# Inteligência Computacional

Luís A. Alexandre

UBI

Ano lectivo 2019-20



## Conteúdo







# Algoritmos genéticos





## Introdução

- Os algoritmos genéticos (AGs) foram inventados por J. Holland durante a década de 1960 (ver Handbook of Evolutionary Computation, p.A2.3:1).
- Permitem resolver problemas de otimização codificando soluções em cromossomas, criando uma população inicial destes cromossomas e evoluindo essa população durante várias gerações até que no fim a solução é dada pelo cromossoma com maior aptidão.
- Os AGs como ferramenta de otimização são usados em múltiplas áreas, entre as quais: CAD, controlo, química, física, economia, logística, robótica, redes neuronais, processamento de sinal e imagem.
- Nesta aula ficaremos a perceber como adaptar o contexto geral visto na aula anterior às especificidades dos AGs.





# Representação dos cromossomas





## Variáveis com valores binários

- A representação original para os cromossomas usada nos AGs era a de uma string binária.
- No entanto é possível termos AGs com genes com valores reais, p.ex..
- Mas se pretendermos usar a representação binária quando os dados não são binários, é necessário encontrar um modo de converter a informação.

## Variáveis com valores nominais

- A informação a usar nos genes pode ser de tipo nominal. Ex.: profissão: {estudante, professor, advogado, outra}.
- O objetivo é representar isto usando uma string binária.
- Pode ser codificada da seguinte forma: cada um dos valores possíveis da variável profissão pode ser codificado como uma string binária de dimensão 2.
- Assim, estudante = 00, professor = 01, advogado =10 e outra=11.
- No caso geral usamos string de dimensão $D$ quando o número de valores possíveis a codificar é $2^D$.





## Variáveis com valores reais

- Quando as variáveis em causa podem assumir valores reais, temos de fazer duas coisas para as representar como strings binárias: restringir o seu domínio e encontrar uma boa representação binária.
- Por exemplo, se tivermos uma variável $z$ que varie entre $[z_{min}, z_{max}]$ e a quisermos representar usando 16 bits, podemos usar a seguinte fórmula para a conversão de valores:

$$z_{bin} = (2^{16} - 1)\frac{z - z_{min}}{z_{max} - z_{min}}$$

- Mais concretamente, se quiséssemos codificar valores de altura de pessoas que partimos do princípio estariam entre $[40, 250]$ cm, usando 16 bits, então a altura $z = 160$cm ficaria em binário nesta representação como 1001001001001001(= 37449).





## Problemas com a codificação binária

- Embora seja comum o seu uso, a codificação binária tem um problema importante: dois números consecutivos podem diferir em muitos bits.
- Este problema é importante pois uma pequena distância nas variáveis deveria implicar uma pequena alteração na aptidão dos respetivos cromossomas.
- A distância normalmente usada para comparar valores em binário é a distância de Hamming que se reduz a uma contagem do número de bits que diferem entre os dois números.
- Exemplo: a distância de Hamming entre os números 3 e 4 em binário é dada por $d_H(011, 100) = 3$.
- Solução: em vez de representar as strings em binário (simples) podemos usar o código Gray.





## Código Gray

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Binário | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Gray | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

- O código Gray goza da propriedade de que números consecutivos diferem apenas de 1 relativamente à distância de Hamming.
- A representação de um número binário pode ser convertida para código Gray usando:

$$g_i = \begin{cases} b_1 & \text{se} \quad i = 1 \\ b_{i-1}\bar{b}_i + \bar{b}_{i-1}b_i & \text{caso contrário} \end{cases}$$

onde $b_i$ representa o bit $i$ do número binário $b_1 b_2 \cdots b_k$ onde $b_1$ é o bit mais significativo. $\bar{b}_i$ representa a negação do bit $b_i$, o $+$ representa o OU lógico e a multiplicação o E lógico.





# Cross-over





## Algoritmo para o cross-over

- O objetivo do cross-over é produzir descendentes a partir de pais selecionados com algum dos operadores de seleção referidos na aula anterior.
- Vejamos um algoritmo para o cross-over entre dois indivíduos, $C_{i_1}$ e $C_{i_2}$:
  1. Criar 2 novos cromossomas inicializados com o material genético dos pais: $\alpha = C_{i_1}$ e $\beta = C_{i_2}$
  2. Achar a máscara $m$.
  3. Para cada gene $j$ do cromossoma, se $m_j = 1$, trocar material genético entre os pais:
     - 3.1 $\alpha_j = C_{i_2,j}$
     - 3.2 $\beta_j = C_{i_1,j}$
  4. Devolver os descendentes $\alpha$ e $\beta$.

## Máscara

- A máscara referida no algoritmo anterior especifica quais os genes que serão alvo de troca durante o processo de cross-over.
- Existem várias formas de calcular esta máscara:
  - uniforme
  - um ponto
  - dois pontos
  - aritmético

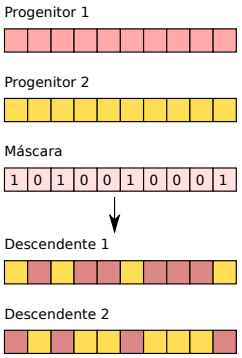




## Cross-over uniforme

- Neste caso, a máscara é criada de forma aleatória.
- Um bit a 1 significa que o alelo deve ser trocado, e se o bit estiver a zero, o alelo não é trocado entre os progenitores.
- Algoritmo ($N$ é n. de genes):
  1. $m_i = 0,\ i = 1, \ldots, N$
  2. Para cada gene $i$
     - 2.1 Obter $\xi \sim U(0,1)$
     - 2.2 Se $(\xi < p)$ então $m_i = 1$

  onde $p$ é a probabilidade de cross-over em cada gene.

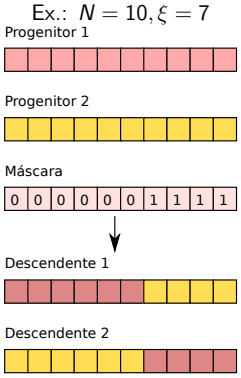







## Cross-over num ponto

- A máscara é criada escolhendo 1 ponto no cromossoma de forma aleatória: todos os alelos a partir desse ponto (inclusive) são trocados.
- Algoritmo ($N$ é n. de genes):
  1. $m_i = 0,\ i = 1, \ldots, N$
  2. Obter $\xi \sim U(1, N)$
  3. Para cada $i \geq \xi, \ldots, N$ fazer $m_i = 1$

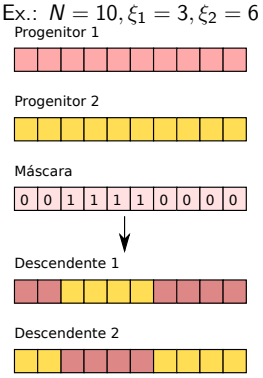







## Cross-over em dois pontos

- A máscara é criada escolhendo 2 pontos no cromossoma de forma aleatória: todos os alelos entre esses pontos (inclusive) são trocados.
- Algoritmo ($N$ é n. de genes):
  1. $m_i = 0,\ i = 1, \ldots, N$
  2. Obter $\xi_1, \xi_2 \sim U(1, N), \xi_1 < \xi_2$
  3. Para cada $i \in \{\xi_1, \ldots, \xi_2\}$ fazer $m_i = 1$








## Cross-over aritmético

- No caso em que os genes têm valores reais, podemos usar o cross-over aritmético.
- Este cross-over consiste em gerar os descendentes $\alpha_i$ e $\beta_i$ de dois progenitores $C_{i_1}$ e $C_{i_2}$ através de

$$\alpha_{i,j} = \xi_1 C_{i_1,j} + (1 - \xi_1) C_{i_2,j}$$

$$\beta_{i,j} = (1 - \xi_2) C_{i_1,j} + \xi_2 C_{i_2,j}$$

  onde $\xi_1, \xi_2 \sim U(0,1)$.
- Exemplo para cromossomas só com um gene: se $C_{i_1,1} = 74.20$ e $C_{i_2,1} = 71.30$, $\xi_1 = 0.22$ e $\xi_2 = 0.51$ então obtemos $\alpha_{i,1} = 71.94$ e $\beta_{i,1} = 72.72$.





# Mutação

## Mutação

- O objetivo da mutação é introduzir novo material genético num indivíduo de forma aleatória.
- O seu papel na procura do ótimo é garantir que se encontram acessíveis todos os possíveis valores dos alelos.
- A probabilidade de ocorrência de mutação num gene é chamada de taxa de mutação, $p_m$.
- Deve ser usado um valor baixo para $p_m$ de forma a não distorcer as boas soluções entretanto encontradas.
- No entanto, outra abordagem que provou ser positiva é a de inicializar $p_m$ com valores relativamente elevados e fazê-la decrescer de forma exponencial com as gerações para que: 1) inicialmente se pesquise numa grande parte do espaço e 2) não existam grandes perturbações nos cromossomas, conforme os indivíduos vão convergindo para a solução ótima.





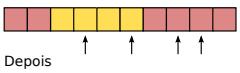
## Mutação aleatória

- Quando as variáveis a codificar nos cromossomas têm valores binários, podemos usar a mutação aleatória.
- Se $C_j$ for um cromossoma selecionado para mutação, temos de seguida o algoritmo para efetuar a mutação aleatória.
- Algoritmo ($N$ é n. de genes):
  1. Para cada $i = 1, \ldots, N$ fazer:
     1.1 Obter $\xi \sim U(0,1)$
     1.2 Se $\xi \leq p_m$ fazer $C_{j,i} = \bar{C}_{j,i}$

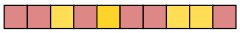
  onde $\bar{C}_{j,i}$ representa a negação de $C_{j,i}$.

Ex.: $N = 10$, $i : \xi \leq p_m, \in \{4, 6, 8, 9\}$

Antes



Depois





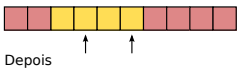


## Mutação inorder

- Outra possibilidade para variáveis com valores binários é a mutação inorder: escolher 2 posições aleatoriamente no cromossoma e apenas os bits entre elas poderão sofrer mutação.
- Se $C_j$ for um cromossoma selecionado para mutação, temos de seguida o algoritmo para efetuar a mutação inorder: ($N$ é n. de genes):
  1. Obter $\xi_1, \xi_2 \sim U(1, N), \xi_1 \leq \xi_2$
  2. Para cada $i = \xi_1, \ldots, \xi_2$ fazer:
     2.1 Obter $\xi \sim U(0,1)$
     2.2 Se $\xi \leq p_m$ fazer $C_{j,i} = \bar{C}_{j,i}$

Ex.: $N = 10, \xi_1 = 4, \xi_2 = 6$

Antes



Depois





## Mutação para variáveis não binárias

- Quando as variáveis têm valores nominais (ex.: profissão), os operadores de mutação vistos atrás devem ser modificados de forma a que os $D$ bits que representam um desses valores sejam aleatoriamente substituídos por outros $D$ bits que representem um valor válido diferente.
- Se as variáveis tiverem valores reais, a mutação ocorre através da adição de um valor aleatório (tipicamente obtido duma distribuição Gaussiana) aos alelos.
- Algoritmo para mutação de alelos reais:
  1. Para cada gene real $j$ fazer:
     1.1 Obter $\xi \sim U(0,1)$
     1.2 Obter um valor $\eta \sim N(0, \sigma^2)$
     1.3 Se $\xi \leq p_m$ fazer $C_{i,j} += \eta$.
- O valor da variância $\sigma^2$ é normalmente inversamente proporcional à aptidão do indivíduo de forma que a indivíduos mais aptos correspondam menores variâncias.





# Programação genética





## Introdução

- A programação genética (PG) é uma especialização dos AGs.
- A diferença principal entre a PG e os AGs está no tipo de representação que é feito dos indivíduos: enquanto que os AGs usam uma string a PG usa árvores.
- Na PG cada indivíduo é um programa executável.
- O algoritmo genérico de um AE pode ser usado para a PG, com as modificações que iremos discutir.

# Representação dos cromossomas





## Gramática

- Cada cromossoma representa um programa sob a forma duma árvore.
- Para podermos representar os programas como árvores temos de definir uma gramática que reflita o problema a resolver.
- Temos de definir dois conjuntos:
  - terminal: contém todas as variáveis e constantes.
  - funções: todas as funções aplicáveis ao conjunto terminal.
- As funções podem ser quaisquer funções matemáticas: exp, sin, XOR, AND, $+$, $/$, etc.
- Estruturas de decisão do tipo SE ... ENTÃO ... SENÃO também podem ser incluídas no conjunto das funções.
- As folhas da árvore são constituídas por elementos do conjunto terminal enquanto que os elementos não folha pertencem ao conjunto de funções.



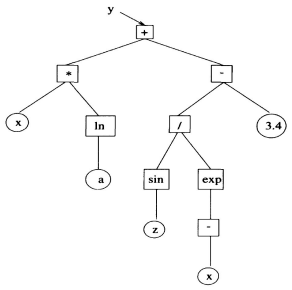


## Exemplo

- Vejamos como exemplo o seguinte programa:

  $y = x*\ln(a)+\sin(z)/\exp(-x)-3.4$

- O conjunto terminal é $\{a, x, z, 3.4\}$ onde $a, x, z \in \mathbb{R}$.
- O conjunto de funções é $\{+, -, *, /, \ln(), \sin(), \exp()\}$.
- A solução ótima é a da figura ao lado.




(Figura de Engelbrecht, p.148)





## Representação

- O espaço de pesquisa na PG é o formado por todos os programas que podem ser gerados a partir da gramática definida para o problema.
- O objetivo da PG é procurar qual o cromossoma que se aproxima mais da função desejada.
- As árvores podem ter um tamanho fixo ou variável:
  - fixo: todas têm a mesma profundidade;
  - variável: apenas é definida uma profundidade máxima. Esta é a abordagem mais comum.
- Existem abordagens em que a profundidade máxima vai aumentado com o número de gerações.





## População inicial

- A população inicial é gerada aleatoriamente, mas obedecendo às restrições impostas pela gramática definida e à profundidade máxima.
- Para cada indivíduo, a raiz da árvore é escolhida do conjunto de funções.
- O número de filhos da raiz e restantes nodos não terminais é dado pelo número de parâmetros que a função escolhida requer.
- Para cada nodo não raiz, é escolhido um elemento de um dos conjuntos (terminal ou de funções), sendo que após um nodo receber um elemento do conjunto terminal, deixa de estar disponível para expansão.





## População inicial

- Exercício: simule a criação de um cromossoma da população inicial para o problema visto atrás. O número total de elementos é 11. Vá sorteando números entre 1 e 11 e escolha os elementos de acordo com a posição que ocupam dentro dos conjuntos, começando pelo dos símbolos terminais. Preencha a árvore por ordem da pesquisa primeiro em profundidade.
- Ex.: o número 3 refere-se ao símbolo terminal $z$ ao passo que o 6 se refere ao símbolo funcional $-$.
- Experimente usando a seguinte sequência de números: 7,8,1,8,2,1,4. Deve obter a função:

$$y = \frac{a}{x/a} * 3.4 = 3.4 * a^2/x$$

# Função de aptidão





## Avaliação da aptidão

- A função de aptidão é dependente do problema em concreto.
- No entanto a ideia é avaliar o programa (indivíduo) o que requer que seja corrido o programa com diferentes parâmetros.
- A média dos resultados da aptidão do indivíduo em cada execução é normalmente usada para medir a aptidão.
- Voltando ao exemplo anterior, consideremos que a verdadeira função era desconhecida e que se possuía apenas um conjunto de dados com três características (que correspondem a valores para as variáveis $a, x, z$) e a respetiva saída desejada ($y$). Além deste conjunto de dados também se possui os conjuntos terminal e de funções.
- A aptidão de um indivíduo pode então ser avaliada executando o programa em todos os padrões do conjunto de dados e medindo o erro quadrático médio nesse conjunto.





## Penalização

- A função de aptidão pode incorporar termos que permitam penalizar determinadas propriedades não desejadas nos indivíduos.
- Por exemplo, em vez de definirmos a priori uma profundidade máxima para as árvores, podemos introduzir um termo na função de aptidão que penalize a profundidade.
- Outra possibilidade seria penalizar árvores cujos elementos tivessem elevado grau (número de filhos).





# Cross-over





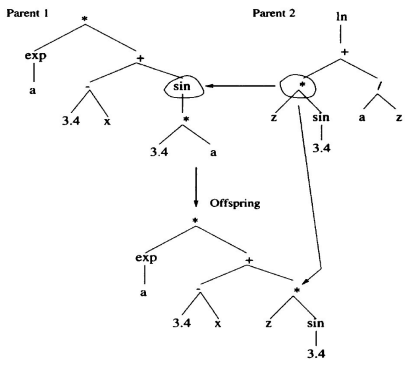
## Cross-over

- Qualquer dos operadores de seleção vistos anteriormente pode ser usado para escolher os dois pais que irão produzir a descendência.
- No caso de se pretender apenas um filho, seleciona-se aleatoriamente um nodo em cada progenitor e a sub-árvore dum dos progenitores substitui a do outro a partir dos nodos selecionados.
- No caso de se querer dois filhos, seleciona-se aleatoriamente um nodo em cada progenitor e as respetivas sub-árvores são trocadas.



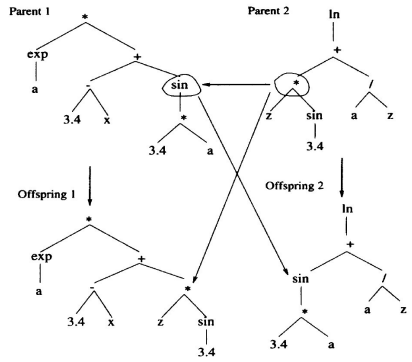


## Cross-over: 1 filho




(Figura de Engelbrecht, p.150)

## Cross-over: 2 filhos




(Figura de Engelbrecht, p.150)





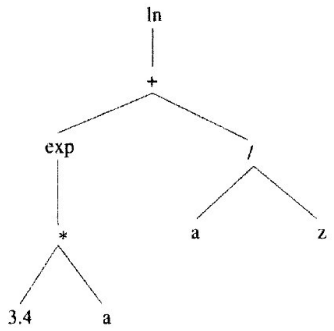
# Mutação





## Mutação

- Iremos ver vários operadores de mutação nas páginas seguintes.
- Todos os exemplos são baseados na seguinte árvore:



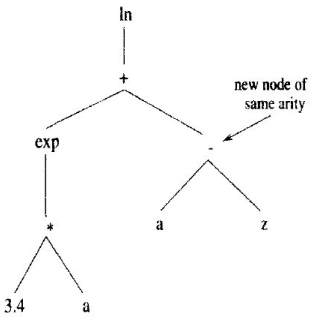

(Figura de Engelbrecht, p.153)





## Mutação de nodos funcionais

- Mutação de nodos funcionais: um nodo não terminal é selecionado e substituído por outro da mesma aridade também selecionado aleatoriamente dentro do conjunto de funções.



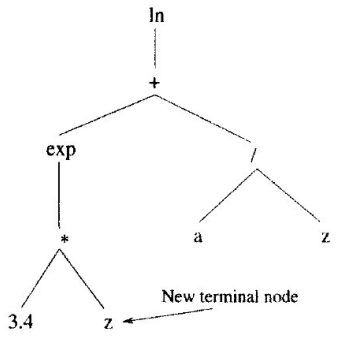

(Figura de Engelbrecht, p.153)





## Mutação de nodos terminais

- Mutação de nodos folha: processo idêntico ao anterior mas agora os nodos são escolhidos do conjunto terminal.



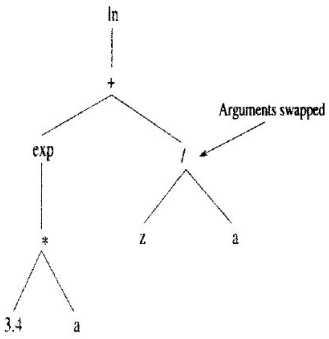

(Figura de Engelbrecht, p.153)





## Mutação por troca

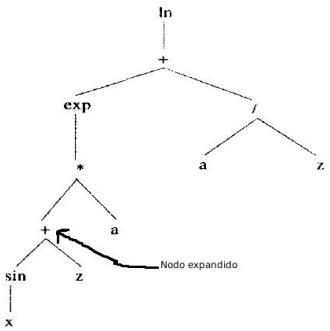
- Mutação por troca: um nodo funcional é escolhido aleatoriamente e os seus argumentos são trocados.




(Figura de Engelbrecht, p.153)

## Mutação por crescimento

- Mutação por crescimento: um nodo é selecionado aleatoriamente e é substituído por uma sub-árvore gerada aleatoriamente. A nova sub-árvore tem altura máxima limitada (pré-definida).



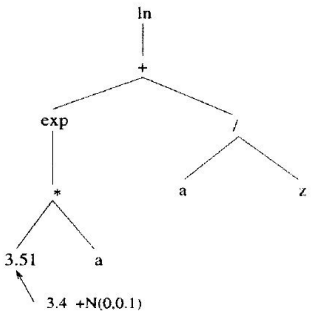

(Figura de Engelbrecht, p.153)





## Mutação Gaussiana

- Mutação Gaussiana: um nodo terminal que represente uma constante é escolhido e o seu valor é alterado através da adição dum valor aleatório proveniente duma distribuição Gaussiana.



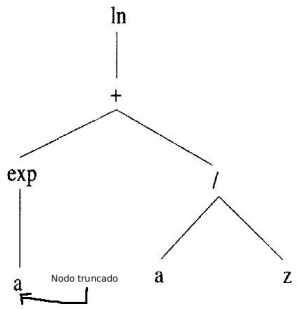

(Figura de Engelbrecht, p.153)





## Mutação por truncatura

- Mutação por truncatura: um nodo funcional é escolhido aleatoriamente e substituído por um nodo aleatório do conjunto terminal. O efeito desta mutação é um encurtamento da árvore.




(Figura de Engelbrecht, p.153)





## Probabilidades de mutação

- Os indivíduos são selecionados de acordo com uma probabilidade de mutação dos indivíduos, $p_{mi}$.
- De seguida, dentro de cada indivíduo (árvore) a mutação de cada nodo é feita de acordo com uma outra probabilidade: $p_m$, a probabilidade de mutação dos genes.
- Quanto maior $p_m$, maiores são as alterações nos genes dum dado indivíduo.
- Quanto maior é $p_{mi}$ maior é o número de indivíduos que em cada geração sofre mutações.
- Num dado algoritmo de PG todos os operadores de mutação podem ser usados ou apenas um subconjunto.
- Quando são usados vários, ou se escolhe de forma aleatória em cada mutação qual usar ou se usam vários de seguida.





## Leitura recomendada

- Engelbrecht, sec. 9.2-9.5, 9.7, 9.8.
- Engelbrecht, cap. 10.