

Particle Swarm Optimization

Bruno lochins Grisci

Universidade Federal do Rio Grande do Sul

bigrisci@inf.ufrgs.br

23 de maio de 2017

Sumário

1 Inspiração

2 Características

3 Pseudocódigo

4 Aplicações

5 Implementação

6 Resultados

7 Referências

Inspiração

Técnica de otimização inspirada no comportamento de grupos de animais como cardumes de peixes ou enxames de insetos.



Características

- Criado na década de 1990;
- Método baseado em populações estáticas (enxame de partículas);
- Sem qualquer tipo de seleção;
- Utiliza mutação dirigida;
- Opera com métricas multidimensionais;
- Normalmente com valores reais;
- Partículas são mutadas em direção à melhor solução conhecida.

Exemplo

Pseudocódigo

```

1: swarmsize  $\leftarrow$  desired swarm size
2:  $\alpha \leftarrow$  proportion of velocity to be retained
3:  $\beta \leftarrow$  proportion of personal best to be retained
4:  $\gamma \leftarrow$  proportion of the informants' best to be retained
5:  $\delta \leftarrow$  proportion of global best to be retained
6:  $\epsilon \leftarrow$  jump size of a particle

7:  $P \leftarrow \{\}$ 
8: for swarmsize times do
9:    $\xrightarrow{P \leftarrow P \cup \{\text{new random particle } \vec{x} \text{ with a random initial velocity } \vec{v}\}}$ 
10:   $\overrightarrow{\text{Best}} \leftarrow \square$ 
11:  repeat
12:    for each particle  $\vec{x} \in P$  with velocity  $\vec{v}$  do
13:      AssessFitness( $\vec{x}$ )
14:      if  $\overrightarrow{\text{Best}} = \square$  or Fitness( $\vec{x}$ ) > Fitness( $\overrightarrow{\text{Best}}$ ) then
15:         $\overrightarrow{\text{Best}} \leftarrow \vec{x}$ 
16:    for each particle  $\vec{x} \in P$  with velocity  $\vec{v}$  do                                 $\triangleright$  Determine how to Mutate
17:       $\vec{x}^* \leftarrow$  previous fittest location of  $\vec{x}$ 
18:       $\vec{x}^+ \leftarrow$  previous fittest location of informants of  $\vec{x}$                  $\triangleright$  (including  $\vec{x}$  itself)
19:       $\vec{x}^t \leftarrow$  previous fittest location any particle
20:    for each dimension  $i$  do
21:       $b \leftarrow$  random number from 0.0 to  $\beta$  inclusive
22:       $c \leftarrow$  random number from 0.0 to  $\gamma$  inclusive
23:       $d \leftarrow$  random number from 0.0 to  $\delta$  inclusive
24:       $v_i \leftarrow \alpha v_i + b(x_i^* - x_i) + c(x_i^+ - x_i) + d(x_i^t - x_i)$ 
25:    for each particle  $\vec{x} \in P$  with velocity  $\vec{v}$  do                                 $\triangleright$  Mutate
26:       $\vec{x} \leftarrow \vec{x} + \epsilon \vec{v}$ 
27:    until  $\overrightarrow{\text{Best}}$  is the ideal solution or we have run out of time
28:    return  $\overrightarrow{\text{Best}}$ 

```

Retirado de Essentials of Metaheuristics (Sean Luke, 2015).

Aplicações

Usos de PSO incluem:

- Treinamento de redes neurais;
- Telecomunicações;
- Sistemas de energia;
- Otimização combinatorial;
- Processamento de sinais;

A metaheurística é usada principalmente em problemas de otimização unimodais sem restrições, mas também foi demonstrada sua utilidade em problemas multi-objetivo, multimodais, com restrições e com variação dinâmica.

Implementação

- Python;
- Numpy;
- Orientado a Objetos:
 - Modularidade;
 - Reúso;
 - Versatilidade

Representação

Para n átomos: location = [x, y, z, x, y, z] velocity = [x, y, z, x, y, z] locations = $Mn \times 6$ velocities = $Mn \times 6$

Controle dos limites

Novas velocidades podem gerar soluções fora do espaço de busca permitido.

```
#This segment of the code changes the velocity if the result does not respect the boundaries
new_velocity = copy.deepcopy(list(new_vel))
for d in xrange(self.dimensions):
    if self.l_boundaries:
        if self.swarm_location[p][d] + (self.movement_step * new_velocity[d]) < self.lower_bounds[d]:
            if (self.movement_step * new_velocity[d]) != 0.0:
                reductor = (self.lower_bounds[d] - self.swarm_location[p][d]) / (self.movement_step * new_velocity[d]) * 0.99
                new_velocity = [di * reductor for di in new_velocity] #reductor multiplied to all dimensions in order to preserve velocity direction
    if self.u_boundaries:
        if self.swarm_location[p][d] + (self.movement_step * new_velocity[d]) > self.upper_bounds[d]:
            if (self.movement_step * new_velocity[d]) != 0.0:
                reductor = (self.upper_bounds[d] - self.swarm_location[p][d]) / (self.movement_step * new_velocity[d]) * 0.99
                new_velocity = [di * reductor for di in new_velocity] #reductor multiplied to all dimensions in order to preserve velocity direction
```

Controle do laço

```
64 #Start with regular loop
65 for i in xrange(min_iterations):
66     locations = pso.get_locations()
67     if atoms == "ca":
68         scores = evaluator(locations, pdb_ref.get_ca_pos(), pdb_mob.get_ca_pos())
69     elif atoms == "backbone":
70         scores = evaluator(locations, pdb_ref.get_backbone_pos(), pdb_mob.get_backbone_pos())
71     elif atoms == "all":
72         scores = evaluator(locations, pdb_ref.get_all_pos(), pdb_mob.get_all_pos())
73     pso.run_step(scores, initial_lr)
74     print(pso.get_best_location())
75     print(pso.get_best_score())
76     print("Finished first run")
77
78 #Try to optimize the initial best solution reducing the "learning" or movement rate
79 little_i = 0
80 while lr >= initial_lr/10.0:
81     locations = pso.get_locations()
82     if atoms == "ca":
83         scores = evaluator(locations, pdb_ref.get_ca_pos(), pdb_mob.get_ca_pos())
84     elif atoms == "backbone":
85         scores = evaluator(locations, pdb_ref.get_backbone_pos(), pdb_mob.get_backbone_pos())
86     elif atoms == "all":
87         scores = evaluator(locations, pdb_ref.get_all_pos(), pdb_mob.get_all_pos())
88     pso.run_step(scores, lr)
89
90     latest_best_scores.append(pso.get_best_score())
91     if len(latest_best_scores) > 100:
92         latest_best_scores.pop(0)
93
94     if little_i > 101:
95         improvement = (latest_best_scores[0] - latest_best_scores[-1])/latest_best_scores[0]
96         if improvement < 0.01 and lr >= initial_lr/10.0:
97             print(little_i, lr)
98             print(pso.get_best_score())
99             lr = lr/2.0
100            latest_best_scores = []
101            little_i = 0
102
103    little_i += 1
```

Leitura do .pdb

Recupera:

- Nome dos átomos;
- Índice dos resíduos de aminoácidos;
- Coordenadas cartesianas dos átomos;

Alinha os átomos de dois arquivos .pdb.

Translação e rotação

```
13 def align(transformation, mob_atoms):
14     translation = np.matrix([transformation[0:3]]*len(mob_atoms))
15     rot = transformation[3:6]
16
17     rotX = np.matrix([[1.0, 0.0, 0.0], [0.0, math.cos(rot[0]), -math.sin(rot[0])], [0.0, math.sin(rot[0]), math.cos(rot[0])]])
18     rotY = np.matrix([[math.cos(rot[1]), 0.0, math.sin(rot[1])], [0.0, 1.0, 0.0], [-math.sin(rot[1]), 0.0, math.cos(rot[1])]])
19     rotZ = np.matrix([[math.cos(rot[2]), -math.sin(rot[2]), 0.0], [math.sin(rot[2]), math.cos(rot[2]), 0.0], [0.0, 0.0, 1.0]])
20     rotXYZ = rotZ * rotY * rotX
21
22     transformed_atoms = np.matrix(copy.deepcopy(mob_atoms))
23     transformed_atoms = transformed_atoms + translation
24     transformed_atoms = transformed_atoms * rotXYZ.transpose()
25
26     transformed_atoms = np.matrix.tolist(transformed_atoms)
27     return transformed_atoms
```

Resultados

	BACKBONE	CA	ALL
REFERENCE	0.0002	5.84E-007	0.0002
1ACW-01	13.5292	13.5886	14.5882
1ACW-02	14.1559	14.2188	14.8492
1ACW-03	11.5880	11.7586	12.4787
1ACW-04	12.3860	12.3765	13.1628
1ACW-05	12.6750	12.7391	13.6928
1ACW-06	15.6788	15.7296	16.2010

Referências

-  [Sean Luke \(2015\)](#)
Essentials of Metaheuristics
-  [Nikhil Padhye \(2012\)](#)
Boundary Handling Approaches in Particle Swarm Optimization
KanGAL Report Number 2012014.

Fim