

Algoritmos Genéticos

Carolina Ribeiro Xavier

Março de 2025

1 O que são algoritmos genéticos?

Algoritmos genéticos são algoritmos de busca baseados na mecânica da seleção natural e na genética natural, inspirado na teoria da evolução apresentado por Charlin Darwin 1.

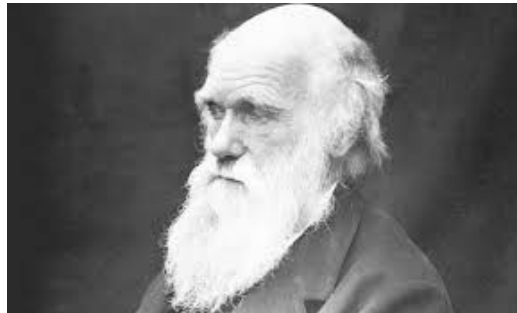


Figure 1: Charles Darwin

Segundo Darwin, os indivíduos mais adaptados ao meio possuem mais chances de propagar suas características, assim, seus hábitos também são passados para novas gerações e acabam por causar uma melhoria na espécie, um exemplo de evolução de espécie é a evolução do próprio *homo sapiens*, como ilustra a Figura 2.

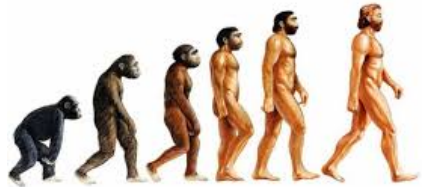


Figure 2: Fases da evolução do *Homo Sapiens*

Os algoritmos genéticos usam estratégias elitistas de sobrevivência dos indivíduos mais aptos e usam fragmentos desses indivíduos para formação de novos indivíduos para próxima geração. Esses algoritmos são genéricos o bastante para resolver problemas de naturezas diferentes, em geral, são promissores para resolver problemas que não possuem soluções analíticas simples e que uma boa aproximação é satisfatória para o usuário.

É um algoritmo que trabalha com um conjunto de soluções candidatas, esse conjunto é denominado população, cada indivíduo dessa população possui uma codificação de uma solução candidata do problema a ser tratado. Esses indivíduos são submetidos a um cálculo de aptidão (fitness) que é realizado por uma função objetivo, essa função mapeia a codificação nos parâmetros que se deseja ajustar através do AG e calcula o quão próxima a solução candidata está da solução desejada. Tanto a codificação quanto a função podem ser direta ou necessitar de funções auxiliares de mapeamento.

Conceitos:

Genes são as características que formam o indivíduo.

Cromossomo ou indivíduo conjunto de genes ou características que formam uma possível solução do problema.

População conjunto de diversos indivíduos ou cromossomos, que são candidatos a solução do problema.

Geração cada etapa ou avanço de passo do AG que produzem uma nova população ou uma alteração expressiva desta.

2 Questões de projeto

Existem várias questões que podem mudar de acordo com o problema a ser resolvido, as mais importantes são as questões relativas à função objetivo escolhida e a representação, sendo que a segunda reflete na implementação de detalhes de todos os operadores genéticos. As principais questões são as listadas a seguir:

- Função objetivo;
- Tipo de representação;
- Estratégia de seleção;
- Cruzamento (taxa) ;
- Mutação (taxa);
- Elitismo

Valores como tamanho da população e número máximo de gerações do AG também devem ser testados para verificar o melhor cenário para solução do problema que se quer resolver.

2.1 Função objetivo

A função objetivo é uma função que avalia a proximidade do indivíduo da solução do problema, esta função pode ser de maximização ou de minimização, pode envolver ajuste de uma curva ou a escolha de uma ordem mais apropriada para dispor um conjunto de elementos. O valor retornado é chamado de fitness ou aptidão do indivíduo, característica importante para seleção de pais e estratégias de reposição da população.

$$f(x) = -20\epsilon^{-0.2\sqrt{\frac{1}{n}\sum x_i^2}} - \epsilon^{\frac{1}{n}\sum \cos(2\pi x_i)} + 20 + \epsilon \quad (1)$$

A Equação 1 é um exemplo de função que pode ser ajustada por um AG e sua forma nos dá o mapeamento direto do valor da função para o valor de aptidão. Esta é uma função multimodal e o nosso objetivo nessa primeira prática é minimizar essa função.

2.2 Tipo de representação

Dependendo da função objetivo e do espaço de busca das variáveis alvo, os tipos de representação podem variar. Muitas vezes a escolha da representação pode criar a necessidade de mapeamento da codificação do indivíduo no parâmetro a ser encontrado propriamente dito.

Para problemas de ajuste de parâmetros, como o da 1 por exemplo, podemos usar um vetor números reais, que refletem exatamente a variável que queremos encontrar, ou uma representação por códigos binários, que aplicando alguma função de transformação, mapeia o código de um indivíduo em um valor factível para a variável alvo.

Para $n = 2$ na 1, por exemplo, teríamos duas variáveis a serem ajustadas, se escolhermos a representação real, então o indivíduo do AG seria um vetor de duas posições do tipo real, daí cada posição é exatamente a solução candidata.

Caso a escolha fosse por uma representação binária ainda teríamos que escolher o número de bits para cada uma das variáveis e uma função para transformar o código binário no número real correspondente. Observe que nesse caso estaríamos discretizando o nosso espaço.

Um exemplo desse mapeamento é dado pela 2, que representa 64 possibilidades de valores para uma variável a princípio contínua com o intervalo possível no espaço de busca de 0,5 à 110. O valor 000000 representa o limite inferior da variável $x_{min} = 0,5$, o valor 111111 representa o valor superior da mesma $x_{max} = 110$.

$$x = x_{min} + \frac{(x_{max} - x_{min})}{2^n - 1} * INT(bin) \quad (2)$$

onde n é o número de bits utilizado e $INT(bin)$ é a função que transforma um código binário em um número inteiro.

$$INT(bin) = \sum_{i=0}^n 2^i * bin[i] \quad (3)$$

2.3 Estratégia de seleção

Uma operação importante que antecede todas as outras é a seleção de pais. Nessa primeira implementação utilizaremos a mais simples, o torneio.

No torneio selecionamos dois indivíduos aleatoriamente na população, comparamos a aptidão de ambos e o melhor entre eles terá uma chance significativamente maior de ser selecionado como um dos pais no cruzamento. Em seguida repetimos o processo excluindo o que já foi selecionado, tendo dois pais escolhidos submetemos esses indivíduos ao cruzamento. Essas operações são realizadas até que tenhamos o número de indivíduos pais igual a duas vezes o tamanho da população.

```
float * torneio(int npop, float *fit){ //considerando minimizacao

    int vpais[npop];
    pv = 0.9;
    int i = 0, vencedor;
    while ( i < npop ){
        p1 = random(0,npop);  p2 = random(0,npop);
        while( p1 == p2 ){
            p2 = random(0,n);
        }
        r = random (0,1);
        if( fit[p2] > fit[p1] ) //p1 e melhor
        {
            vencedor = p1;
            if(r>pv)  vencedor = p2;
        }else // p2 e melhor
            vencedor = p2;
            if(r>pv) vencedor = p1;
        }
        vpais[i] = vencedor;
        i++;
    }
    return vpais;
}
```

2.4 Cruzamento

O cruzamento é uma operação muito importante para **intensificação** do espaço de busca, ele acontece entre dois indivíduos selecionados por algum

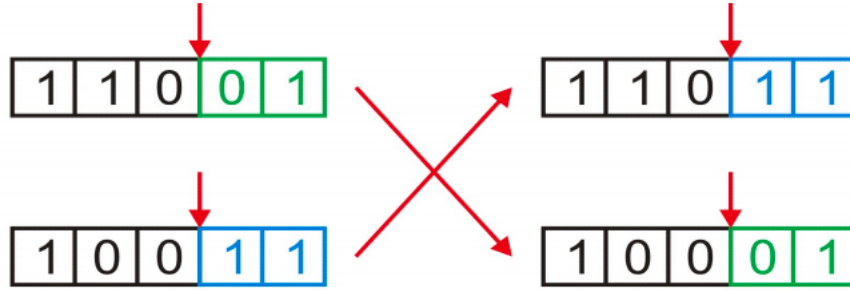


Figure 3: Cruzamento de um ponto

critério.

O tipo cruzamento mais comum para representação por códigos binários é o cruzamento de n pontos. Nesse tipo de cruzamento são gerados, em geral, dois filhos para comporem a nova população. Um exemplo de cruzamento de $n=2$ pontos pode ser ilustrado pela Figura 3.

A cada geração do AG um novo conjunto de indivíduos é gerado para substituir parte ou toda população de indivíduos da geração anterior, uma parte desse indivíduos são gerados através das operações de cruzamento, para isso é preciso definir a taxa que essa operação deve ocorrer. Valores comuns para essa taxa de cruzamento variam de 60% a 100%.

2.5 Mutação

A operação de mutação tem o papel de **diversificação** da população, muitas vezes os indivíduos ficam presos a ótimos locais e precisam de uma perturbação maior para que seja possível escapar desses locais.

Após a aplicação do operador de cruzamento, o operador de mutação é aplicado. Para cada indivíduo (por vezes para cada gene ou alelo do indivíduo) da população é sorteado um valor entre 0 e 1, caso esse valor seja menor ou igual a taxa de mutação estipulada o indivíduo (ou seu gene/alelo) é invertido (mutado).

Para isso é preciso fixar uma taxa de mutação ou alguns critérios de como ela pode variar durante a execução do algoritmo. Valores comuns para essa taxa não costumam ultrapassar 20%, sendo mais comuns valores de 1% a 10%.

2.6 Elitismo

O elitismo é uma estratégia de manutenção da(s) melhor(es) solução(ções) através das gerações. Essa estratégia deixa a solução do algoritmo mais elegante, pois a curva de convergência se torna, no mínimo, monótona, não oscilando quanto a aproximação da solução desejada.

3 Fluxograma

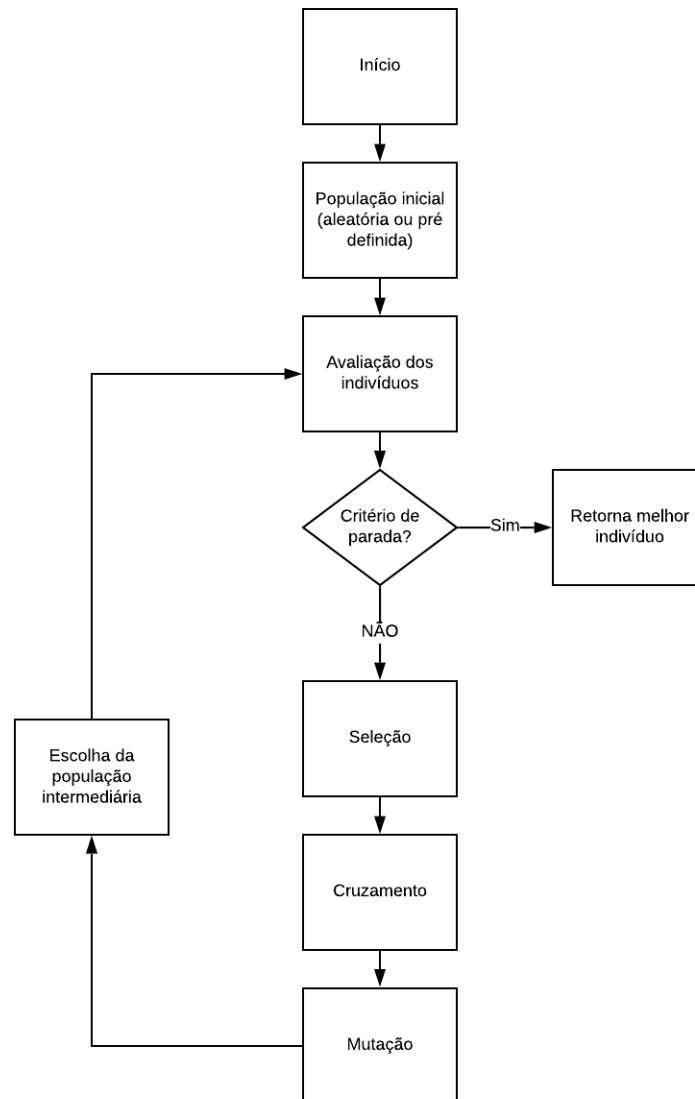


Figure 4: Fluxograma de um algoritmo genético básico

4 Implementação

Agora vamos implementar um AG simples com representação binária dos parâmetros reais da função da Equação 1 de acordo com o fluxograma da Figura 4. Para isso você deve escolher a dimensão da função (pode ser 2 inicialmente), a precisão de cada um dos parâmetros (pode ser a mesma para todos, sugestão 6 bits) e o intervalo de busca para cada um dos parâmetros (sugestão de -2 a 2).

A seguir você deve definir a estrutura de dados que você irá armazenar os indivíduos e seus respectivos valores de fitness, além de uma função que transforme a codificação do indivíduo nos parâmetros da função.

Outros valores a serem definidos são:

- Tamanho da população;
- Número máximo de gerações a serem executadas;
- Critério de seleção de pais (sugestão - torneio)
- Taxa de cruzamento (sugestão - 100%)
- Taxa de mutação (sugestão 10%)
- Sugere-se usar o elitismo, que compreende a sobrevivência do melhor indivíduo para geração seguinte.

5 Pseudocódigo

O código apresentado abaixo é um código em fakeC, que nunca foi compilado, e serve somente para termos uma noção melhor do que devemos implementar nesta prática.

```
void genericAG(){
    int npop = 100; //tamamho da populacao
    int nger = 100; // numero de geracoes
    int nelite = 2;
    int pop[100][6], pop_intermediaria[100][6];

    int pais[100]; //se cada cruzamento gerar 2 filhos
    float fit[100];

    float Pc = 1; // probabilidade de cruzamento
    float Pm = 0.05; // probabilidade de mutacao

    cria_populacao_inicial (npop, pop);

    g = 0;
    while( g < nger){
        fit = avalia_populacao(npop, pop);
        pais = seleciona_pais(npop, pop, fit);
        pop_intermediaria = cruzamento(npop, pais, Pc);
        mutacao(npop, pop_intermediaria, Pm);
        elitismo(nelite);
        pop = pop_intermediaria.copy();
        g++;
    }
    imprime_populacao_final(pop);
}
```