%3C%3D+12.1+and+4.1+%3C%3D+y+%3C%3D+5.8

Kesimpulan

Dari percobaan di atas dapat dibuktikan bahwa Particel Swarm Optimization (PSO) mampu memberikan solusi yang tepat hingga 4 angka di belakang koma. Hal-hal yang mempengaruhi performa PSO antara lain adalah jumlah populasi awal dan nilai *velocity dampening*. Semakin banyak partikel yang digunakan, kemungkinan besar akan memberikan hasil yang baik dan menemukan angka yang tepat sebagai *threshold* untuk *velocity dampening* juga akan memberikan hasil yang baik, threshold ini harus diatur sedemikian rupa sehingga tidak terlalu membatasi pergerakan partikel namun juga tidak menjadikan partikel bergerak terlalu bebas.