Universidade de Brasília

Campus Darcy Ribeiro

Davi V. R. Feliciano

Título

Brasília

2022

Davi V. R. Feliciano

Título

Dissertação apresentada como Projeto de Trabalho de Conclusão de Curso para obtenção do grau de Bacharelado em Física pelo programa de graduação do Instituto de Física da Universidade de Brasília.

Universidade de Brasília

Campus Darcy Ribeiro

Orientador: Luiz Antonio Ribeiro Junior

Brasília

23 de abril de 2022

Resumo

Nas diversas áreas do conhecimento humano é bem conhecida uma série de problemas de otimização. A solução desses problemas consiste em encontrar em um conjunto de configurações, um subconjunto, ou mesmo um elemento que melhor satisfaça um ou mais vínculos previamente determinados. Uma estratégia famosa por solucionar problemas de tal classe de forma rápida e eficiente consiste no emprego dos algoritmos genéticos. São assim chamados devido à forte inspiração em fenômenos da biologia evolutiva — como mutação, recombinação (ou *crossover*) e seleção — na elaboração de suas etapas de execução. Neste trabalho é proposto um algoritmo genético para a otimização de funções reais, isto é, para a procura dos pontos nos quais a função é máxima (ou mínima). Sua implementação é feita em Python com o uso da biblioteca NumPy. O algoritmo é aplicado na otimização de algumas funções de exemplo, e é feita uma discussão acerca dos resultados, e da eficácia e performance do programa.

Palavras-chaves: Algoritmo Genético. Otimização Numérica. Python. NumPy.

Abstract

Escreva o abstract aqui.

Keywords: Genetic Algorithm. Numerical Optimization. Python. NumPy.

Lista de ilustrações

Sumário

1 Introdução

1.1 Problemas de Otimização

Em matemática e ciência da computação, problemas de otimização consistem naqueles em que a solução é um elemento de um conjunto de candidatos que melhor satisfaz uma determinada série de condições. Um exemplo clássico dessa classe é o problema da mochila (*knapsack problem*, em inglês)[1], no qual procura-se, dentre um conjunto de itens com diversos preços e pesos, o subconjunto que maximiza o valor total. Esse valor é calculado somando-se os preços dos itens postos na mochila.

Entretanto a solução deve respeitar um vínculo: o peso total dos objetos escolhidos não deve exceder o peso máximo suportado pela mochila. Tal problema não é NP-completo, e, ainda que seja possível encontrar a solução exata usando algoritmos de programação dinâmica, a complexidade de tais algoritmos é $\mathcal{O}(nw)$, onde n é o número de itens e w é a capacidade da bolsa. Assim, o tempo de execução para muitos itens — que é o caso de interesse, usualmente — pode ser mais longo do que o desejado.

Outro exemplo comum de problema de otimização é o de otimização numérica, do qual constituem solução os pontos de máximo ou mínimo global de uma função $f : \mathbb{R}^n \to \mathbb{R}$. Tomemos como exemplo simples a função

$$f(x, y) = \cos^2(n\pi r) \exp\left(-\frac{r^2}{\sigma^2}\right) , \tag{1.1}$$

definida no domínio $[-1, 1] \times [-1, 1]$, cujo gráfico se encontra na figura 1. Analiticamente é fácil determinar o ponto de máximo global em $r = 0$ usando métodos de cálculo para funções de múltiplas variáveis.

Contudo, não é sempre que pode-se dispor de tal facilidade. Nos casos em que a função objetivo é de maior complexidade, com domínio contido espaços de maior dimensão; em que a função tem evolução temporal; em que a função possui diversas descontinuidades; ou em que a função depende de variáveis aleatórias, o problema se torna impraticável de resolver de forma analítica. Isto posto, como desenvolver um algoritmo para encontrar a solução?

Uma classe de algoritmos mais simples que se propõem a resolver esse problema é são os algoritmos do tipo *hill climbing*. Sugerido pelo nome, o funcionamento desses algoritmos consiste, resumidamente, em sortear um ponto inicial no domínio da função, calcular o gradiente da função no ponto, e seguir na direção resultante por uma distância pré-definida, e repetir esses passos até que o módulo do vetor gradiente seja próximo de zero, respeitando uma tolerância previamente determinada.

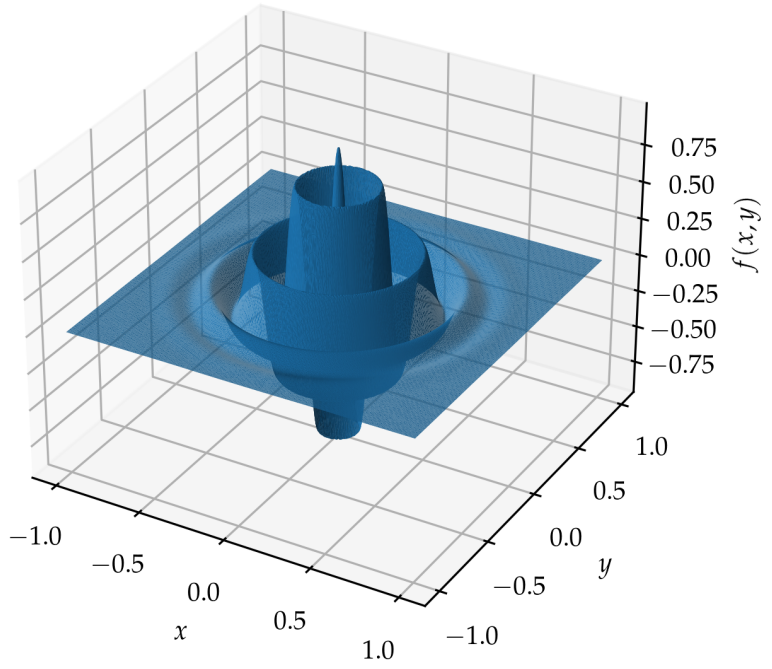


Figura 1 – Gráfico da função definida na equação 1.1, com $n = 9$ e $\sigma = 0,4$.

O problema com esse tipo de algoritmo é que, quando aplicado em funções como a definida anteriormente, a tendência é de que a solução encontrada seja um máximo local na grande maioria das vezes. No caso da função definida pela equação 1.1, o máximo local só seria encontrado pelo algoritmo se o ponto inicial fosse tal que $r < 1/2n$. Se os pontos iniciais do algoritmo forem gerados aleatoriamente, para a domínio definido, a probabilidade de um ponto pertencer a esta região é $P(n) = \pi/16n^2$. Para $n = 9$, como na figura 1, $P \approx 0,2\%$.

Ademais, é possível que esse tipo de algoritmo indique erroneamente como solução pontos de sela e planícies em determinados casos, o que o torna ainda menos eficaz para o propósito.

Uma outra categoria de algoritmos que podem resolver ambos os problemas de forma rápida e chegar a uma solução que se aproxima o suficiente da solução exata são os algoritmos genéticos. Tais algoritmos, quando bem implementados podem chegar a uma solução aproximada para o problema da mochila de forma rápida para valores de n e w ordens de grandeza superiores aos que tornariam praticável o uso da estratégia proposta anteriormente.

No problema de otimização numérica, um algoritmo desse tipo é vantajoso pois é capaz de chegar na solução de forma rápida na maioria dos casos, identificando não só o máximo global, como também os máximos locais contidos na região de busca. Outra vantagem é que esse método pode ser aplicado sem problemas em espaços de busca com grande número de dimensões, ou em funções não estacionárias, contínuas ou descontínuas. Além disso, como será mostrado posteriormente, o algoritmo é paralelizável, o que pode acelerar a obtenção de uma solução.

1.2 O Algoritmo Genético

Os algoritmos genéticos foram primeiramente desenvolvidos por John H. Holland(2)(3) na década de 60. Levam esse nome pois sua formulação teve como forte influência o processo de evolução natural que fomenta a origem das espécies desde o surgimento da vida em nosso planeta. Em sua execução, é inicializada uma população de indivíduos da mesma espécie[1]. Cada indivíduo corresponde a um candidato a solução do problema de otimização proposto, e tem sua identidade codificada pelo seu material genético[2].

Então, sob algum critério, é selecionada uma parcela desses indivíduos, a qual dará origem a novos descendentes, cujos cromossomos serão gerados pela recombinação dos cromossomos correspondentes nos pais. Nesse passo é introduzida uma probabilidade de mutação nos genes dos filhos.

Ao final desse processo, descarta-se a parcela não selecionada da população, dando origem a uma nova geração, formada pelos pais seus filhos. Esse processo é repetido iterativamente até que uma condição de parada seja atingida. Essa condição pode ser imposta sobre o número de gerações ou sobre o tempo de execução do programa, por exemplo.

Respeitando as diferenças de implementação em cada passo, devido as peculiaridades de cada tipo de problema de otimização proposto, essas etapas são presentes em todo algoritmo genético, e são bem resumidas no fluxograma presente na figura 2.

---

[1] Isso significa que seus respectivos códigos genéticos possuem a mesma estrutura: mesma número de cromossomos com uma mesma quantidade de genes. Assim, pode haver reprodução entre tais indivíduos.

[2] No caso do problema da mochila, por exemplo, uma estrutura natural para o material genético é um cromossomo binário, com n genes, cada um correspondente a um objeto do problema. Por outro lado, no problema de otimização numérica, dois cromossomos seriam necessários, um para codificar cada coordenada em cada dimensão do espaço de busca.

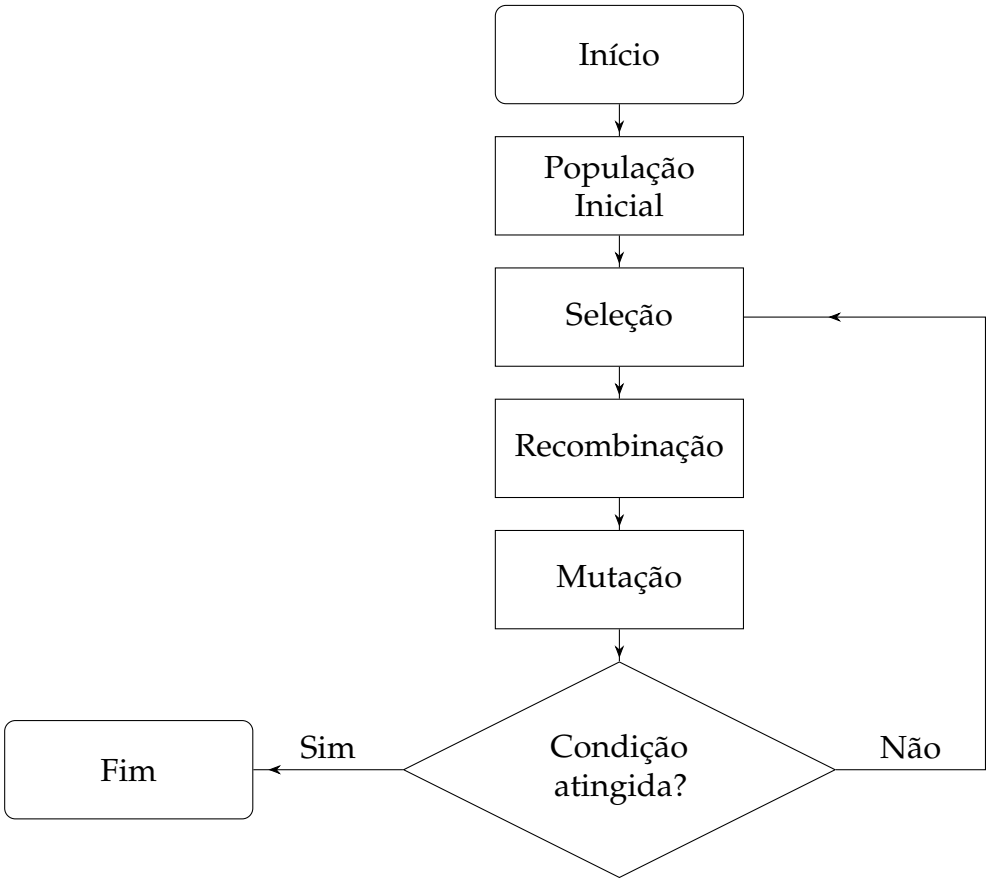


Figura 2 – Fluxograma geral de um algoritmo genético.

1.3 A Implementação

A linguagem de programação escolhida para a implementação do algoritmo foi Python. Amplamente utilizada no meio acadêmico, trata-se de uma linguagem de fácil compreensão e aprendizado, além de contar com diversas bibliotecas já implementadas com o objetivo de resolver problemas recorrentes.

As duas principais bibliotecas utilizadas neste projeto foram NumPy[4] e Matplotlib[5]. A primeira se trata de uma biblioteca para manipulação numérica e vetorizada de matrizes. É implementada na linguagem C, o que confere performance às operações numéricas, mantendo a facilidade de uso e versatilidade características da linguagem Python. Já a segunda, se trata de uma biblioteca para a criação de gráficos matemáticos, que será de suma importância na ilustração de forma clara dos resultados obtidos.

Os detalhes acerca da codificação escolhida para o material genético dos indivíduos, bem como os métodos utilizados nas etapas de seleção, recombinação e mutação foram propostos em 2006 por Roncaratti, Gargano e Silva[6], e serão abordados de forma breve no capítulo seguinte.

2 Metodologia

2.1 Codificação

Seja uma população de n indivíduos representada pelo conjunto

$$A = \{A_1\ ,\ \cdots\ ,\ A_k\ ,\ \cdots\ ,\ A_n\}\ , \tag{2.1}$$

onde cada indivíduo tem seu código genético representado por m cromossomos de l bits, de forma que, usando uma notação matricial,

$$A_k = \begin{bmatrix} a_{11}^{(k)} & \cdots & a_{1m}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{l1}^{(k)} & \cdots & a_{lm}^{(k)} \end{bmatrix}\ , \tag{2.2}$$

onde $a_{ij}^{(k)} \in \{0, 1\}$, sendo i, j e k inteiros tais que $i \in [0, l]$, $j \in [0, m]$ e $k \in [0, n]$.

Para um problema de otimização numérica de uma função $f : \mathcal{C} \subseteq \mathbb{R}^n \to \mathbb{R}$, deve existir um mapa injetivo entre $\mathbf{A}$ e o conjunto

$$X = \{X_1\ ,\ \cdots\ ,\ X_k\ ,\ \cdots\ ,\ X_n\}\ , \tag{2.3}$$

onde

$$X_k = \left(x_1^{(k)},\ \cdots\ ,\ x_j^{(k)},\ \cdots\ ,\ x_m^{(k)}\right) \tag{2.4}$$

são vetores de $\mathbb{R}^n$ e representam a posição do respectivo indivíduo no espaço de busca. Tal espaço é um subconjunto de $\mathcal{C}$ definido pelo produto cartesiano

$$\mathcal{S} = \bigtimes_{j=1}^{m} [x_j^{(min)}\ ,\ x_j^{(max)}]\ , \tag{2.5}$$

sendo $x_j^{(min)}$ e $x_j^{(max)}$, respectivamente, os valores mínimo e máximo para a coordenada $x_j^{(k)}$.

Como l bits são capazes de representar números naturais em $[0, 2^l)$, uma forma natural de mapear tal intervalo em $[x_j^{(min)}\ ,\ x_j^{(max)}]$ é

$$x_j^{(k)} = x_j^{(min)} + \frac{x_j^{(max)} - x_j^{(min)}}{2^l - 1} \sum_{i=1}^{l} a_{ij}^{(k)} 2^{i-1}\ , \tag{2.6}$$

cuja relação inversa é determinada por

$$\sum_{i=1}^{l} a_{ij}^{(k)} 2^{i-1} = \left\lfloor \frac{(x_j^{(k)} - x_j^{(min)})(2^l - 1)}{x_j^{(max)} - x_j^{(min)}} \right\rfloor\ . \tag{2.7}$$

2.2 Seleção

Para o processo de seleção é introduzida uma nova função, denominada função desempenho. Tal função é alguma função $g : \mathbb{R} \to \mathbb{R}$ a partir da qual se calcula a probabilidade de seleção do indivíduo k

$$P_k = \frac{g(f(X_k))}{\sum_{j=1}^{n} g(f(X_j))} \, . \tag{2.8}$$

Existe uma liberdade para a escolha da função desempenho, desde que se respeite a condição a função seja sempre positiva. A ideia é que se escolha uma função desempenho que selecione os melhores indivíduos, mas ainda mantenha uma boa variedade genética na próxima geração. Assim, em toda geração, em média, haverá indivíduos em todo o espaço de busca, evitando que a solução convirja de forma muito rápida[1].

$$a = \begin{cases} \frac{\mu(h-1)}{f_{max}-\mu} \, , & \text{se } f_{min} > \frac{h\mu - f_{max}}{h-1} \\ \frac{\mu}{\mu - f_{min}} \, , & \text{caso contrário} \end{cases} \tag{2.9}$$

$$b = \begin{cases} \frac{\mu(f_{max}-h\mu)}{f_{max}-\mu} \, , & \text{se } f_{min} > \frac{h\mu - f_{max}}{h-1} \\ -\frac{\mu f_{min}}{\mu - f_{min}} \, , & \text{caso contrário} \end{cases} \tag{2.10}$$

Nesse trabalho a função desempenho usada foi uma função linear $g(x) = ax + b$ sendo a e b definidos pelas equações 2.9 e 2.10, onde

$$f_{min} = \min_{X_k \in X} f(X_k) \, , \tag{2.11}$$

$$f_{max} = \max_{X_k \in X} f(X_k) \, , \tag{2.12}$$

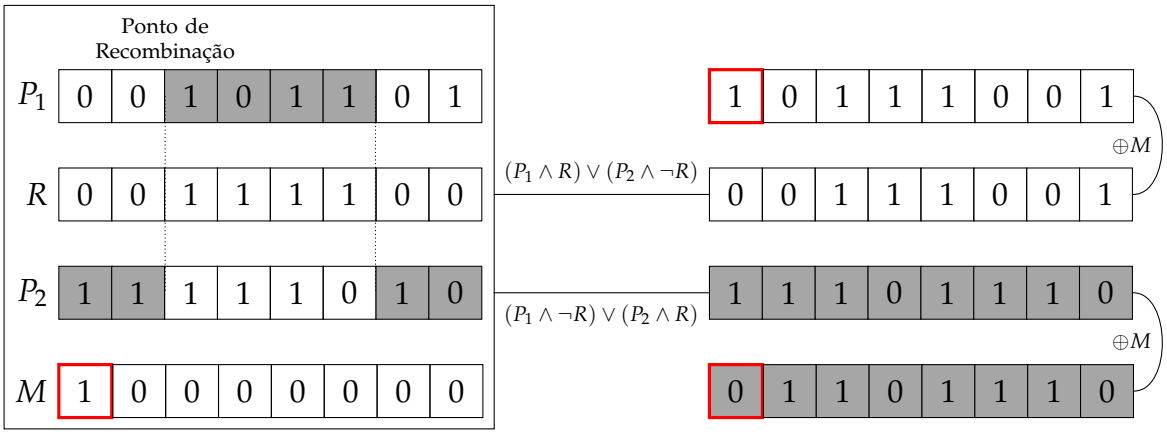
h é um parâmetro real maior que 1 e μ é a média do valor de f sobre a população, dada por

$$\mu = \sum_{k=1}^{n} \frac{f(X_k)}{n} \, . \tag{2.13}$$

A seleção do indivíduos que serão recombinados para formar a geração seguinte é feita por meio de uma roleta simples, permitindo ou não repetições. São feitos $n/2$ sorteios, já que para cada par de indivíduos recombinados, dois novos serão gerados[2]. Desse modo, a geração subsequente será constituída pelos indivíduos pais selecionados e os seus respectivos filhos, seguindo com n indivíduos.

[1] Retomando a função usada como exemplo no capítulo anterior, definida na equação 1.1, se escolhermos uma função de desempenho que privilegie de forma desproporcional pontos onde o valor da função é maior. Dependendo da distribuição inicial dos indivíduos, é possível que após algumas gerações, toda a população se concentre em $r = 1/2n$, que é um máximo local.

[2] Por esse motivo, n deve ser um múltiplo de 4.



2.3 Recombinação e Mutação

3 Resultados Preliminares

Escreva os resultados aqui.

4 Conclusão

Texto da conclusão.

5 Cronograma

Escreva aqui o cronograma.

Referências

1 Wikipedia contributors. *Knapsack problem — Wikipedia, The Free Encyclopedia*. 2021. <https://en.wikipedia.org/w/index.php?title=Knapsack_problem&oldid=1060217808>. [Online; accessed 20-April-2022].

2 HOLLAND, J. H. Genetic algorithms. *Scientific American*, JSTOR, v. 267, n. 1, p. 66–73, 1992. Disponível em: <http://papers.cumincad.org/data/works/att/7e68.content.pdf>.

3 Wikipedia contributors. *John Henry Holland — Wikipedia, The Free Encyclopedia*. 2021. <https://en.wikipedia.org/w/index.php?title=John_Henry_Holland&oldid=1041115860>. [Online; accessed 20-April-2022].

4 HARRIS, C. R. et al. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

5 HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

6 RONCARATTI, L. F.; GARGANO, R.; SILVA, G. M. e. A genetic algorithm to build diatomic potentials. *Journal of Molecular Structure: THEOCHEM*, Elsevier, v. 769, n. 1-3, p. 47–51, 2006.