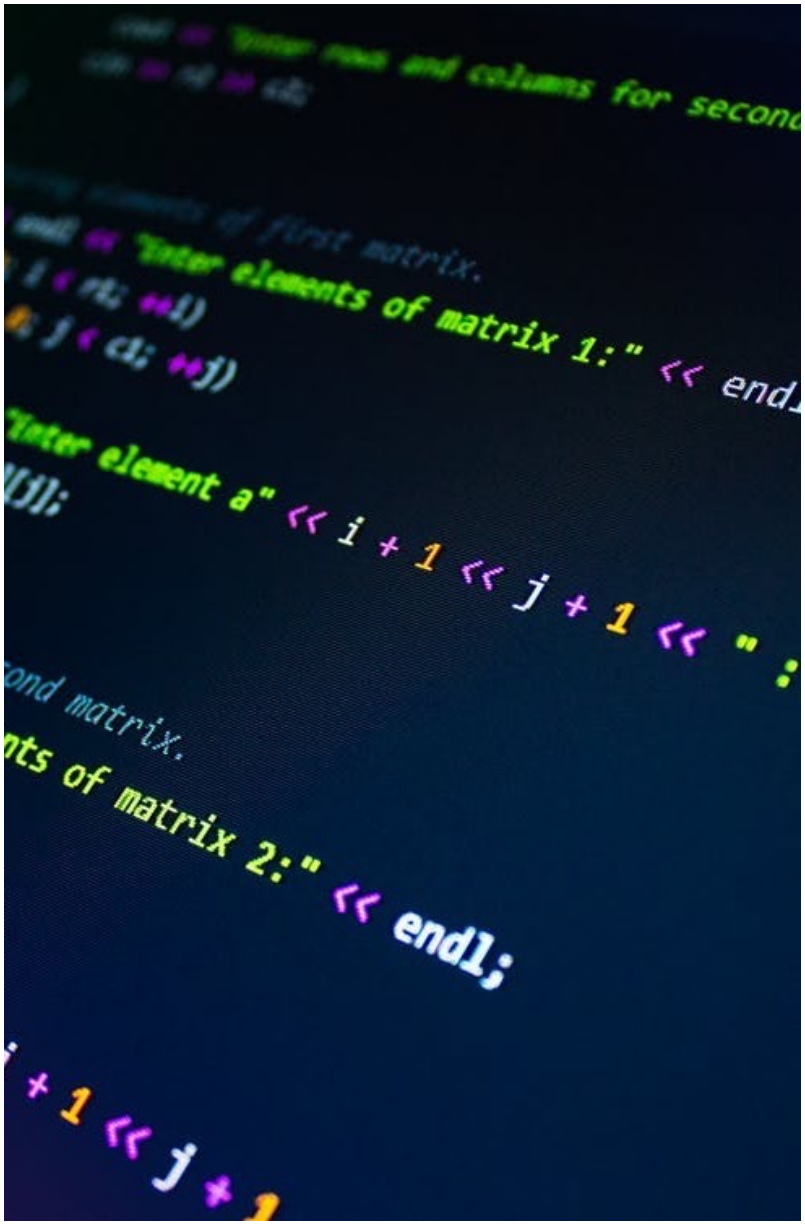
Insper

Supercomputação



Aula - 06

- Exploration / Exploitation
- Resolução da mochila por algoritmos genéticos
- Algoritmos aleatorizados

Heurísticas

- Vimos que heurísticas são “truques” usados para resolver um problema rapidamente

- Uma boa heurística consegue obter resultados aproximados ou ganhos de curto prazo, porém não garante resultados ótimos, nem resultados bons em todas as situações!

Heurísticas - Limitações

- E se a solução gerada não for boa? Conseguimos “tentar” de novo e gerar outras parecidas?

- Será que é possível melhorar a solução gerada? Como?

Exploration vs. Exploitation

Exploration

Exploitation

Exploration:

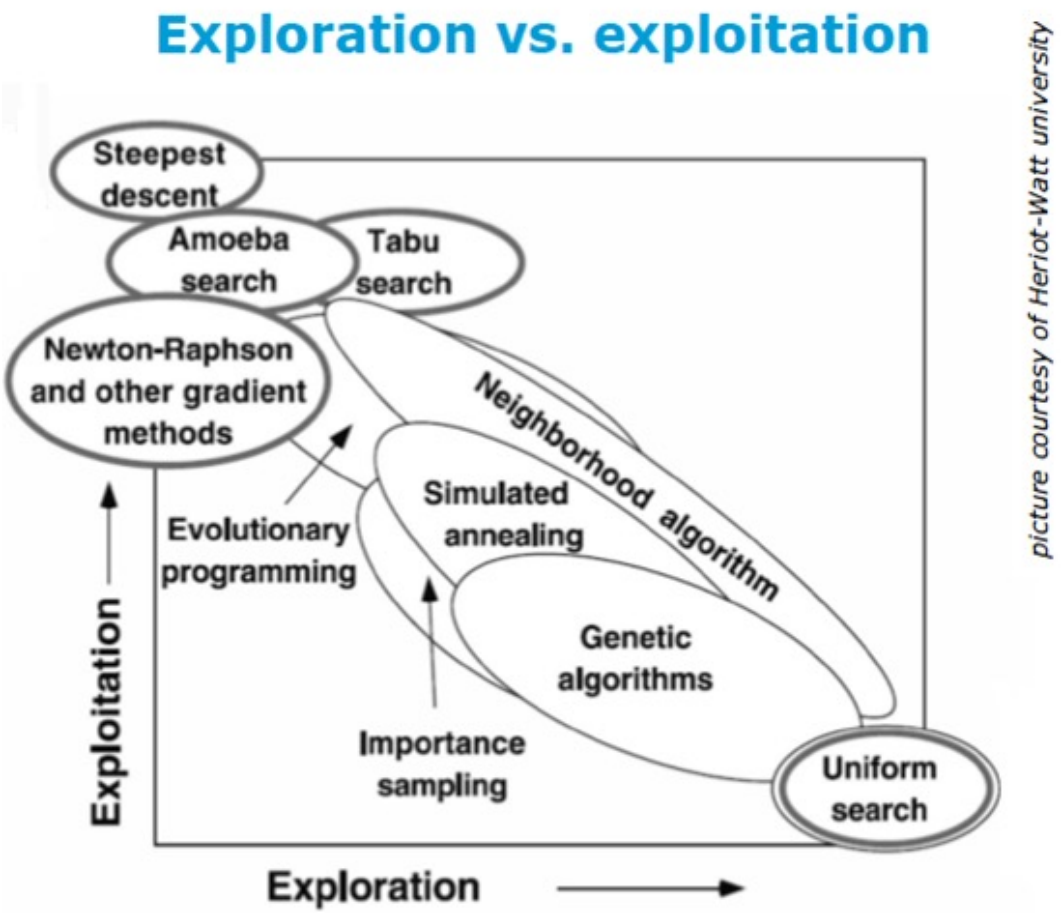
- decisão não localmente ótima feita "de propósito"
- visa adicionar variabilidade nas soluções geradas

Exploitation:

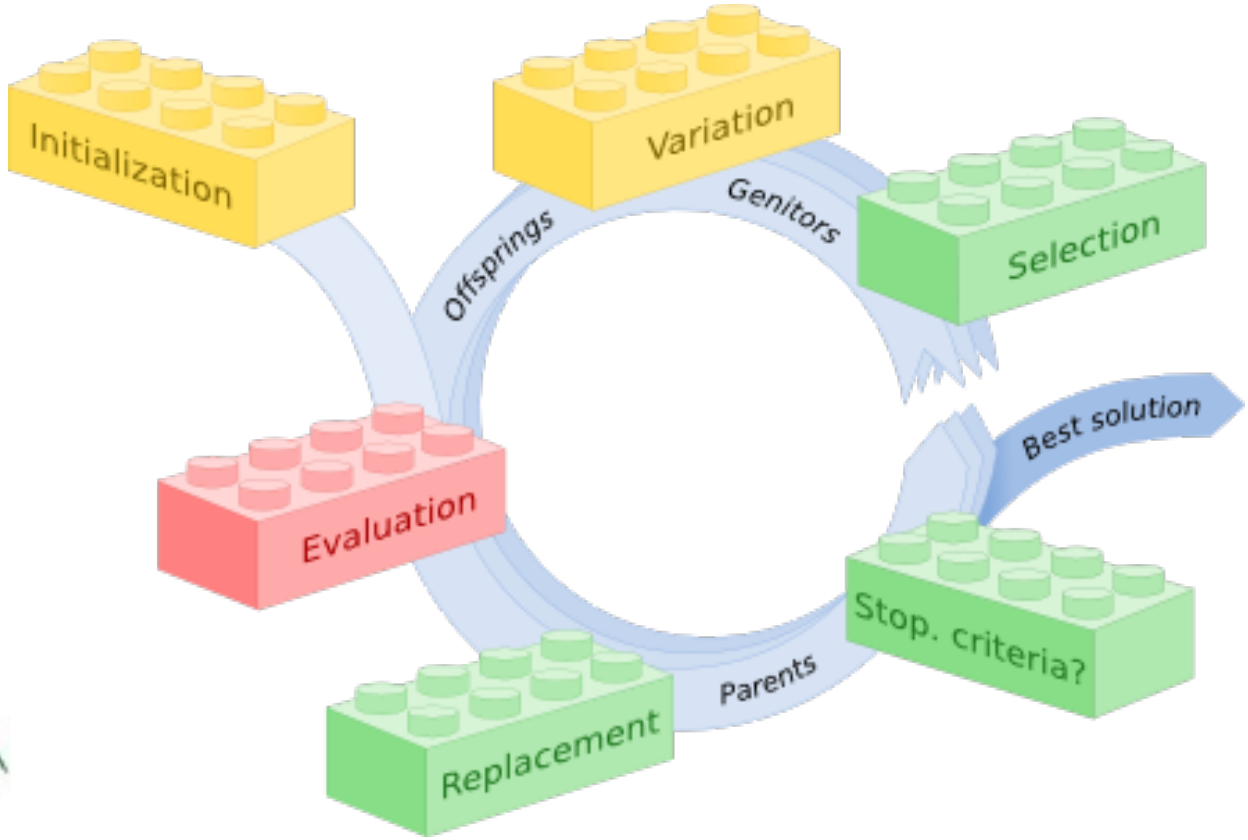
- explorar alguma propriedade do problema
- pode ser uma intuição que leve a bons resultados em curto prazo

Exploration vs. Exploitation

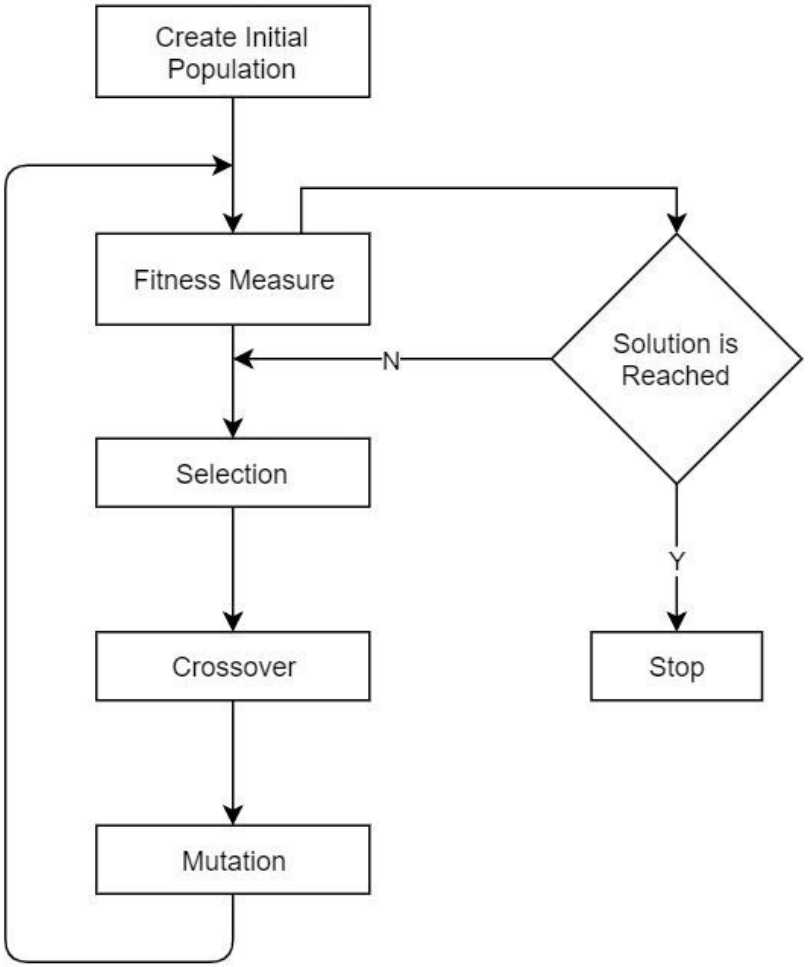
- Possíveis técnicas.
  Você reconhece alguma?



Algoritmos genéticos



Algoritmo Genético para a Mochila Binária



Exploration vs Exploitation em Algoritmos Genéticos

- Estratégia de Reprodução (cross-over)
- Taxa de mutação
- Estratégia de elitismo

Possíveis itens:

Item	Peso	Valor
1	12	32
2	2	29
3	1	29
4	5	4
5	6	2
6	4	20
7	7	34
8	1	39
9	1	48
10	2	44

Capacidade da mochila: 35

População Inicial (n=8)

```
Tamanho da população = (8, 10)
População inicial:
 [[1 0 1 1 0 0 0 1 1 1]
 [1 0 0 0 0 1 1 0 1 1]
 [0 0 0 0 1 0 1 1 1 1]
 [0 0 0 0 0 1 0 1 0 0]
 [1 1 1 1 0 0 1 0 1 0]
 [1 0 0 0 1 1 1 1 0 0]
 [1 1 1 0 0 0 1 0 0 0]
 [0 1 1 0 1 1 0 1 0 1]]
```

Fitness

```
array([[196],
       [178],
       [167],
       [ 59],
       [176],
       [127],
       [124],
       [163]])
```

Seleção

```
[1 0 1 1 0 0 0 1 1 1]    [1 0 0 0 0 1 1 0 1 1]
[0 0 0 0 1 0 1 1 1 1]    [1 1 1 1 0 0 1 0 1 0]
```

Cross-Over

```
[1 0 1 1 0 0 0 1 1 1]    [1 0 1 1 0 1 1 0 1 1]
[1 0 0 0 0 1 1 0 1 1]    [1 0 0 0 0 0 0 1 1 1]
```

Mutação

```
[1 0 0 0 0 0 0 1 1 1]    [1 0 0 0 1 0 0 1 1 1]
```

Repetir por N gerações

$$fitness = \sum_{i=1}^{n} c_i v_i; if \sum_{i=1}^{n} c_i w_i \leq kw$$

$$fitness = 0; otherwise$$

Mochila Binária por Algoritmos Genéticos

Implementação em Python

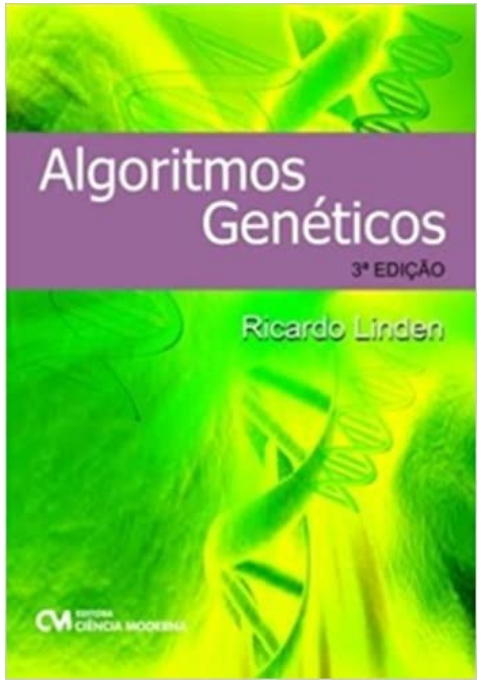
- Clique aqui para conhecer a implementação em Python para a mochila binária por meio de algoritmos genéticos

- Atenção:

- 0,5 ponto extra na nota final se entregar a implementação em C++

Para saber mais ...

- https://www.algoritmosgeneticos.com.br



Extra ... No mundo real ...

Automated Antenna Design with Evolutionary Algorithms

Gregory S. Hornby* and Al Globus
University of California Santa Cruz, Mailtop 269-3, NASA Ames Research Center, Moffett Field, CA
Derek S. Linden
JEM Engineering, 8683 Cherry Lane, Laurel, Maryland 20707
Jason D. Lohn
NASA Ames Research Center, Mail Stop 269-1, Moffett Field, CA 94035

G. S. Hornby, J. D. Lohn, and D. S. Linden

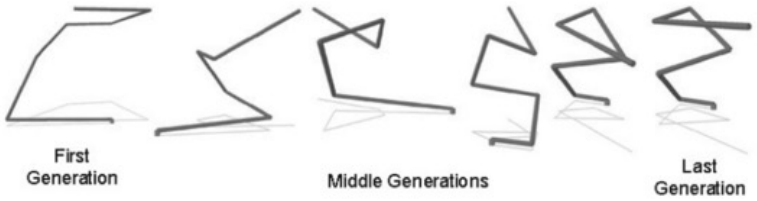


Figure 10: Sequence of evolved antennas leading up to antenna ST5-33.142.7.

De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?

De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?
    - Alternar heurísticas **de vez em quando**
    - **De vez em quando faço** uma escolha qualquer
    - Inverto a heurística **de vez em quando**

De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?
  - Alternar heurísticas **de vez em quando**
  - **De vez em quando faço** uma escolha qualquer
  - Inverto a heurística **de vez em quando**

Aleatoriedade

Exploration

- Exploration requer a capacidade de criar um programa que executa de maneira diferente a cada execução

- Precisamos:
    - 1. Criar uma fonte de aleatoriedade;
    - 2. Uma maneira de gerar sequências de números aleatórios

Números aleatórios

- Um gerador de números aleatórios é impossível de ser criado usando um computador

- 1. É impossível predizer qual será o próximo número aleatório “de verdade”;

- 2. Um computador executa uma sequência de comandos conhecidos, baseando-se em dados guardados na memória. A execução é, portanto, determinística

Números (pseudo-)aleatórios

- Gerador de números pseudo-aleatório (pRNG): algoritmo determinístico que gera sequências de números que parecem aleatórias

- 1. Determinístício: produz sempre a mesma sequência.

- 2. Sequências que parecem aleatórias: não conseguiríamos distinguir uma sequência gerada por um pRNG e uma sequência aleatória de verdade

Gerando números aleatórios

- Sorteio de números aleatórios é dado por 2 elementos:

- 1. Gerador: produz bits aleatórios a partir de um parâmetro seed. Cada seed gera uma sequência diferente de bits

- 2. Distribuição de probabilidade: gera sequência de números a partir de um conjunto de números

Entendendo (em Python)

Random Generators

```
In [1]: import numpy as np

In [2]: np.random.rand()
Out[2]: 0.9535543896720104

In [3]: np.random.seed(123)

In [4]: np.random.rand()
Out[4]: 0.6964691855978616
In [5]: np.random.rand()
Out[5]: 0.28613933495037946

In [6]: np.random.seed(123)

In [7]: np.random.rand()
Out[7]: 0.696469185597861
In [8]: np.random.rand()
Out[8]: 0.28613933495037946
```

Pseudo-random numbers
Mathematical formula
Starting from a seed

Same seed: same random numbers!
Ensures "reproducibility"

numpy.random.rand

`random.rand(d0, d1, ..., dn)`

Random values in a given shape.

> **Note**
>
> This is a convenience function for users porting code from Matlab, and wraps **`random_sample`**. That function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like **`numpy.zeros`** and **`numpy.ones`**.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Distributions

beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution.
exponential([scale, size])	Draw samples from an exponential distribution.
f(dfnum, dfden[, size])	Draw samples from an F distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
geometric(p[, size])	Draw samples from the geometric distribution.
gumbel([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.

Atividade prática 1

Resolvendo a mochila binária por meio de heurísticas com aleatoriedade

Atividade prática 2

E se tudo fosse aleatório?

Implemente uma solução completamente aleatória para a mochila binária

Discussão

- Adicionar aleatoriedade melhorou os resultados?
- Qual a qualidade das soluções aleatórias?

Obrigado