Algoritmo PSO

Trabalho desenvolvido para a Disciplina de Inteligência Artificial

Alunos: Maria Luiza e Tarcísio Bruni

Explicação Teórica

O algoritmo de Otimização por Enxame de Partículas (*Particle Swarm Optimization*) ou PSO , é um algoritmo dentro do ramo da Inteligência Artificial e Computação Evolutiva e foi concebido por James Kennedy e Russel Eberhart. O objetivo deste algoritmo está em otimizar um problema de maneira iterativa, baseado em aspectos semelhantes aos da iteração de cardumes e revoadas.

Ele está entre as meta-heurísticas de algoritmos de otimização baseados em padrões da natureza mais populares e é inspirado no comportamento social e cooperativo exibido por várias espécies de forma a realizar as necessidades no espaço de busca. Por se tratar de uma meta-heurística, realiza pouca ou quase nenhuma premissa sobre algum problema que é proposto e com isso pode procurar soluções em grande espaço de busca. Por sua vez há de se observar que:

- Não garante que uma solução ideal seja achada e,
- Guia-se por experiência pessoal (pBest), experiência global (gBest), e o movimento atual das particulas para decidir o próximo espaço a ser analisado.

O algoritmo PSO está dentro das soluções da computação inspiradas nos fenômenos da natureza, como por exemplo a Computação Evolucionária e Inteligência Coletiva – dentre os estudos dessas áreas também estão Algoritmos Genéticos, Programação Coletiva, Evolução Gramatical e Programação Evolutiva.

O algoritmo resolve um problema criando uma população de partículas, como soluções candidatas. Essas partículas são movidas em torno do espaço de pesquisa e tais movimentos são realizados de acordo com fórmulas matemáticas (equação da velocidade) sobre a posição e velocidade correntes de cada membro do bando.

Problema Proposto

O problema sugerido pelo professor envolve utilizar algum algoritmo de otimização por enxame de partículas para minimar a função *Eggholder*, que é uma função clássica na condução de testes para otimização de funções. O objetivo final então, é se aproximar o máximo possível do mínimo global desta função, que é exibida abaixo:

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}$$

Instalação e Execução

A construção do programa utilizou a versão 3 do Python, então recomendamos o uso dessa mesma versão para execução do arquivo

main.py. Segue link da documentação da linguagem para as instalações da versão 3:

- https://docs.python.org/3/using/index.html

Continuando…

- Faça um clone do projeto ou faça o download dos arquivos
- Por meio da linha de comando caminhe até o diretório onde se encontram os arquivos-fonte
- Execute o comando *python main.py*

O comando acima **gera** os arquivos com resultados separados pelos processamentos de número de testes , quantidade de iterações e número da população.

Implementação

A estrutura da implementação tomou como base não somente o pseudocódigo passado pelo professor, mas também por meio de inferências/deduções com base nos materias pesquisados (referências ao final do documento). Para fins de transparência, segue o modelo de pseudocódigo que foi usado como suporte:

1- Determinação do número de partículas
2- Inicialização dos elementos iniciais dentro do domínio especificado
3- Atribuição de velocidade normalizada a todas as partículas
4- Loop iterativo nas partículas processando-as da seguinte forma:

- Calculo da Função Fitness para posição corrente e, definição da melhor posição da partícula

5- Identificação da melhor partícula global (gBest)
6- Loop iterativo nas partículas processando-as da seguinte forma:

- Calculo da nova velocidade, com base na equação. (Para cada dimensão da partícula)
- Atualização da posição em função do cálculo da velocidade e posição anterior

7- Realizar as operações enquanto não chegar na condição de parada

Descrição dos Arquivos:

- *main.py* - Arquivo de chamada principal onde são especificados a quantidade de testes para rodar, a quantidade de iterações do PSO e quantidade de populações.
- *pso.py* - Arquivo com a implementação do algoritmo junto com funções de validação, que são listadas no escopo do problema.
- *Particula.py* - Arquivo com a Classe que representa uma entidade Particula.

> *Contém os atributos de:*

- Posição x atual,
- Posição y atual,
- Velocidade x atual,
- Velocidade y atual,

- Valor do Melhor Fitness para a própria particula,
- Posição x e y do melhor Fitness

- *persistencia.py* - Arquivo com funçoes para exportação dos resultados.

Trechos mais importantes da implementação segundo o Pseudocódigo

Inicialização das Partículas

```python
def gera_populacao_inicial(numero_populacao):
    # Estrutura de dados para armazenar as particulas
    lista_populacao = []

    # 3. Atribua uma velocidade inicial (v) igual para todas as partículas.
    velocidade_x = random.uniform(-77,77)
    velocidade_y = random.uniform(-77,77)

    # Cria as particulas da população
    for _ in range(numero_populacao):
        # 2. Inicialize aleatoriamente a posição inicial (x) (x,y) de cada partícula p de P.
        x = random.uniform(-512,512)
        y = random.uniform(-512,512)

        # Instancia a particula
        particula = Particula(x, y, velocidade_x, velocidade_y)
        lista_populacao.append(particula)

        valor_fitness = particula.calcula_aptidao()
        particula.set_valor_fitness(valor_fitness)

    return lista_populacao
```

Cálculo do Fitness e Checagem de pBest

```
# Calcule sua aptidão fp = f (p). | Calculo do Fitness
valor_fitness = particula.calcula_aptidao()

# Definindo a melhor posição da partícula p até o momento (pBest)
if (valor_fitness < particula.get_valor_fitness()):
    particula.set_valor_fitness(valor_fitness)
    particula.set_x_y_best(particula.x_atual, particula.y_atual)
```

Identificação do gBest

```
# Descobrindo a partícula com a melhor aptidão de toda a população (gBest).
lista_ordenada = list(lista_populacao)
lista_ordenada = sorted(lista_ordenada , key=Particula.get_valor_fitness)
```

Atualização das Velocidades x e y

```
# Atualizando a velocidade da partícula pela fórmula:
# vi(t+1) = (W * vi(t)) + (ϕ1 * rand1 * (pB − xi(t))) + (ϕ2 * rand2 * (gB − xi(t))
constante = const

# Calcula a equacao da velocidade x
a = w * particula.velocidade_x
b = constante * random.uniform(0,1)
c = particula.x_best - particula.x_atual
d = constante * random.uniform(0,1)
e = g_best.x_best - particula.x_best

velocidade_x = a + b * c + d * e

# Verifica se a velocidade x ultrapassou o limite
velocidade = verifica_velocidade(velocidade_x)
particula.set_velocidade_x(velocidade)
```

Atualização das Posições x e y

```
# Atualizando a posição da particula pela fórmula:
# xi(t+1) = xi(t) + vi(t+1)
# Limite de [-512, 512]
# Se ultrapassar o limite, zerar a velocidade
nova_posicao_x = particula.x_atual + velocidade_x

posicao, velocidade = verifica_posicao(nova_posicao_x, velocidade_x)
particula.set_velocidade_x(velocidade)
particula.x_atual = posicao
```

Resultados

As tabelas a seguir mostram os resultados gráficos (média e melhor) de gBest em cada iteração, processados em uma pilha de 10 testes para os casos de:

- 20 Iterações e 50 Indivíduos
- 20 Iterações e 100 Indivíduos
- 50 Iterações e 50 Indivíduos
- 50 Iterações e 100 Indivíduos
- 100 Iterações e 50 Indivíduos
- 100 Iterações e 100 Indivíduos

Referências

- Slides e Aulas em Sala
- PSO, WikiPédia
- Algoritmo de Otimização por Enxame de Partículas, Youtube