Computação Natural

UNIDADE 04

Abordando bibliotecas para aplicação de algoritmos genéticos

Vários parâmetros são fundamentais para determinar a eficácia de um algoritmo genético, incluindo o tamanho da população, a probabilidade de cruzamento, a probabilidade de mutação e o número máximo de gerações. Por exemplo, definir um número limitado de gerações pode resultar em valores de *fitness* subótimos, pois o algoritmo pode não ter tempo suficiente para evoluir para soluções mais eficazes.

A biblioteca DEAP (FORTIN *et al*., 2012) oferece uma variedade de operadores que podem ser utilizados para configurar e aprimorar algoritmos genéticos. Alguns desses operadores são destacados a seguir. Vale lembrar que a documentação da biblioteca, disponível no *link*, fornece um leque ainda mais amplo de opções. Os operadores implementados no DEAP podem ser visualizados na figura 1.

Inicialização	Cruzamento	Mutação	Seleção	Migração
initRepeat()	cxOnePoint()	mutGaussian()	selTournament()	migRing()
initIterate()	cxTwoPoint()	mutShuffleIndexes()	selRoulette()	
initCycle()	cxUniform()	mutFlipBit()	selNSGA2()	
	cxPartialyMatched()	mutPolynomialBounded()	selNSGA3()	
	cxUniformPartialyMatched()	mutUniformInt()	selSPEA2()	
	cxOrdered()	mutESLogNormal()	selRandom()	
	cxBlend()		selBest()	
	cxESBlend()		selWorst()	
	cxESTwoPoint()		selTournamentDCD()	
	cxSimulatedBinary()		selDoubleTournament()	
	cxSimulatedBinaryBounded()		selStochasticUniversalSampling()	
	cxMessyOnePoint()		selLexicase()	
			selEpsilonLexicase()	
			selAutomaticEpsilonLexicase()	

Figura 1: Operadores implementados no DEAP.Fonte: https://deap.readthedocs.io/en/master/api/tools.html

Adicionalmente, a biblioteca DEAP disponibiliza operadores exclusivos para a programação genética, como ilustrado na figura 2. Para explorar em detalhes qualquer operador que desperte seu interesse, recomendamos visitar o *link* fornecido anteriormente. Lá, você poderá clicar especificamente no operador que pretende utilizar. A documentação da DEAP é bastante abrangente e inclui uma variedade de exemplos práticos, facilitando a compreensão e a aplicação desses operadores em seus projetos.

Inicialização	Cruzamento	Mutação	Controle de *bloat*
genFull()	cxOnePoint()	mutShrink()	staticLimit()
genGrow()	cxOnePointLeafBiased()	mutUniform()	selDoubleTournament()
genHalfAndHalf()	cxSemantic()	mutNodeReplacement()	
		mutEphemeral()	
		mutInsert()	
		mutSemantic()	

Figura 2: Operadores específicos de programação genética no DEAP.
Fonte: https://deap.readthedocs.io/en/master/api/tools.html

A documentação do DEAP detalha uma diversidade de operadores para algoritmos genéticos e permite combinações desses operadores para personalizar o comportamento do algoritmo. Por exemplo, a combinação de **seleção por roleta**, **cruzamento por ponto único** e **mutação por troca de *bit*** é uma das possíveis configurações. Esses operadores são indicados na documentação como **selRoulette()**, **cxOnePoint()**, e **mutFlipBit()**, respectivamente. Na prática, ao implementar esses operadores no DEAP, adotamos uma nomenclatura consistente para facilitar a identificação: operadores de seleção são registrados como ***select***, operadores de

cruzamento como ***mate***, e operadores de mutação como ***mutate***. Isso cria um padrão claro e facilita a leitura e a manutenção do código, conforme exemplificado na figura 3, que mostra como esses operadores são aplicados juntos. Lembramos que os fundamentos desses três tipos de operadores foram explorados na semana anterior, proporcionando uma base sólida para a compreensão e aplicação prática dessas técnicas.

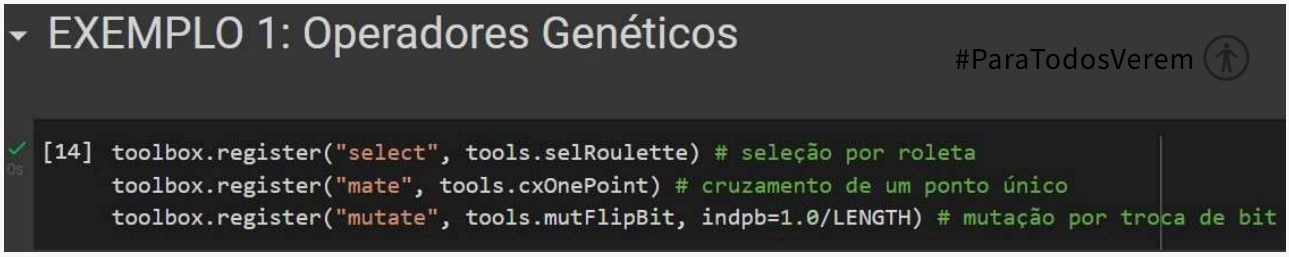

Figura 3: Primeiro exemplo com operadores genéticos. Fonte: O autor (2022).

Aplicando os conhecimentos adquiridos na aula anterior, para realizar uma **seleção por torneio**, um **cruzamento por dois pontos** e uma **mutação de troca de *bit*** utilizando a biblioteca DEAP, o trecho de código correspondente, ilustrado na figura 4, empregaria os seguintes operadores: **selTournament()** para a seleção, **cxTwoPoint()** para o cruzamento e **mutFlipBit()** para a mutação.

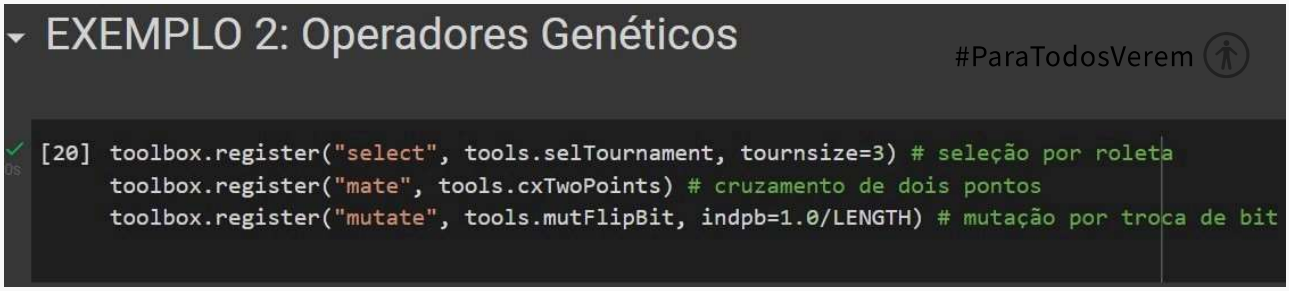

Figura 4: Segundo exemplo com operadores genéticos. Fonte: O autor (2022).

Se a intenção fosse implementar uma **seleção aleatória**, um **cruzamento uniforme** e uma **mutação por troca de bit** utilizando a biblioteca DEAP, o código correspondente, que seria representado na Figura 5, utilizaria **selRandom()** para a seleção, **cxUniform()** para o cruzamento e **mutFlipBit()** para a mutação. Entre esses, o conceito de seleção aleatória é o único que não foi abordado na aula anterior.

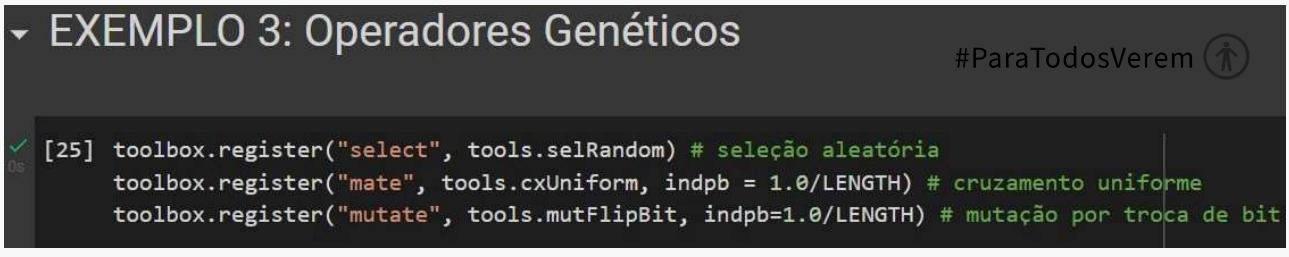

Figura 5: Terceiro exemplo com operadores genéticos. Fonte: O autor (2022).

Nesta videoaula, abordaremos a aplicação de algoritmos genéticos para solucionar o clássico problema de otimização conhecido como *knapsack* ou problema da mochila. Esse desafio consiste em selecionar um conjunto de itens, cada um com seu respectivo peso e valor, para colocar dentro de uma mochila de capacidade limitada, de forma a maximizar o valor total sem exceder o peso máximo permitido. O objetivo é encontrar a combinação ideal de itens que otimize o valor acumulado dentro das restrições de peso, exemplificando um problema de otimização combinatória que os algoritmos genéticos estão particularmente bem equipados para resolver. Durante a aula, exploraremos como esses algoritmos podem ser configurados e utilizados para buscar a melhor solução possível para o problema da mochila.

IMPORTANTE

Olá, estudante, todos os códigos que vamos ver nas videoaulas estão disponíveis no github (https://github.com/yohangumiel/Aulas-PUC-PR/tree/main/Computacao-Natural). Para esta aula, utilize este código.

Conclusão da Unidade e Referências

Concluímos a quarta semana da nossa disciplina, na qual continuamos a aprofundar nosso entendimento sobre algoritmos genéticos. Durante esta semana, nos familiarizamos com a biblioteca DEAP, uma ferramenta valiosa para a implementação desses algoritmos, que oferece uma gama de operadores para inicialização, cruzamento, mutação, seleção e migração.

Exploramos também uma série de operadores e como as suas combinações podem ser empregadas na criação de algoritmos eficientes. Com esse conhecimento, você agora tem a capacidade de ajustar o código e experimentar com sua própria seleção de operadores, personalizando a forma como seu algoritmo genético opera.

No conteúdo audiovisual desta semana, aplicamos esses algoritmos ao resolver o problema da mochila, um teste clássico de otimização. Por meio da prática, aplicamos os conceitos teóricos em um contexto prático, demonstrando o uso efetivo de

operadores em um cenário de otimização do mundo real. Isso nos permitiu ver em ação a teoria que estudamos, consolidando o aprendizado e preparando-nos para desafios futuros.

FORTIN, F. A. *et al*. DEAP: evolutionary algorithms made easy. **The Journal of Machine Learning Research**, [s.l.], v. 13, n. 1, p. 2171-2175, 2012.