Algoritmo PSO

Trabalho desenvolvido para a Disciplina de Inteligência Artificial

Alunos: Maria Luiza e Tarcísio Bruni

Explicação Teórica

O Algoritmo Genético Binário representa o uso computacional da Inteligência Artificial inspirada na Teoria da Evolução das espécies de Charles Darwin. O uso de Algoritmos Genéticos projeta melhores buscas e otimizações dentro de um domínio de interesse.

A aplicação deste código consiste em tentar várias soluções começando por levantamento de alguns indíviduos escolhidos via seleção natural, no qual dentro esse grupo é realizada a aplicação de cruzamento (*crossover*) e calculado uma chance de ocorrência de mutação para o genoma (bits) nas gerações seguintes.

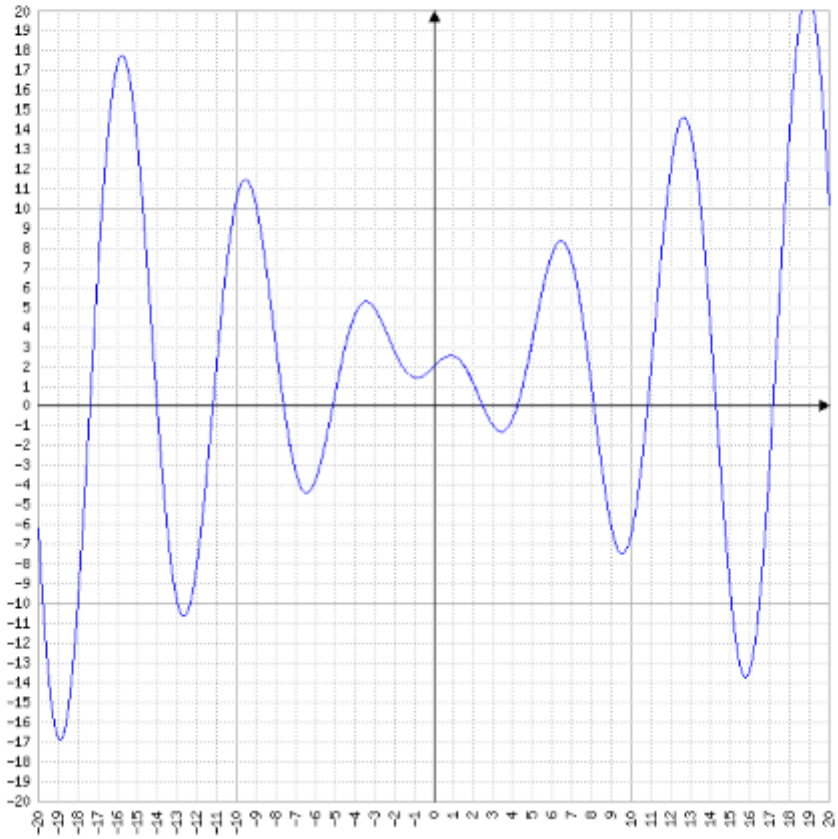
Os algoritmos genéticos possuem algumas características como por exemplo não ser deterministico, trabalhar com uma população de soluções de maneira simultanea e utilizar apenas informações de custo e recompensa. Também são computacionalmente fáceis de serem implementados e são facilmente hibridizados com outras técnicas dentro do campo de pesquisa da Inteligência Artificial.

Problema Proposto

O desafio proposto pelo professor é utilizar a implementação de algum algortmo genético para minimizar a função descrita abaixo:

$$f(x) = \cos(x) * x + 2$$

$$f(-19) = -16{,}785$$



Restrições

Algumas observações realizadas a fim de delimitar o domínio de implementação como por exemplo:

- Assumir de x ∈ [-20,+20]
- Codificar x como vetor binário
- Criar uma população inicial com 10 indivíduos
- Usar seleção por torneio (n=2)
- Aplicar Crossover com taxa de 60% (Crossover de 1 ponto uniforme)
- Aplicar Mutação com taxa de 1%
- Usar 10 gerações e 20 gerações

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- Geração da população inicial
2- Avaliação de cada indivíduo data sua sequência de bits
3- Loop iterativo nas partículas processando-as da seguinte forma:

> - Seleção dos indivíduos mais aptos
> - Criação de filhos no processo de crossover e mutação
> - Armazenamento de dados em uma nova população
> - Nova avaliação dos indivíduos

Descrição dos Arquivos:

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

> Contém os atributos de:
>
> - Valor binário,
> - aptidão,
> - Valor binário decodificado

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

Trechos mais importantes da implementação segundo o Pseudocódigo

Populacao Inicial

```
def gera_populacao_inicial(numero_populacao):
    lista_populacao = []

    for _ in range(numero_populacao):
        valor_binario = monta_valor_binario()
        cromossomo = Cromossomo(valor_binario)
        cromossomo.decodificado = decodificacao(valor_binario)
        cromossomo.aptidao = calcula_aptidao(cromossomo.decodificado)
        lista_populacao.append(cromossomo)

    return lista_populacao
```

Selecao por Torneio

```
for _ in range(0,len(lista_populacao)):
    # Aleatoriamente escolhe dois cromossomos para comparar
    posicao = random.randint(0,len(lista_populacao)-1)
    cromossomo_1 = lista_populacao[posicao]

    posicao = random.randint(0,len(lista_populacao)-1)
    cromossomo_2 = lista_populacao[posicao]

    # Compara qual cromossomo é o melhor (menor aptidao)
    if cromossomo_1.aptidao < cromossomo_2.aptidao:
        lista_selecionados.append(cromossomo_1)
    else:
        lista_selecionados.append(cromossomo_2)
```

Decodificação

```
def decodificacao(valor_binario):
    qtd_bits = len(valor_binario)
    valor_decimal = int(valor_binario, 2)
    return -20 + ( (20+20) * ( valor_decimal / ((2**qtd_bits)-1) ) )
```

Cross Over

```
def crossover(cromossomoA,cromossomoB):
    tamanhoCromossomo = len(cromossomoA.valor_binario)
    pontoCorte = random.randint(1,tamanhoCromossomo-1)
     parteUmA: str
    parteUmA = cromossomoA.valor_binario[:pontoCorte]
    parteDoisA = cromossomoA.valor_binario[pontoCorte:]
    parteUmB = cromossomoB.valor_binario[:pontoCorte]
    parteDoisB = cromossomoB.valor_binario[pontoCorte:]

    valorBinfilhoUm = parteUmA + parteDoisB
    filhoUm = Cromossomo(valorBinfilhoUm)
    filhoUm.decodificado = decodificacao(valorBinfilhoUm)
    filhoUm.aptidao = calcula_aptidao(filhoUm.decodificado)

    valorBinfilhoDois = parteUmB + parteDoisA
    filhoDois = Cromossomo(valorBinfilhoDois)
    filhoDois.decodificado = decodificacao(valorBinfilhoDois)
    filhoDois.aptidao = calcula_aptidao(filhoDois.decodificado)

    return filhoUm,filhoDois
```

Mutação

```
def mutacao(filho):
    enfermeira = ''
    for bit in filho.valor_binario:
        taxaMutacao = random.uniform(0,1)
        if(taxaMutacao <= 0.07):
            if bit == '0':
                enfermeira += '1'
            else:
                enfermeira += '0'
        else:
            enfermeira += bit

    filho.valor_binario = enfermeira
    return filho
```

Remove pior filho

```
#Ordenação dos filhos em ordem crescente de aptidão
auxiliar = lista_populacao_nova.copy()
auxiliar = sorted(auxiliar , key=Cromossomo.get_aptidao)

#Removendo o pior filho
piorFilho = auxiliar[-1].aptidao
```

Mantem melhor pai

```
#Ordenação dos pais em ordem crescente de aptidão
auxiliar = lista_populacao.copy()
auxiliar = sorted(auxiliar , key=Cromossomo.get_aptidao)
melhor_pai = auxiliar[0]
```

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:

- 10 Gerações e 10 Indivíduos
- 10 Gerações e 20 Indivíduos
- 20 Gerações e 10 Indivíduos
- 20 Gerações e 20 Indivíduos

Referências

- Slides e Aulas em Sala