Particle Swarm Optimization

Bruno Iochins Grisci

Universidade Federal do Rio Grande do Sul

bigrisci@inf.ufrgs.br

24 de maio de 2017

Sumário

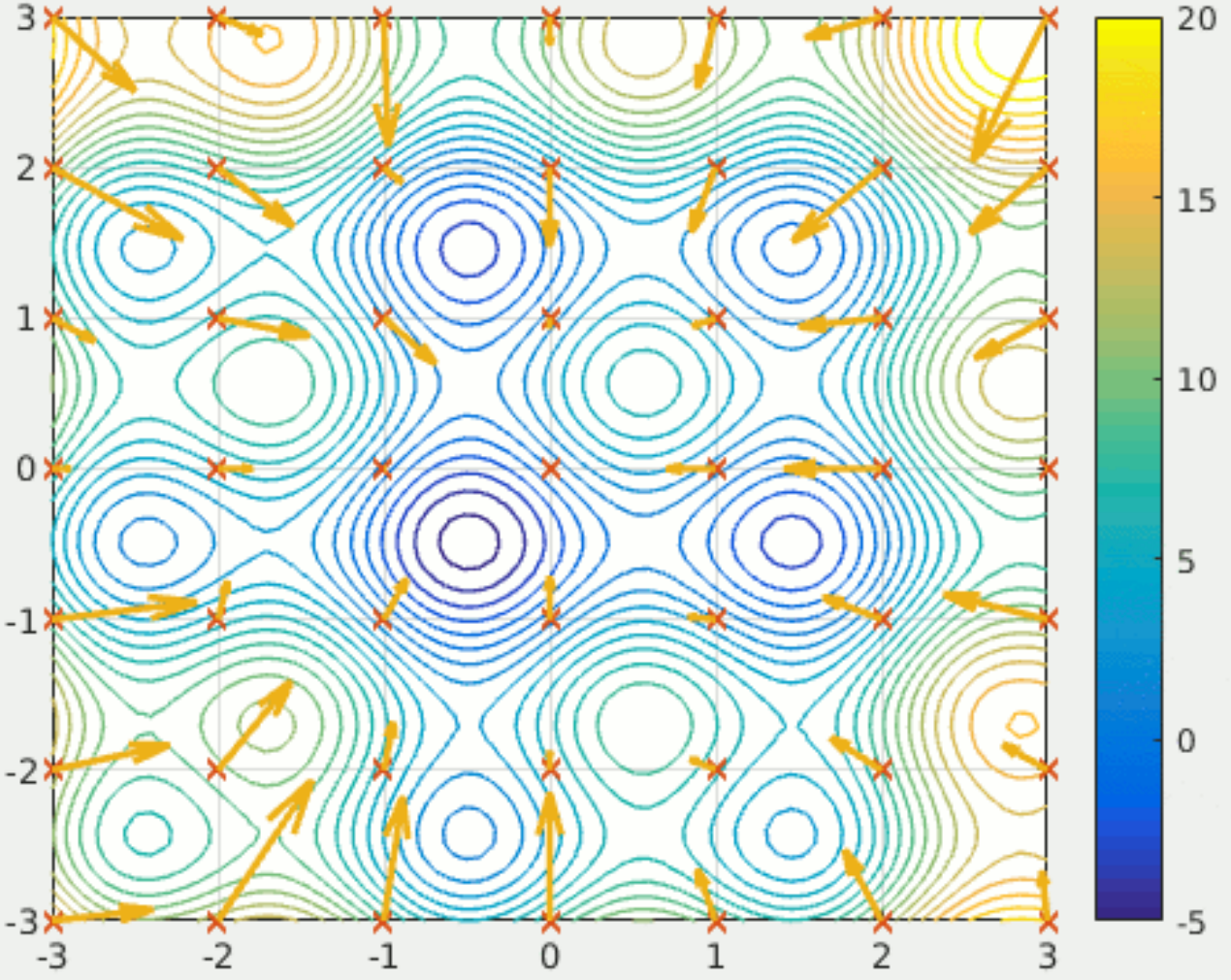
Inspiração

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

$swarmsize \leftarrow$ desired swarm size
$\alpha \leftarrow$ proportion of velocity to be retained
$\beta \leftarrow$ proportion of personal best to be retained
$\gamma \leftarrow$ proportion of the informants' best to be retained
$\delta \leftarrow$ proportion of global best to be retained
$\epsilon \leftarrow$ jump size of a particle

$P \leftarrow \{\}$
for $swarmsize$ times **do**
    $P \leftarrow P \cup \{$new random particle $\vec{x}$ with a random initial velocity $\vec{v}\}$
$\overrightarrow{Best} \leftarrow \square$
repeat
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do**
        AssessFitness($\vec{x}$)
        if $\overrightarrow{Best} = \square$ or Fitness($\vec{x}$) > Fitness($\overrightarrow{Best}$) **then**
            $\overrightarrow{Best} \leftarrow \vec{x}$
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do** ▷ Determine how to Mutate
        $\vec{x}^* \leftarrow$ previous fittest location of $\vec{x}$
        $\vec{x}^+ \leftarrow$ previous fittest location of informants of $\vec{x}$ ▷ (including $\vec{x}$ itself)
        $\vec{x}^! \leftarrow$ previous fittest location any particle
        for each dimension i **do**
            $b \leftarrow$ random number from 0.0 to β inclusive
            $c \leftarrow$ random number from 0.0 to γ inclusive
            $d \leftarrow$ random number from 0.0 to δ inclusive
            $v_i \leftarrow \alpha v_i + b(x_i^* - x_i) + c(x_i^+ - x_i) + d(x_i^! - x_i)$
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do** ▷ Mutate
        $\vec{x} \leftarrow \vec{x} + \epsilon\vec{v}$
until $\overrightarrow{Best}$ is the ideal solution or we have run out of time
return $\overrightarrow{Best}$

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;
- Nunpy;
- 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 = M[n][6]
velocities = M[n][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

```
#Start with regular loop
for i in xrange(min_iterations):
    locations = pso.get_locations()
    if atoms == "ca":
        scores = evaluator(locations, pdb_ref.get_ca_pos(), pdb_mob.get_ca_pos())
    elif atoms == "backbone":
        scores = evaluator(locations, pdb_ref.get_backbone_pos(), pdb_mob.get_backbone_pos())
    elif atoms == "all":
        scores = evaluator(locations, pdb_ref.get_all_pos(), pdb_mob.get_all_pos())
    pso.run_step(scores, initial_lr)
print(pso.get_best_location())
print(pso.get_best_score())
print("Finished first run")

#Try to optimize the initial best solution reducing the "learning" or movement rate
little_i = 0
while lr >= initial_lr/16.0:
    locations = pso.get_locations()
    if atoms == "ca":
        scores = evaluator(locations, pdb_ref.get_ca_pos(), pdb_mob.get_ca_pos())
    elif atoms == "backbone":
        scores = evaluator(locations, pdb_ref.get_backbone_pos(), pdb_mob.get_backbone_pos())
    elif atoms == "all":
        scores = evaluator(locations, pdb_ref.get_all_pos(), pdb_mob.get_all_pos())
    pso.run_step(scores, lr)

    latest_best_scores.append(pso.get_best_score())
    if len(latest_best_scores) > 100:
        latest_best_scores.pop(0)

    if little_i > 101:
        improvement = (latest_best_scores[0] - latest_best_scores[-1])/latest_best_scores[0]
        if improvement < 0.01 and lr >= initial_lr/16.0:
            print(little_i, lr)
            print(pso.get_best_score())
            lr = lr/2.0
            latest_best_scores = []
            little_i = 0

    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	PYMOL
REFERENCE	0.0002	5.84E-007	0.0002	0.000
1ACW-01	13.5292	13.5886	14.5882	13.570
1ACW-02	14.1559	14.2188	14.8492	14.216
1ACW-03	11.5880	11.7586	12.4787	11.758
1ACW-04	12.3860	12.3765	13.1628	6.827
1ACW-05	12.6750	12.7391	13.6928	11.730
1ACW-06	15.6788	15.7296	16.2010	15.729

Referências

Sean Luke (2015)

Essentials of Metaheuristics

Nikhil Padhye (2012)

Boundary Handling Approaches in Particle Swarm Optimization

KanGAL Report Number 2012014.

Fim