Universidade de Fortaleza - UNIFOR

Sistemas Inteligentes - T296

Msc. Prof. Paulo Cirillo Souza Barbosa
Centro de Ciências Tecnológicas - CCT
Universidade de Fortaleza
Fortaleza, Ceará, Brasil

Sumário

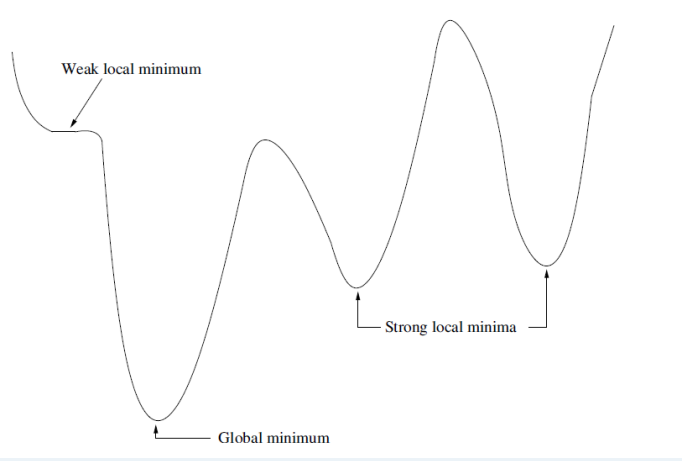
Otimização e Busca - Problemática.

- A **otimização** é uma área da matemática aplicada que possui o foco no estudo de métodos de resolução de problemas em que se procura **minimizar** ou **maximizar** uma função numérica.
- Tal processo é realizado pela escolha sistemática dos valores de certas variáveis comumente conhecidas como variáveis de decisão.
- A **busca** pode ser vista como uma metodologia de resolução de problemas que toma como base a sua formulação em um **espaço de estados** e um elemento neste espaço é visto como uma solução para o problema.
- Tendo em vista que nem toda solução tem a mesma qualidade, busca-se encontrar a **ótima** para o problema.
- Exemplos: Projeto de circuitos integrados, escalonamento de jornadas de trabalho, arranjo físico de maquinário em indústria, otimização de rede de telecomunicações, roteamento de veículos.

Otimização e Busca - Formalização do problema de otimização.

- Um problema de otimização geralmente apresentam três componentes básicas:
  1. Uma **função objetivo/custo** $f : \mathbb{R}^p \rightarrow \mathbb{R}$, que é o critério que deseja-se otimizar. Ou seja, a quantidade a ser minimizada ou maximizada.
  2. O **conjunto de de variáveis de decisão** $\mathbf{x} \in \mathbb{R}^p$, que afetam diretamente a função objetivo. Considerando $\mathbf{x}$ como variável independente, então $f(\mathbf{x})$ quantifica a qualidade da solução candidata $\mathbf{x}$
  3. Um **conjunto de restrições**, que limita os valores que podem ser atribuídos às variáveis independentes.
- Além disso, outros conceitos importantes como:
  1. **Espaço de estados** (S).
  2. **Vizinhança*** (V): Dado um ponto $\mathbf{x} \in S$, $V(\mathbf{x})$ representa todos os pontos $\mathbf{y} \in S$ que satisfazem $|\mathbf{x} - \mathbf{y}| \leq \varepsilon$.
  3. **Ótimo local ($\mathbf{x}_l*$):** $\mathbf{x}_l*$ é um mínimo/máximo local se $\forall \mathbf{x} \in V(\mathbf{x}_l*)$, $F(\mathbf{x}_l*) \leq F(\mathbf{x})$ (ou para máximo: $F(\mathbf{x}_l*) \geq F(\mathbf{x})$
  4. **Ótimo global ($\mathbf{x}_g*$):** $\mathbf{x}_g*$ é um mínimo/máximo global se $\forall \mathbf{x} \in S$, $F(\mathbf{x}_g*) \leq F(\mathbf{x})$ (ou para máximo: $F(\mathbf{x}_g*) > F(\mathbf{x})$)

Otimização e Busca.



Otimização e Busca.

- Um problema de otimização, pode ser categorizado por:
  1. A quantidade de variáveis: problema univariado ou multivariado.
  2. O tipo da variável: problema de domínio contínuo, discreto, misto ou até por permutações de inteiros (combinatória).
  3. Grau de não-linearidade da função objetivo.
  4. Restrições utilizadas: que definem a restrição no espaço de estados (neste caso o espaço é reduzido para $\mathbb{F}$).
  5. Quantidade de soluções ótimas: unimodal $\times$ multimodal.
  6. Quantidade de critérios de otimização.

Otimização e Busca.

- Um **algoritmo** de otimização, busca uma solução ótima por iterações que realizam uma pertubação de uma solução ótima corrente, na esperança de encontrar uma nova solução ótima.
- Os algoritmos também possuem suas categorias: local, global, determinístico ou estocástico.
- Um problema de otimização sem restrição, pode ser escrito da seguinte forma:

$$\begin{aligned} &minimize\ f(x), \mathbf{x} = (x_1, x_2, \cdots, x_p) \\ &subject\ to\ x_j \in dom(x_j) \end{aligned}$$

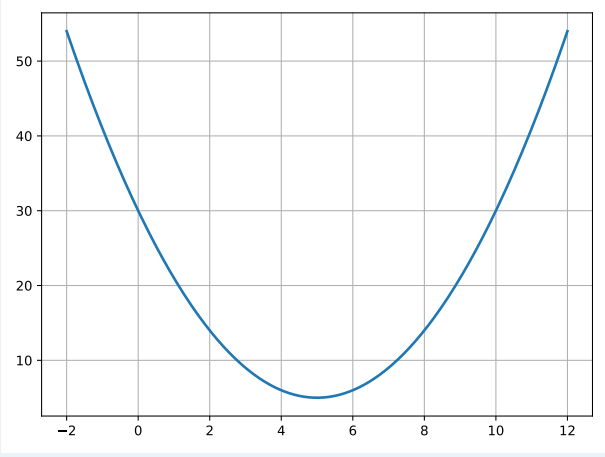
em que $dom(x_j)$ é o domínio da variável x_j e para um problema contínuo, este domínio para cada variável é o conjunto dos reais ($\mathbb{R}$).

Otimização e Busca - **busca local**.

- O problema pode ter natureza **discreta** ou **contínua**;
- No caso contínuo, chamado também de otimização numérica, pode-se ter um número infinito de possíveis soluções.
- No caso discreto, também chamado de otimização combinatória, as soluções são frutos de uma certa combinação de parâmetros discretos e possuem um número finito de soluções.
- Exemplos: sintonização de controladores PID ou redes neurais.
- Solução do problema do Caixeiro Viajante ou problema das 8 Rainhas.

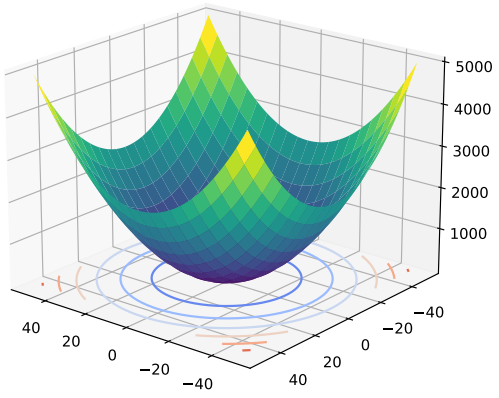
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



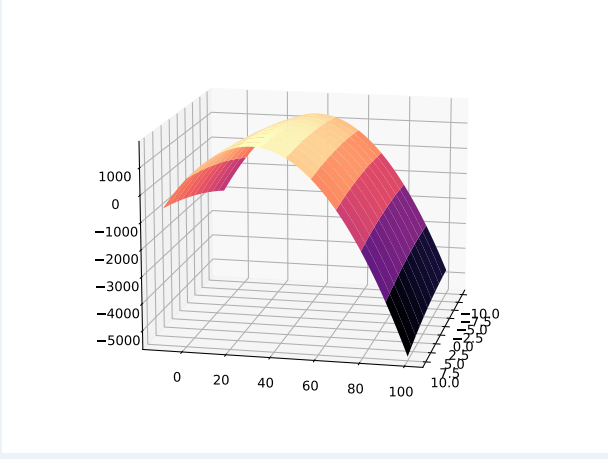
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



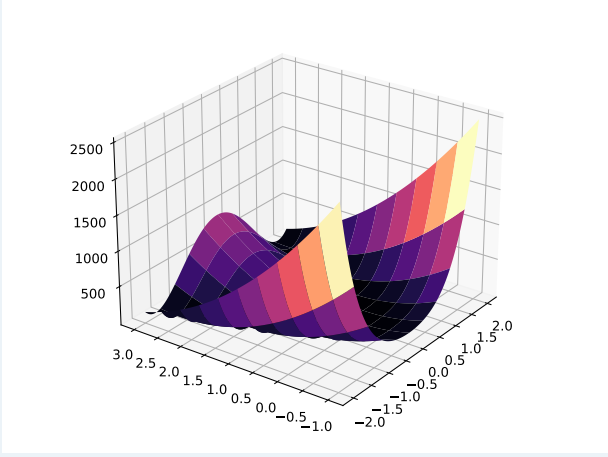
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



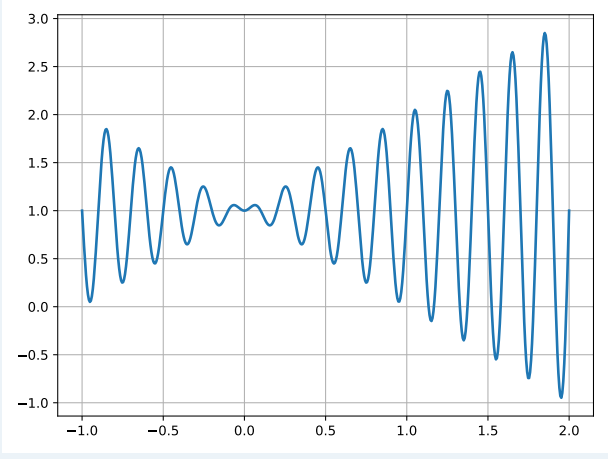
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



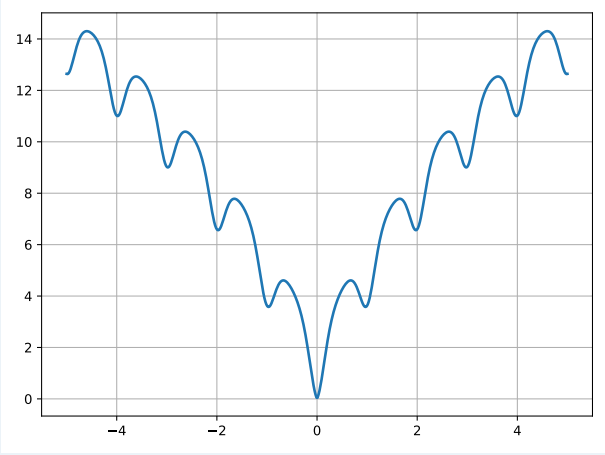
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



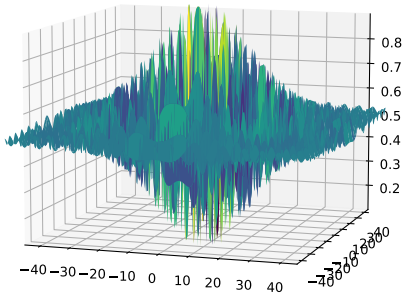
Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



Otimização e Busca - **busca local**.

- Exemplos de funções: unimodal $\times$ multimodal



Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*).

- É um tipo de algoritmo de busca heurística local.
- A partir de um estado inicial, escolhe um sucessor melhor "subir sempre"até encontrar um pico.
- Para problemas convexos, este algoritmo encontra valores de ótimo global.
- Caso o problema seja não-convexo (multimodal), encontra apenas o ótimo local.
- O algoritmo é baseado nos seguintes passos:

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*).

Algorithm 1: Pseudocódigo busca por subida de encosta.

```
1:  Inicializar o ponto inicial (zero ou limite de domínio) x_0
2:  definir o valor ε para candidato vizinho.
3:  Definir uma quantidade máxima de iterações max_it e quantidade máxima de candidatos (possíveis vizinhos) max_n.
4:  Melhor valor x_best ← x_0 e melhor valor computado f_best ← f(x_best).
5:  i ← 0
6:  while i < max_it E houver melhoria do
7:      j ← 0
8:      while j < max_n do
9:          j ← j + 1
10:         melhoria ← false
11:         y ← candidato(x_best).
12:         F ← f(y)
13:         if F > f_best then
14:             x_best ← y
15:             f_best ← F.
16:             melhoria ← true
17:             break
18:         end if
19:         j ← j + 1
20:     end while
21:     i ← i + 1
22: end while
23: FIM
```

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*).

Algorithm 2: Pseudocódigo candidato.

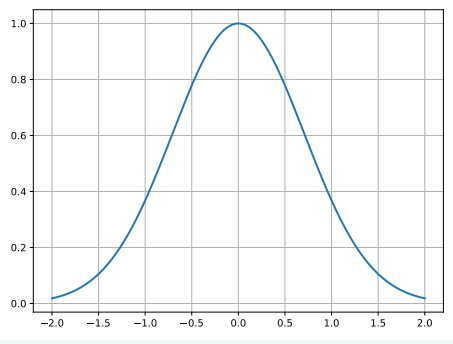
1: Definir o valor ε.
2: A partir de x Gerar vizinho-candidato aleatório y que respeite: $|x - y| <= \varepsilon$
3: FIM.

- Este algoritmo no que lhe concerne pode ser facilmente implementado em Python utilizando a biblioteca Numpy:
- **np.random.uniform**(low=x-E,high=x+E)

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 1.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = e^{-x^2} \quad -2 \geq X \geq 2$$

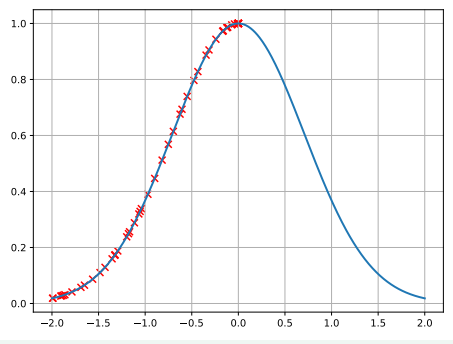


- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 1.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = e^{-x^2} \quad -2 \geq X \geq 2$$

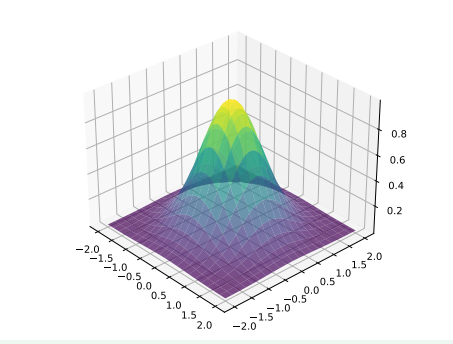


- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} \quad -2 \geq x \geq 2 \qquad -2 \geq y \geq 2$$

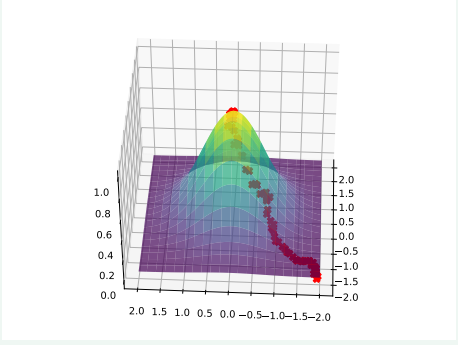


- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} \quad -2 \geq x \geq 2 \qquad -2 \geq y \geq 2$$

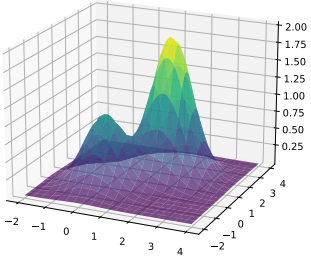


- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} + 2e^{-((x-1.7)^2+(y-1.7)^2)} \quad -2 \geq x \geq 4 \qquad -2 \geq y \geq 4$$

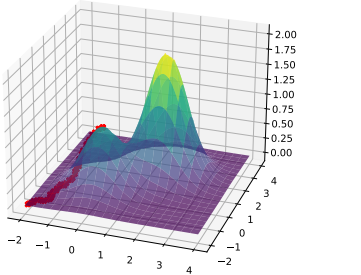


- É um problema convexo (unimodal)?

Otimização e Busca - **busca local** Algoritmo da Subida de Encosta (*Hill Climbing*) Exemplo 2.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x,y) = e^{-x^2+y^2} + 2e^{-((x-1.7)^2+(y-1.7)^2)} \quad -2 \geq x \geq 4 \qquad -2 \geq y \geq 4$$



- É um problema convexo (unimodal)?

Otimização e Busca - Busca Aleatória (*Random Search*)

- Considerando a definição de função objetivo/custo anterior: $f : \mathbb{R}^p \rightarrow \mathbb{R}$, que produz uma saída escalar $y \in \mathbb{R}$ e possui p ($p \geq 1$ variáveis de entrada.
- Pode-se formalmente escrever:

$$y = f(\mathbf{x}) = f(x_1, x_2, \cdots, x_p),$$

em que **x** é um vetor cujas componentes são variáveis x_i, $i = 1, \cdots, p$

- Nos três exemplos anteriores, respectivamente, tem-se $p = 1$, $p = 2$, $p = 2$ e são funções contínuas e de variação suave.
- As duas primeiras, são **convexas**, ou seja, possuem apenas **um** ponto extremo chamado de máximo local.
- A última é um exemplo de função não convexa, tendo em vista que há dois picos. Um de máximo local e outro máximo global.

Otimização e Busca - Busca Aleatória (*Random Search*)

- Em um problema formal de otimização de problema contínuo, o valor da variável x para o qual $y = f(x)$ produz seu maior valor, é chamado de valor **ótimo** de x (ou x_{opt}).
- Nesse sentido, x_{opt} também pode ser vinculado ao mínimo da função-custo.
- Se o problema envolve uma função com duas variáveis, **busca-se** um vetor **ótimo** $\mathbf{x}_{opt} = \begin{bmatrix} x_{opt} & y_{opt} \end{bmatrix}^T$.
- O problema de minimização de funções (sem restrição) pode ser formalizado matematicamente como: O vetor $\mathbf{x} \in \mathbb{R}^p$ é o vetor **ótimo** se:

$$f(\mathbf{x}_{opt}) < f(\mathbf{x}), \quad \forall \mathbf{x} \neq \mathbf{x}_{opt} \text{ OU } \mathbf{x}_{opt} = \arg\min_{\forall \mathbf{x}} f(\mathbf{x})$$

Otimização e Busca - Busca Aleatória (*Random Search*)

- A vizinhança já discutida, também pode ser formalmente escrita na forma **restrição de caixa**.
- Essa é descrita na forma de intervalos com limites inferiores e superiores para cada uma das p variáveis que compõem o **vetor solução** $\mathbf{x} \in \mathbb{R}^p$.
- Assim, escrevendo como a $j-$ésima componente de $\mathbf{x}$ como x_j, e seus limites inferior e superior como x_j^l e x_j^u, pode-se escrever a restrição do tipo caixa como:

$$x_j^l \leq x_j \leq x_j^u$$

- Ou através da versão vetorial:

$$\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$$

- Nesta última, $\mathbf{x}^l$ e $\mathbf{x}^u$ contém respectivamente, os limites inferiores e superiores para todas as componentes do vetor solução $\mathbf{x}$.

Otimização e Busca - Busca Aleatória (*Random Search*)

- **OBS:** O uso de "$\leq$"na última equação é uma *liberdade poética*, pois tal comparação não pode ser realizada entre dois vetores.
- Para lidar com valores que excedem o limite imposto pela restrição, podem-se utilizar os seguintes métodos:
  1. Gerar um número aleatório uniforme no intervalo $x_j^l \leq x_j \leq x_j^u$:

     $$x_j \sim U(x_j^l, x_j^u), \text{ se } x^j < x_j^l \text{ ou } x_j > x_j^u$$

     em que $u \sim U(a, b)$ representa um número aleatório uniformemente distribuído no intervalo (a,b).
  2. Forçar que a solução que ocasiona uma violação de limite, assuma o extremo limite mais próximo. Se $x^j < x_j^l$, então $x^j = x_j^l$. Se $x^j > x_j^u$, então $x^j = x_j^u$

Otimização e Busca - Busca Aleatória Local (*Local Random Search*)

- O algoritmo de busca aleatória local (LRS, do inglês *local random search*), consiste em testar soluções candidatas em uma vizinhança próxima ao $\mathbf{x}_{best}$.
- É uma heurística estocástica, ao dependerem de rotinas que geram números aleatórios.
- Possui uma única solução a cada iteração.
- Não é um método bioinspirado.
- Não depende de gradiente
- Pode ser utilizado para funções descontínuas.

Otimização e Busca - Busca Aleatória (*Local Random Search*)

- O algoritmo possui a seguinte sequência de passos:
  - 1.1 Especificar as restrições do tipo caixa $\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$ para todas as variáveis (especificar o domínio).
  - 1.2 Especificar o desvio-padrão σ do ruído (perturbação aleatória) a ser adicionado à melhor solução corrente para gerar candidatos.
  - 2 Inicializar a solução melhor inicial $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$.
  - 3 Avaliar a melhor solução inicial $\mathbf{x}_{opt}$, ou seja, calcular $f(\mathbf{x}_{opt})$.
  - 4 Gerar uma solução candidata $\mathbf{x}_{cand}$ com valor $\mathbf{x}_{best} + \mathbf{n}$. Em que $\mathbf{n} \in \mathbb{R}^d$ segue uma distribuição normal variada de vetor médio nulo e matriz covariância $\sigma^2\mathbf{I}_d$, ou seja, $\mathbf{n} \sim N(\mathbf{0}_d, \sigma^2\mathbf{I}_d)$. OBS: Caso haja violação das restrições, aplicar um dos métodos descritos anteriormente.
  - 5 Avaliar a melhor solução candidata $\mathbf{x}_{cand}$, ou seja, calcular $f(\mathbf{x}_{cand})$.

Otimização e Busca - Busca Aleatória (*Local Random Search*)

- O algoritmo possui a seguinte sequência de passos:
  6. Verificar como a solução candidata está com relação à melhor solução corrente.

$$Se\ f(\mathbf{x}_{cand}) > f(\mathbf{x}_{best}),\ \text{Então, } \mathbf{x}_{best} = \mathbf{x}_{cand} \quad f(\mathbf{x}_{best}) = f(\mathbf{x}_{cand})$$

  7. Vá para o passo 4 até o algoritmo não ter convergido (permaneça inalterado por um certo número de iterações) ou até o número máximo de iterações (N_{max}) seja atingido.
  8. Se uma das condições do passo 7 seja verdadeira, encerre a execução do algoritmo e retorne $\mathbf{x}_{best}$ e $f(\mathbf{x}_{best})$

Otimização e Busca - busca aleatória local (LRS).

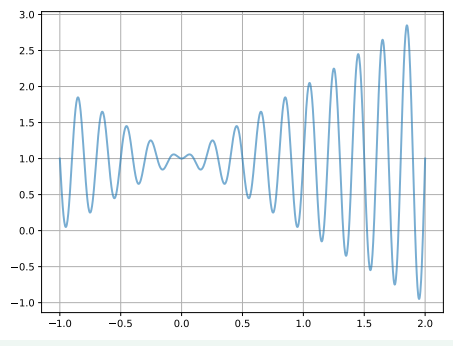
Algorithm 3: Pseudocódigo busca aleatória local.

1: Definir uma quantidade máxima de iterações N_{max}.
2: Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.
3: Definir valor de σ (perturbação aleatória).
4: $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$
5: $f_{best} = f(\mathbf{x}_{best})$
6: $i \leftarrow 0$
7: **while** $i < N_{max}$ **do**
8: $\quad \mathbf{n} \leftarrow\sim N(0, \sigma)$
9: $\quad \mathbf{x}_{cand} =\leftarrow \mathbf{x}_{best} + \mathbf{n}$
10: $\quad$ Verificar a violação da restrição em caixa.
11: $\quad f_{cand} = f(\mathbf{x}_{cand})$
12: $\quad$ **if** $f_{cand} > f_{best}$ **then**
13: $\quad\quad \mathbf{x}_{best} = \mathbf{x}_{cand}$
14: $\quad\quad f_{best} = f_{cand})$
15: $\quad$ **end if**
16: **end while**
17: FIM.

Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

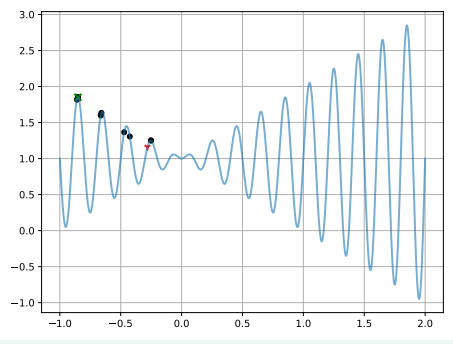


- É um problema convexo (unimodal)?

Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

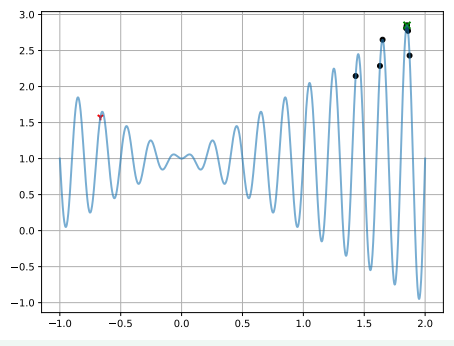


- É possível encontrar o máximo global (para este caso). Se sim, o que deve ser feito?

Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$



- É possível encontrar o máximo global (para este caso). Se sim, o que deve ser feito?

Otimização e Busca - Busca Aleatória Global (*Random Search*)

- O algoritmo de busca aleatória global (GRS), consiste em testar soluções candidatas que são geradas aleatoriamente dentro do domínio da função a ser otimizada.
- Esse método será utilizado para encontrar o ponto ótimo e o valor ótimo correspondente da função de interesse.
- O algoritmo é baseado em heurística, pois, não é derivado a partir de princípios matemáticos formais, mas sim de conhecimento intuitivo ou informal sobre o domínio do problema.
- Para determinados casos, não se conhece os pontos mínimos e máximos de uma função custo/objetivo. Dessa maneira, é um algoritmo que não possui garantia de encontrar a solução ótima.
- Para contornar tal problema, várias execuções do algoritmo devem acontecer. A identificação da solução subótima é dada pela solução com a maior **frequência** (moda das soluções).

Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo a ser apresentado é **estocástico** pois:
  1. Exige de uma solução inicial aleatória. Ou seja, a solução de partida, é gerada aleatoriamente.
  2. A geração de um candidato potencial a cada iteração é dependente de rotinas em que são gerados números aleatórios.
- É um método de solução única, em que apenas uma solução-candidata é gerada a cada iteração. Diferente de métodos populacionais como GAs
- Não é um método de inspiração biológica, pois sua formulação possui motivações puramente computacionais. Diferente de algoritmos bioinspirados.
- É um método livre de gradiente, ou seja, não requer cálculo de gradientes para determinar as direções de atualização de soluções. Por isso, **podem** ser utilizados para funções descontínuas (discretas).

Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo a ser apresentado é **estocástico** pois:
  1. Exige de uma solução inicial aleatória. Ou seja, a solução de partida, é gerada aleatoriamente.
  2. A geração de um candidato potencial a cada iteração é dependente de rotinas em que são gerados números aleatórios.
- É um método de solução única, em que apenas uma solução-candidata é gerada a cada iteração. Diferente de métodos populacionais como GAs
- Não é um método de inspiração biológica, pois sua formulação possui motivações puramente computacionais. Diferente de algoritmos bioinspirados.
- É um método livre de gradiente, ou seja, não requer cálculo de gradientes para determinar as direções de atualização de soluções. Por isso, **podem** ser utilizados para funções descontínuas (discretas).

Otimização e Busca - Busca Aleatória (*Random Search*)

- O algoritmo possui a seguinte sequência de passos:
  1. Especificar as restrições do tipo caixa $\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u$ para todas as variáveis (especificar o domínio).
  2. Inicializar a solução melhor inicial $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$.
  3. Avaliar a melhor solução inicial $\mathbf{x}_{best}$, ou seja, calcular $f(\mathbf{x}_{best})$.
  4. Gerar uma solução candidata $\mathbf{x}_{cand}$ com valores aleatórios uniformes: $\mathbf{x}_{cand} \sim U(\mathbf{x}^l, \mathbf{x}^u)$
  5. Avaliar a melhor solução candidata $\mathbf{x}_{cand}$, ou seja, calcular $f(\mathbf{x}_{cand})$.
  6. Verificar como a solução candidata está com relação à melhor solução corrente.

$$Se\ \ f(\mathbf{x}_{cand}) > f(\mathbf{x}_{best}),\ \text{Então},\ \mathbf{x}_{best} = \mathbf{x}_{cand}\quad f(\mathbf{x}_{best}) = f(\mathbf{x}_{cand})$$

  7. Vá para o passo 4 até o algoritmo não ter convergido (permaneça inalterado por um certo número de iterações) ou até o número máximo de iterações (N_{max}) seja atingido.
  8. Se uma das condições do passo 7 seja verdadeira, encerre a execução do algoritmo e retorne $\mathbf{x}_{best}$ e $f(\mathbf{x}_{best})$

Otimização e Busca - busca aleatória global.

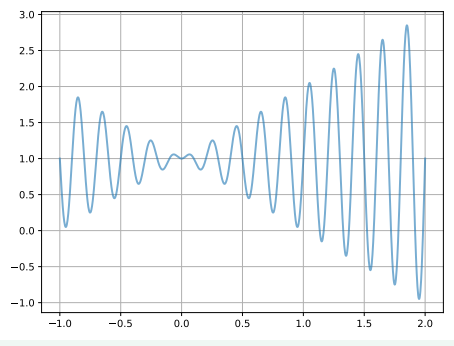
Algorithm 4: Pseudocódigo busca aleatória global.

1: Definir uma quantidade máxima de iterações N_{max}.
2: Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.
3: $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$
4: $f_{best} = f(\mathbf{x}_{best})$
5: $i \leftarrow 0$
6: **while** $i < N_{max}$ **do**
7: $\quad \mathbf{x}_{cand} = \leftarrow \sim U(\mathbf{x}^l, \mathbf{x}^u)$
8: $\quad f_{cand} = f(\mathbf{x}_{cand})$
9: $\quad$ **if** $f_{cand} > f_{best}$ **then**
10: $\quad\quad \mathbf{x}_{best} = \mathbf{x}_{cand}$
11: $\quad\quad f_{best} = f_{cand})$
12: $\quad$ **end if**
13: **end while**
14: FIM.

Otimização e Busca - **busca aleatória global** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$

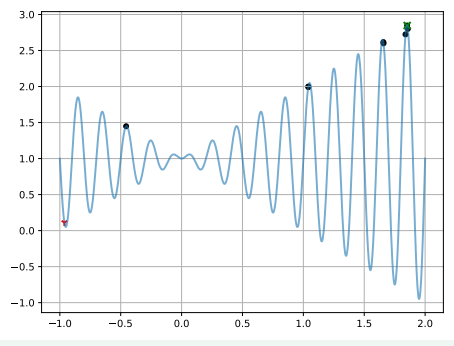


- É um problema convexo (unimodal)?

Otimização e Busca - **busca aleatória local** Exemplo.

- Considere que deseja-se maximizar o problema contínuo:

$$f(x) = x \cdot \sin(10 \cdot \pi \cdot x) + 1$$



- É um problema convexo (unimodal)?

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Algoritmo baseado em processo físico de têmpera.
- Tal procedimento é realizado para aumentar a dureza de certos metais ao aquecê-los em alta temperatura inicialmente e em seguida resfriando-os gradualmente.
- Na computação, é um algoritmo para solução de problema de otimização, através de uma técnica probabilística a fim de encontrar um ótimo global.
- Nesse sentido, pode ser utilizado para minimização da função-custo ou maximização da função-objetivo ($\max f(x)$ / $\min(-f(x))$).
- Imagine um exemplo em que será colocado uma bola de pingue-pongue em um dos gráficos não-convexos exibidos anteriormente.
- Deseja-se que essa bola atinja o ponto mínimo do gráfico.

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Pode ser que ao soltar a bola, ela acabe em um mínimo local (a depender a da função-custo é muito provável que aconteça).
- Contudo, se agitarmos a superfície, a bola pode ser deslocada para fora do mínimo local.
- O truque está em agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global.
- A solução de têmpera simulada trata-se de agitar inicialmente com força (ou seja, em alta temperatura) e depois reduzir gradualmente a intensidade da agitação (baixar a temperatura).
- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Pode ser que ao soltar a bola, ela acabe em um mínimo local (a depender a da função-custo é muito provável que aconteça).
- Contudo, se agitarmos a superfície, a bola pode ser deslocada para fora do mínimo local.
- O truque está em agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global.
- A solução de têmpera simulada trata-se de agitar inicialmente com força (ou seja, em alta temperatura) e depois reduzir gradualmente a intensidade da agitação (baixar a temperatura).
- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Diferente do algoritmo de subida de encosta, faz-se a escolha de um movimento aleatório. Caso essa escolha melhore a situação, essa é aceita. Caso contrário, o algoritmo aceita o movimento com alguma probabilidade menor que 1.
- A probabilidade diminui exponencialmente com a "má qualidade"do vizinho. Esta também diminui à medida que temperatura é reduzida.
- Ou seja, utiliza-se uma abordagem de escalonamento da temperatura ao longo das iterações do algoritmo.
- Comumente, a aceitação probabilística é baseada na distribuição de Boltzmann-Gibbs dada por.

$$P_{ij} = e^{-\frac{f(x_j) - f(x_i)}{T}}$$

em que x_j é o candidato, x_i é o ótimo (corrente) e T é a temperatura no instante atual.

Otimização e Busca - busca aleatória global.

Algorithm 5: Pseudocódigo Têmpera Simulada.

1: Definir uma quantidade máxima de iterações N_{max} e T.
2: Definir $\mathbf{x}^l$ e $\mathbf{x}^u$.
3: Definir valor de σ (perturbação aleatória).
4: $\mathbf{x}_{best} \sim U(\mathbf{x}^l, \mathbf{x}^u)$
5: $f_{best} = f(\mathbf{x}_{best})$
6: $i \leftarrow 0$
7: **while** $i < N_{max}$ **do**
8: $\quad \mathbf{n} \leftarrow\sim N(0, \sigma)$
9: $\quad \mathbf{x}_{cand} =\leftarrow \mathbf{x}_{best} + \mathbf{n}$
10: $\quad$ Verificar a violação da restrição em caixa.
11: $\quad f_{cand} = f(\mathbf{x}_{cand})$
12: $\quad$ **if** $f_{cand} < f_{best}$ **then**
13: $\quad\quad \mathbf{x}_{best} = \mathbf{x}_{cand}$
14: $\quad\quad f_{best} = f_{cand}$
15: $\quad$ **else if** $P_{ij} \geq U(0, 1)$ **then**
16: $\quad\quad \mathbf{x}_{best} = \mathbf{x}_{cand}$
17: $\quad\quad f_{best} = f_{cand}$
18: $\quad$ **end if**
19: $\quad i \leftarrow i + 1$
20: $\quad$ escalona(T)
21: **end while**
22: FIM.

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x \in \mathbb{R}, [-5, 5]$, construa uma implementação da têmpera simulada.

$$f(x) = -20e^{-0.2 \cdot |x|} - e^{\cos(2\pi \cdot x)} + 20 + e^1$$

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0 T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x \in \mathbb{R}, [-5, 5]$, construa uma implementação da têmpera simulada.

$$f(x) = -20e^{-0.2 \cdot |x|} - e^{\cos(2\pi \cdot x)} + 20 + e^1$$

Otimização e Busca - Têmpera Simulada (*Simulated annealing*)

- Faz sentido destacar que a convergência do algoritmo pode mudar a partir do escalonamento definido.

$$T_{i+1} = 0.99 \cdot T_i$$

$$T_{i+1} = \frac{T_i}{1 + 0.99 \cdot \sqrt{T_i}}$$

$$T_{i+1} = \frac{T_i}{1 + \frac{T_0 - t_i}{(i-1)T_0T_i} \cdot \sqrt{T_i}}$$

- Por exemplo, dada a função a seguir com domínio de $x_1 \in \mathbb{R}, [-1, 2]$ e $x_2 \in \mathbb{R}, [-1, 2]$ construa uma implementação da têmpera simulada.

$$f(x, y) = x^2 \sin(4 * \pi x) - y \sin(4 * \pi y + \pi) + 1$$

Otimização/Busca - Computação Evolucionária.

- O processo evolucionário pode ser abstraído como um processo de busca/otimização em que se tem o objetivo de melhorar a capacidade de um organismo (ou sistema) para sobreviver em ambientes dinâmicos e competitivos.
- Evolução é um tema amplo e pode ter diferentes interpretações (ex: cósmica, química, orgânico, sistemas feitos pelo homem), contudo, as aulas conduzidas têm um foco na evolução biológica.
- Mesmo com este foco, os debates na área são amplos, com a visão Darwinista e Lamarckista sendo as mais populares e aceitas.
- Enquanto Darwin (1809-1882) é considerado o fundador das teorias da evolução e origem comum, Lamarck (1749-1829) é possivelmente o primeiro a teorizar sobre a evolução biológica.

Otimização/Busca - Computação Evolucionária (Teoria de Lamarck).

- A teoria de evolução de Lamarck é baseada na **hereditariedade**. Ou seja, na herança de características adquiridas.
- A ideia principal é que indivíduos se adaptam ao longo de suas vidas e transmitem suas características aos seus descendentes, que continuam a se adaptar.
- Essa adaptação é baseada no conceito de **uso e desuso**. Ou seja, com o tempo os indivíduos perdem características que não são necessárias e aprimoram aquelas que são úteis.

Otimização/Busca - Computação Evolucionária (Teoria de Darwin e Wallace).

- A teoria da **seleção natural** de Darwin é a base da evolução biológica e pode se resumida como:
  - Em um mundo com recursos limitados e populações estáveis, cada indivíduo compete com outros para sobrevivência.
  - Os indivíduos que possuem as melhores características (traços), são mais propensos a sobrevivência e reprodução, passando suas características para sua prole.
  - Essas características desejáveis são herdadas pelas próximas gerações e ao longo do tempo, tornam-se dominantes entre a população.
- A segunda parte da teoria de Darwin afirma que, durante a produção de um organismo descendente, eventos aleatórios causam modificações nas características do organismo filho. Caso essas novas características são um benefício para o organismo, então as chances de sobrevivência são aumentadas.

Otimização/Busca - Computação Evolucionária (EC).

- A Computação Evolucionária (EC, do inglês *Evolutionary Computing*) refere-se a resolução de problemas em sistemas que utilizam modelos computacionais baseados no processo evolucionário biológico como:
  1. Seleção Natural;
  2. Sobrevivência do mais **apto**;
  3. Reprodução;
- Estas são as componentes fundamentais desses sistemas.

Algoritmo Evolucionário Genérico.

- A **evolução** via seleção natural de uma população de indivíduos aleatórios pode ser vista como uma busca no espaço de estados. Neste caso os possíveis valores do espaço são **cromossomos**.
- Nesse sentido, algoritmos evolucionários são baseados em busca estocástica a fim de encontrar uma solução **ótima** para determinado problema.
- A busca evolucionária é influenciada pelos seguintes componentes dos Algoritmos Evolucionários:
  1. **Codificação** (*encoding*) de soluções para o problema em forma de cromossomo;
  2. Uma **função** para avaliar a **aptidão** dos indivíduos.
  3. **Inicialização** de uma população inicial
  4. Operadores de **seleção**;
  5. Operadores de **reprodução**.

Algoritmo Evolucionário Genérico

Algorithm 6: Algoritmo Evolucionário Genérico.

1: Seja $t = 0$ o contador de gerações;
2: Crie e inicialize uma população $p-$dimensional, $C(0)$ com N indivíduos.
3: **while** critério de parada não foi aceito **do**
4: Avalie a função de aptidão, $f(\mathbf{x}_i(t))$ de cada indivíduo (cromossomo), $\mathbf{x}_i(t)$;
5: Aplique a operação de **Seleção**, para gerar $C(t+1)$;
6: Aplique a **Reprodução*** para criar descendentes;
7: Avance para a próxima geração, $t = t + 1$;
8: **end while**
9: FIM.

- Exemplo de partida para EA Genetic Cars.

Algoritmo Evolucionário Genérico

- As maneiras diferentes que os componentes dos Algoritmos Evolucionários podem ser implementados, resultam em diferentes paradigmas:
  1. **Algoritmos Genéticos (GA)**: Modelam a evolução genética;
  2. **Programação Genética (GP)**: Baseada em algoritmos genéticos, mas com programas individuais (representados por árvores).
  3. **Programação Evolucionária (EP)**: Que é uma derivação da simulação de comportamento adaptativo na evolução.
  4. **Estratégias Evolucionárias:** que são voltados para a modelagem de parâmetros estratégicos que controlam a variação da evolução (evolução da evolução).
  5. **Evolução Cultural (CE)**: que modela a evolução cultural de uma população e como a cultura influencia a evolução genética e fenotípica dos indivíduos.
  6. **Algoritmos Coevolucionários:** em que indivíduos inicialmente "desprovidos"evoluem através da cooperação ou em concorrência entre si, adquirindo as características necessárias para sobreviver.
- Apesar de existirem diferentes paradigmas, ambos compartilham a ideia que soluções melhores evoluem gradualmente a partir de uma população de soluções, contudo, diferem em sua representação das soluções, operadores genéticos e

Algoritmos Evolucionários - O cromossomo

- Na natureza, os organismos possuem características que influenciam sua capacidade de sobreviver e reproduzir.
- Essas características são representadas por uma imensa cadeia de informações contidas no cromossomo do organismo.
- Cromossomos são estruturas de moléculas compactas entrelaçadas de DNA.
- Cada cromossomo possui uma abundância de **genes**, em que cada gene representa a unidade de hereditariedade.
- Os genes determinam diversos aspectos da anatomia e fisiologia e cada indivíduo possui uma sequência única de genes.

Algoritmos Evolucionários - O cromossomo

- No contexto de EC, cada indivíduo representa uma solução-candidata de um problema de **otimização**.
- As características de um indivíduo são representadas por um **cromossomo** (algumas obras também referem-se a **genoma**).
- Essas características são referentes às variáveis do problema de otimização.
- Cada variável que necessita ser otimizada, é chamada de **gene**, a menor unidade de informação possível.
- As características de um indivíduo podem ser dividias em duas classes de informação evolucionária:
  1. Genótipo.
  2. Fenótipo.

Algoritmos Evolucionários - O cromossomo

- As características de um indivíduo podem ser dividias em duas classes de informação evolucionária:
  1. Genótipo:
     - Descreve a composição genética de um indivíduo, e como foi herdado de seus pais.
     - Representa qual tipo de gene o indivíduo possui.
     - É herdado dos pais e pode ser transmitido para a prole.
  2. Fenótipo:
     - São os traços comportamentais de um indivíduo
     - Este define a aparência de um indivíduo em termos comportamentais observáveis.
     - Normalmente essas características podem ser vistas na interação entre genótipo e ambiente.

Algoritmos Evolucionários - O cromossomo (representação)

- Um passo importante no projeto de EA, é encontrar uma representação apropriada das soluções-candidatas
- A complexidade e desempenho dos algoritmos de busca dependem desta representação.
- Alguém arrisca um novo palpite sobre qual seria essa representação?

Algoritmos Evolucionários - O cromossomo (representação)

- Um passo importante no projeto de EA, é encontrar uma representação apropriada das soluções-candidatas
- A complexidade e desempenho dos algoritmos de busca dependem desta representação.
- Alguém arrisca um novo palpite sobre qual seria essa representação?
- Bom, a maioria dos EAs representas suas soluções-candidatas como um vetor de um tipo específico de dados.
- Uma exceção está presenta na programação genética, em que seus indivíduos são representados em forma de árvore.
- A representação clássica, é um vetor de valores binários com tamanho fixo.
- Nesse caso, cada cromossomo é um vetor com tamanho n_d bits.

Algoritmos Evolucionários - O cromossomo (representação)

- A representação clássica, é um vetor de valores binários com tamanho fixo.
- Se o espaço de estados do problema possui uma dimensão p, cada indivíduo consiste em p variáveis, em que cada variável é **codificada** como uma sequência de bits.
- Se o problema possui variáveis de valores nominais (categóricos), cada variável pode ser codificada como uma cadeia de bits n_d-dimensional, em que 2^{n_d} é o número total de valores nominais para aquela variável.
- A abordagem comum para problemas com variáveis de valores contínuos, é projetar uma função que realiza o mapeamento de cada valor real para um vetor de bits com dimensão n_d. Ou seja, $h : \mathbb{R} \rightarrow (0, 1)^{n_d}$.
- Como os problemas anteriores, o domínio do espaço contínuo precisa ser restrito a um limite finito, $[x_{min}, x_{max}] = [l, u]$.

Algoritmos Evolucionários - O cromossomo (representação)

- Uma codificação binária padrão pode ser utilizada para transformar um indivíduo na forma $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_j & \cdots & x_p \end{bmatrix}$, com $x_j \in \mathbb{R}$ para a forma de valores binários $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_j & \cdots & \mathbf{b}_p \end{bmatrix}$, em que $\mathbf{b}_j \in \mathbb{R}^{n_d}$
- Neste caso, o total de bits é $n_b = p \cdot n_d$.
- A decodificação é um passo importante, pois, cada indivíduo é avaliado pela sua função aptidão.
- Assim, a função que decodifica cada $\mathbf{b}_j$ para sua representação em ponto flutuante pode é dada por:

$$\Phi(\mathbf{b}_j) = x_{min,j} + \frac{x_{max,j} - x_{min,j}}{2^{n_d} - 1} \cdot \left(\sum_{l=0}^{n_d - 1} b_j[n_d - l - 1] \cdot 2^l \right)$$

- Essa abordagem foi inicialmente proposta e utilizada para resolver problemas com variáveis contínuas em 75 por Holland e De Jong.

Algoritmos Evolucionários - O cromossomo (representação)

- Faz sentido destacar que a representação em sequência de bits, a busca é realizada em um espaço de estado discreto.
- Assim, o algoritmo pode falhar em encontrar um ótimo preciso.
- Uma maneira de contornar é definir a quantidade de bits que deseja-se trabalhar. Quando mais bits, mais precisão numérica se tem.
- Longos cromossomos são difíceis de manipular.
- De fato, a conversão número real para sequência de bits com tamanho n_d, a acurácia máxima é:

$$\frac{x_{max,j} - x_{min,j}}{2^{n_d} - 1}$$

- Problemáticas associadas a essa representação clássica serão discutidas em um momento futuro oportuno.

Algoritmos Evolucionários - População Inicial

- O primeiro passo para aplicar EA de modo a resolver um problema de otimização é gerar a população inicial. A maneira padrão para gerar essa população inicial é atribuir valores aleatórios para cada gene de cada cromossomo.
- O objetivo de uma seleção aleatória é garantir que a população inicial tem uma representação uniforme do espaço de estados, evitando que regiões sejam negligenciadas pelo processo de otimização.
- O tamanho da população inicial N tem consequências em termos de custo computacional e capacidade de exploração.
- Um grande número de indivíduos faz com que a diversidade aumente, e assim, melhorando a habilidade de exploração. Contudo, maior é a complexidade computacional por geração.
- Uma população pequena, pode representar uma pequena parte do espaço de estados. Assim, enquanto a complexidade computacional é baixa, o EA pode necessitar de mais gerações para convergência do que para uma população maior

Algoritmos Evolucionários - Função de Aptidão (*Fitness Function*)

- No modelo Darwinista de evolução, indivíduos com as melhores características tem melhores chances de sobreviver e reproduzir.
- Para determinar a capacidade de um indivíduo no EA de sobreviver, uma função matemática é usada para quantificar quão boa é a solução por ele representada.
- A função aptidão (ou *fitness function*) f mapeia uma representação de um cromossomo para um escalar, $f : \mathbf{\Gamma}^p \rightarrow \mathbb{R}$. Em que $\mathbf{\Gamma}$ representa o tipo de dado dos elementos do cromossomo p-dimensional.
- A função aptidão pode representar a função objetivo que descreve o problema de otimização.
- Geralmente, a função de aptidão produz uma medida absoluta de aptidão. Ou seja, a solução representada por um cromossomo é diretamente avaliada usando a função objetivo.

Algoritmos Evolucionários - Função de Aptidão (*Fitness Function*)

- Não necessariamente a representação cromossômica corresponde à representação esperada pela função objetivo.
- Nesses casos a função aptidão detalhada pode ser escrita:

$$f : S_C \xrightarrow{\Phi} S_X \xrightarrow{\Psi} \mathbb{R} \xrightarrow{\Upsilon} \mathbb{R}_+$$

- Em que S_C representa o espaço de estados da função objetivo.
- Φ, Ψ e Υ representam, respectivamente, a função decodificadora cromossômica, a função objetivo e a função escalonadora (opcional).
- Esta última, do ponto de vista de projeto de um EA, é utilizada para garantir valores positivos da função aptidão.

Algoritmos Evolucionários - Função de Aptidão (*Fitness Function*)

- É importante destacar que a existem diferentes problemas de otimização e que cada um tem uma influência em como a função aptidão é formulada.
  1. **Otimização sem restrição** é aquela que a função aptidão é simplesmente a função objetivo.
  2. **Otimização com restrição** é utilizada quando a função aptidão possui duas funções objetivos. A função objetivo original e uma função de restrição com penalidade.
  3. **Otimização com múltiplos objetivos** que utiliza uma abordagem de soma ponderada, agregando todos os sub-objetivos na função de otimização.
  4. **Otimização em problemas dinâmicos e ruidosos** em que os valores obtidos pela função mudam ao longo do tempo.
- Mais detalhes sobre problemas diferentes de otimização podem ser encontrados em:
  - Convex Optimization Book

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção**)

- **Seleção** é um dos principais operadores em EA e relaciona diretamente com o conceito da sobrevivência do mais apto de Darwin.
- O objetivo principal da seleção é destacar as melhores soluções.
- Isso é realizado por meio de dois passos principais em EA:
  1. **Seleção de uma nova população** selecionada no final de cada **geração** para servir como população da próxima geração. Essa escolha deve ser feita na prole, ou na combinação de prole e pais, de modo que os indivíduos selecionados sobrevivam as próximas gerações.
  2. **Reprodução:** A prole é criada através de outros operadores evolucionários (recombinação, mutação). Nesse sentido, a seleção para mutação deve acontecer para os indivíduos "fracos", com a esperança de resultar melhores características (traços) nestes. Para o caso da recombinação, os melhores indivíduos devem ser selecionados para que a próxima prole tenha o material genético destes indivíduos "superiores".

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção**)

- Há diversos operadores de seleção, contudo serão destacados os amplamente utilizados em EAs.
- Os operadores de seleção são categorizados por sua **pressão seletiva** (*takeover time*). É definido pela velocidade com que a melhor solução ocupará toda a população por aplicação repetida do operador de seleção.
- Um operador com uma alta pressão seletiva diminui a diversidade da população de modo mais rápido e leva a uma convergência prematura com soluções subótimas, bem como limita a capacidade de exploração da população.

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção Proporcional**)

- Seleção Proporcional:
  - É baseado na probabilidade de seleção proporcional a função de aptidão.
  - Desta maneira, os melhores indivíduos são selecionados com base na função aptidão.

$$p_i = \frac{f_\Psi(\mathbf{x}_i(t))}{\sum_{k=1}^{N} f_\Psi(\mathbf{x}_k(t))}$$

  - Em que N representa o total de indivíduos na população.
  - Esta abordagem pode ser mais generalista, ao utilizar a função escalonadora para limitar aos $\mathbb{R}_+$

$$p_i = \frac{f_\Upsilon(\mathbf{x}_i(t))}{\sum_{k=1}^{N} f_\Upsilon(\mathbf{x}_k(t))}$$

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção Proporcional**)

- Seleção Proporcional:
  - Esta abordagem pode ser mais generalista, ao utilizar a função escalonadora para limitar aos $\mathbb{R}_+$

$$p_i = \frac{f_\Upsilon(\mathbf{x}_i(t))}{\sum_{k=1}^{N} f_\Upsilon(\mathbf{x}_k(t))}$$

  - Para problemas de minimização, a função $f_\Upsilon(\cdot)$ ou $\Upsilon(\cdot)$ pode simplesmente ser:

$$f_\Upsilon(\mathbf{x}_i(t)) = \Upsilon(\mathbf{x}_i(t)) = f_{max}(t) - f_\Psi(\mathbf{x}_i(t))$$

$$f_\Upsilon(\mathbf{x}_i(t)) = \Upsilon(\mathbf{x}_i(t)) = \frac{1}{1 + f_\Psi(\mathbf{x}_i(t)) - f_{min}(t)}$$

  - Nessas, $f_\Psi(\mathbf{x}_i(t)) = \Psi(\mathbf{x}_i(t))$ é a função de aptidão não escalonada, e $f_{min}(t)$ e $f_{max}(t)$ são respectivamente, o mínimo e o máximo observado até aquela geração

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção Proporcional**)

- Seleção Proporcional:
  - Esta abordagem pode ser mais generalista, ao utilizar a função escalonadora para limitar aos $\mathbb{R}_+$

$$p_i = \frac{f_\Upsilon(\mathbf{x}_i(t))}{\sum_{k=1}^{N} f_\Upsilon(\mathbf{x}_k(t))}$$

  - Para problemas de maximização, a função $f_\Upsilon(\cdot)$ ou $\Upsilon(\cdot)$ pode simplesmente ser:

$$f_\Upsilon(\mathbf{x}_i(t)) = \Upsilon(\mathbf{x}_i(t)) = \frac{1}{1 + f_{max}(t) - f_\Psi(\mathbf{x}_i(t))}$$

  - Nessas, $f_\Psi(\mathbf{x}_i(t)) = \Psi(\mathbf{x}_i(t))$ é a função de aptidão não escalonada, e $f_{min}(t)$ e $f_{max}(t)$ são respectivamente, o mínimo e o máximo observado até aquela geração

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção Proporcional**)

- A seleção dos indivíduos pode ser realizada utilizando o método da **roleta** que é um dos exemplos mais clássicos de seletor proporcional para funções de aptidão normalizadas.

Algorithm 7: Seleção pelo método da Roleta.

1: Seja $i = 1$, em que i representa o $i-$ésimo cromossomo.
2: Computar a probabilidade p_i do cromossomo $\mathbf{x}_i$.
3: $Soma = p_i$
4: $r \sim U(0, 1)$
5: **while** $Soma < r$ **do**
6: $\quad i = i + 1$
7: $\quad Soma = Soma + p_i$
8: **end while**
9: retorna $\mathbf{x}_i$ como indivíduo selecionado.

Algoritmos Evolucionários - Operadores Evolucionários (Seleção)

- Outro método para seleção é chamado de **seleção por torneio**. Este, seleciona um grupo de n_{st} indivíduos aleatórios da população (em que $n_{st} < N$).
- O desempenho dos n_{st} são avaliados e o melhor indivíduo deste grupo é selecionado.
- Escolhendo um grupo não tão grande, a seleção por torneio previne que o melhor indivíduo sempre domine.
- Contudo, um n_{st} pequeno, faz com que as chances de indivíduos ruins sejam selecionados, aumentem.
- Mesmo sendo um método que utiliza a função aptidão para selecionar o melhor indivíduo, a seleção aleatória dos indivíduos reduz a pressão seletiva em comparação a seleção proporcional.
- Contudo, a pressão seletiva é diretamente proporcional a quantidade de indivíduos no torneio. Se $n_{st} = N$, então, o melhor indivíduo sempre será selecionado, resultando em uma alta pressão seletiva.

Algoritmos Evolucionários - Operadores Evolucionários (Seleção)

- Outro método para seleção é chamado de **seleção por torneio**. Este, seleciona um grupo de n_{st} indivíduos aleatórios da população (em que $n_{st} < N$).
- O desempenho dos n_{st} são avaliados e o melhor indivíduo deste grupo é selecionado.
- Escolhendo um grupo não tão grande, a seleção por torneio previne que o melhor indivíduo sempre domine.
- Contudo, um n_{st} pequeno, faz com que as chances de indivíduos ruins sejam selecionados, aumentem.
- Mesmo sendo um método que utiliza a função aptidão para selecionar o melhor indivíduo, a seleção aleatória dos indivíduos reduz a pressão seletiva em comparação a seleção proporcional.
- Contudo, a pressão seletiva é diretamente proporcional a quantidade de indivíduos no torneio. Se $n_{st} = N$, então, o melhor indivíduo sempre será selecionado, resultando em uma alta pressão seletiva.

Algoritmos Evolucionários - Operadores Evolucionários (**Seleção**)

- Há também a seleção por **Elitismo**. Em que os melhores indivíduos de uma população atual são mantidos para uma próxima geração. Neste caso, nenhuma operação de mutação é realizada.
- Algumas bibliografias consideram que o Elitismo é um operador a parte.
- É uma seleção determinística.
- É importante ter em mente que quanto maior o grupo elitista, menor é a diversidade da nova população.

Algoritmos Evolucionários - Operadores de Reprodução

- Reprodução é o processo de criação da prole através da seleção de parentes ao aplicar os operadores:
  1. Recombinação
  2. Mutação.
- A recombinação é o processo de criar um ou mais indivíduos, através da combinação genética selecionada de modo aleatório de dois ou mais pais.
- A mutação é o processo de modificar valores dos genes em um cromossomo de maneira aleatória. O objetivo principal desta abordagem é introduzir um novo material genético na população, e assim, aumentar a diversidade.
- A mutação precisa ser aplicada com cuidado, pois, pode distorcer um bom material genético de indivíduos aptos.
- Normalmente, define-se uma probabilidade de aplicação da mutação. Esta pode ser proporcional a aptidão dos indivíduos. Quanto menor a aptidão do indivíduo, maior é sua mutação.

Algoritmos Evolucionários - Critérios de Parada

- Os diferentes operadores evolucionários são aplicados nos EA, até que um critério de parada seja atingido.
- Um critério de parada simples é limitar o número máximo de gerações que o EA é permitido.
- Outro critério pode ser associado à convergência da população. Neste caso a **convergência** acontece quando uma população se torna estagnada. Ou seja, quando não há modificações genotípicas ou fenotípicas na população.

Algoritmos Evolucionários - Critérios de Parada

- Os critérios de convergência podem ter a natureza:
  1. Parada quando nenhuma melhoria é observada ao longo de uma quantidade de gerações:
     - Esta pode ser identificada ao monitorar a aptidão do melhor indivíduo.
     - Se não há mudança significante ao longo de uma janela de gerações, então o EA deve ser parado.
  2. Parada quando não há modificações na população.
     - Se em um determinado período de gerações consecutivas, as transformações médias genotípicas são pequenas, então EA deve ser parado.
  3. Parada quando uma solução aceitável é encontrada:
     - Considerando que $\mathbf{x}^*(t)$ representa o ótimo da função aptidão, então, se o melhor indivíduo atual $\mathbf{x}_i$ é um tal que $\Psi(\mathbf{x}_i) \leq |\Psi(\mathbf{x}^*) - \varepsilon|$, então uma solução aceitável foi encontrada. Nesta, ε representa o erro limite.
     - É importante destacar que um valor alto para ε significa encontrar soluções possivelmente ruins.
     - Escolher um valor muito pequeno para ε pode causar a repetição infinita do EA (caso o máximo de gerações não seja definido).

Algoritmos Genéticos.

- Algoritmos Genéticos (GAs) são possivelmente o primeiro modelo que simula sistemas genéticos.
- Seu conceito foi desenvolvido por uma combinação de pesquisadores: Fraser (1957), Bremermann (1962) e Holland (1975).
- A popularização está muito associada a Holland, que é considerado o pai de Algoritmos Genéticos.
- GAs, tem a característica de expressar seus indivíduos utilizando genótipos.
- É baseado na produção de populações a cada geração através da aplicação sequencial de três operadores:
  1. Seleção: para modelar a sobrevivência do mais apto.
  2. Recombinação: para modelar reprodução.
  3. Mutação: para modelar a diversidade.

Algoritmos Genéticos - Representação Cromossômica.

- Em GA, o $i-$ésimo cromossomo ou indivíduo da população é representado com um vetor-linha de dimensão p, ou seja, $\mathbf{x} \in \mathbb{R}^p$:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{ip} \end{bmatrix}$$

- Em que o elemento $x_{ij} \in \mathbb{R}$ representa o $j-$ésimo gene do $i-$ésimo cromossomo.

Algoritmos Genéticos - Representação Cromossômica.

- Assim, uma população de N indivíduos na geração t é representada como uma matriz $\mathbf{P}(t) = \mathbf{X}(t)$ de dimensão $N \times p$

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \vdots \\ \mathbf{x}_i(t) \\ \vdots \\ \mathbf{x}_N(t) \end{bmatrix} =$$

Algoritmos Genéticos - Representação Cromossômica.

- Assim, uma população de N indivíduos na geração t é representada como uma matriz $\mathbf{P}(t) = \mathbf{X}(t)$ de dimensão $N \times p$

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \vdots \\ \mathbf{x}_i(t) \\ \vdots \\ \mathbf{x}_N(t) \end{bmatrix} = \begin{bmatrix} x_{11}(t) & x_{12}(t) & \cdots x_{1j}(t) & \cdots & x_{1p}(t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i1}(t) & x_{i2}(t) & \cdots x_{ij}(t) & \cdots & x_{ip}(t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{N1}(t) & x_{N2}(t) & \cdots x_{Nj}(t) & \cdots & x_{Np}(t) \end{bmatrix}$$

- Cada linha da matriz representa uma solução-candidata para o problema de interesse.
- A cada geração, N soluções candidatas são avaliadas pelo valor da função aptidão: $f_i(t) = f(\mathbf{x}_i(t))$.

Algoritmos Genéticos - Representação Cromossômica.

- O GA canônico proposto por Holland segue o mesmo princípio do algoritmo já exibido anteriormente, contudo, com as seguintes características específicas:
  - Uma representação cromossômica por sequência binária.
  - Seleção proporcional é utilizada para selecionar indivíduos para recombinação.
  - Recombinação de ponto único como principal método de reprodução.
  - Mutação uniforme proposta como operador em "plano de fundo"de mínima importância.
- Vale ressaltar que a **mutação** originalmente não era um operador de importância em GA canônico. A medida que as pesquisas se aprofundaram no tema, é que o poder explorativo da mutação foi utilizado para melhorar as capacidades de busca de GAs.
- Além disso, como é de se esperar, após a proposta do modelo canônico, existem muitas variações de GAs que se diferem em representação, operadores de seleção, operadores de recombinação, operadores de mutação.

Algoritmos Genéticos - Operador de Recombinação.

- O operador de recombinação pode ser dividido em três categorias baseadas na quantidade de parentes.
  1. **Assexuada**: em que a prole é gerada a partir de um parente.
  2. **Sexuada**: em que dois parentes são utilizados para produzir um ou dois filhos.
  3. **Múltipla**: em que mais de dois pares são utilizados para produzir um ou mais filhos.
- Além disso, o operadores de recombinação podem ser baseados na forma (schema) de represtação cromossômica (operadores específicos para binários e operadores específicos para números em ponto flutuante).
- A seleção de parentes é realizada utilizando qualquer um dos operadores já discutidos nos slides anteriores.

Algoritmos Genéticos - Operador de Recombinação.

- A seleção de parentes é realizada utilizando qualquer um dos operadores já discutidos nos slides anteriores (exemplo: método do torneio).
- Contudo, não há garantia de que os parentes selecionados, irão acasalar.
- A recombinação é aplicada probabilisticamente.
- Ou seja, cada par possui uma probabilidade de produzir filhos. Uma alta $(0,85 < p_rs < 1)$ probabilidade é utilizada para isso acontecer.
- **Observações**:
  1. Como há uma probabilidade de seleção de parentes, pode ser o caso em que o casal selecionado para reprodução seja composto de um mesmo indivíduo. Neste caso, a prole é uma cópia do parente.
  2. É possível também que um indivíduo esteja presente em mais de uma operação de recombinação.
- Deve-se ainda levar em consideração que a recombinação tem uma política de substituição.

Algoritmos Genéticos - Recombinação (Representação Binária).

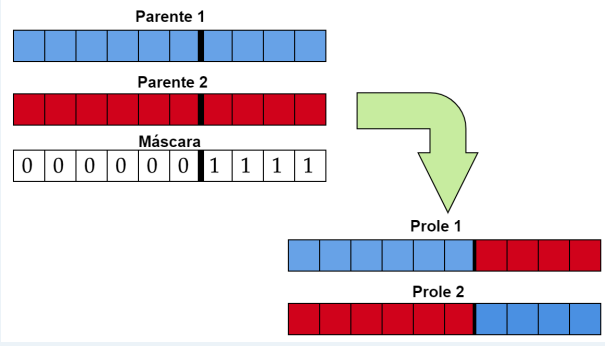
- Em geral, as operações de recombinação em uma representação binária é **sexuada**.
- Considere $\mathbf{x}_1(t)$ e $\mathbf{x}_2(t)$ como dois indivíduos selecionados para acasalamento. A abordagem genérica para produção da prole é:

Algorithm 8: Recombinação Genérica por sequência binária.

Seja $\tilde{\mathbf{x}}_1(t) = \mathbf{x}_1(t)$ e $\tilde{\mathbf{x}}_2(t) = \mathbf{x}_2(t)$;
if $U \sim (0,1) \leq p_r$ **then**
  Compute a máscara binária $\mathbf{m}(t)$;
  for $j = 0, \cdots, n_d$ **do**
    if $m_j == 1$ **then**
      Troque os bits:
      $\tilde{\mathbf{x}}_{1j}(t) = \tilde{\mathbf{x}}_{2j}(t)$
      $\tilde{\mathbf{x}}_{2j}(t) = \tilde{\mathbf{x}}_{1j}(t)$
    end if
  end for
end if

Algoritmos Genéticos - Recombinação (Representação Binária).

- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação de um ponto:** proposto por Holland, faz com que se decida, de modo aleatório, um ponto de recombinação na sequência de bits. As sequências binárias após esse ponto são trocadas entre parentes.



Algoritmos Genéticos - Recombinação (Representação Binária).

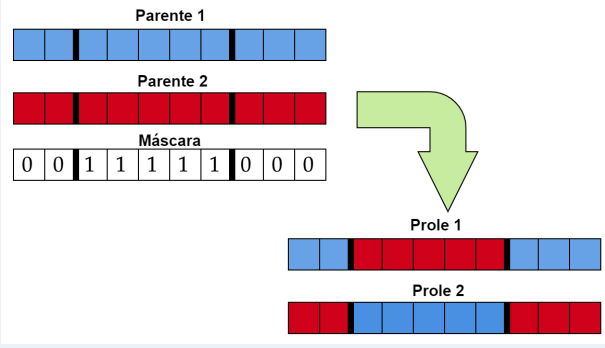
- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação de um ponto:** proposto por Holland, faz com que se decida, de modo aleatório, um ponto de recombinação na sequência de bits. As sequências binárias após esse ponto são trocadas entre parentes.

Algorithm 9: Geração da Máscara pela recombinação de um ponto.

1: Selecione o ponto de recombinação $\xi \sim U(1, n_d - 2)$;
2: Inicialize a máscara com zeros: $m_j(t) = 0, \ \ \forall \ \ j = 0, 1, \cdots, n_d - 1$;
3: **for** $j = \xi$ até $n_d - 1$ **do**
4: $\quad m_j(t) = 1$
5: **end for**
6: FIM.

Algoritmos Genéticos - Recombinação (Representação Binária).

- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação de dois pontos:** Neste caso, duas posições são selecionadas de modo aleatório e as sequências binárias dos parentes são trocadas.



Algoritmos Genéticos - Recombinação (Representação Binária).

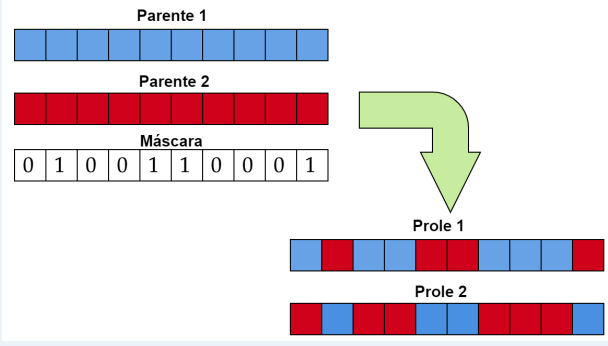
- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação de dois pontos:** Neste caso, duas posições são selecionadas de modo aleatório e as sequências binárias dos parentes são trocadas.

Algorithm 10: Geração da Máscara pela recombinação de um ponto.

1: Selecione os pontos de recombinação $\xi_1, \xi_2, \sim U(1, n_d - 2)$;
2: Inicialize a máscara com zeros: $m_j(t) = 0, \quad \forall \ j = 0, 1, \cdots, n_d - 1$;
3: **for** $j = \xi_1$ até ξ_2 **do**
4: $\quad m_j(t) = 1$
5: **end for**
6: FIM.

Algoritmos Genéticos - Recombinação (Representação Binária).

- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação Uniforme:** Neste caso, a máscara é gerada de maneira aleatória.



Algoritmos Genéticos - Recombinação (Representação Binária).

- Há diferentes maneiras de como computar a máscara de bits:
  1. **Recombinação Uniforme:** Neste caso, a máscara é gerada de maneira aleatória.

Algorithm 11: Geração da Máscara pela recombinação de um ponto.

1: Inicialize a máscara com zeros: $m_j(t) = 0, \quad \forall \ j = 0, 1, \cdots, n_d - 1$;
2: **for** $j = 0$ até n_d **do**
3: **if** $U(0,1) \leq p_x$ **then**
4: $m_j(t) = 1$
5: **end if**
6: **end for**
7: FIM.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

- Para modelos não canônicos, pode-se utilizar operadores de recombinação em representações em ponto flutuante da sequência genética.
- Realizando um comparativo, nas operações de recombinação de uma sequência binária, são trocados bits de informação entre parentes. Na representação em ponto flutuante no que lhe concerne, os operadores são desenvolvidos para misturar as informações dos pais selecionados.
- Um dos primeiros métodos aplicados nesta representação, chama-se método de Wright:
  - Este, faz uma operação linear entre dois parentes $\mathbf{x}_1(t)$ e $\mathbf{x}_2(t)$, gerando três filhos candidatos $(\mathbf{x}_1(t) + \mathbf{x}_2(t))$, $(1.5\mathbf{x}_1(t) - 0.5\mathbf{x}_2(t))$ e $(-0.5\mathbf{x}_1(t) + 1.5\mathbf{x}_2(t))$
  - As duas melhores soluções são selecionadas como prole.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

- O método de Wright pode ser descrito:

Algorithm 12: Recombinação Genérica por sequência binária.

1: Seja $\mathbf{x}_1(t)$ e $\mathbf{x}_2(t)$ o par de cromossomo presente no grupo de selecionados;
2: **if** $U \sim (0,1) \leq p_c$ **then**
3: Indivíduo 1: $\tilde{\mathbf{x}}_1(t) \leftarrow \mathbf{x}_1(t) + \mathbf{x}_2(t)$
4: Indivíduo 2: $\tilde{\mathbf{x}}_2(t) \leftarrow 1.5\mathbf{x}_1(t) - 0.5\mathbf{x}_2(t)$
5: Indivíduo 3: $\tilde{\mathbf{x}}_3(t) \leftarrow -0.5\mathbf{x}_1(t) + 1.5\mathbf{x}_2(t)$
6: Retorne os dois indivíduos da prole com os maiores valores de aptidão.
7: **end if**
8: FIM.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

- Outro método amplamente utilizado, foi proposto por Deb e Agrawal, chamado de método de Recombinação Binária Simulada (SBX, do inglês, *Simulated Binary Crossover*):
  - Este, faz uma operação entre dois parentes $\mathbf{x}_1(t)$ e $\mathbf{x}_2(t)$ para gerar a prole:

$$\tilde{x}_{1j} = 0.5((1+\gamma_j)x_{1j}(t) + (1-\gamma_j)x_{2j}(t))$$

$$\tilde{x}_{2j} = 0.5((1-\gamma_j)x_{1j}(t) + (1+\gamma_j)x_{2j}(t))$$

  - Para todos os $j-$ésimos termos. Ou seja, $j = 1, \cdots, p$
  - γ deve ser calculado, utilizando a seguinte regra:

$$\gamma_j = \begin{cases} (2r_j)^{\frac{1}{\eta+1}} & , r_j \leq 0.5 \\ \left(\frac{1}{2(1-r_j)}\right)^{\frac{1}{\eta+1}} & C.C \end{cases}$$

- Em que $r_j \sim U(0,1)$ e $\eta > 0$ é o índice de distribuição.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

- O operador SBX tem as seguintes características:
  1. Gera sua prole simetricamente em torno dos pais, prevenindo viés em torno de um dos pais.
  2. Nesse sentido, viés significa um filho ser mais parecido com um dos pais.
  3. η é um hiperparâmetro do algoritmo e deve ser definido > 0. Os autores do algoritmo, sugerem um valor igual a 1.
  4. Para grandes valores de η, há uma alta probabilidade de que a prole será criada próxima dos pais. Ou seja, a prole será muito semelhante aos pais.
  5. Para valores pequenos de η, a prole será mais distante dos pais, ou seja, menos semelhantes.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

- O método SBX pode ser reescrito de modo vetorial, ao considerar:
  - Este, faz uma operação entre dois parentes $\mathbf{x}_1(t)$ e $\mathbf{x}_2(t)$ para gerar a prole:

$$\tilde{\mathbf{x}}_1 = 0.5((\mathbf{1}_p + \boldsymbol{\gamma}) \circ \mathbf{x}_1(t) + (\mathbf{1}_p - \boldsymbol{\gamma}) \circ \mathbf{x}_2(t))$$

$$\tilde{\mathbf{x}}_2 = 0.5((\mathbf{1}_p - \boldsymbol{\gamma}) \circ \mathbf{x}_1(t) + (\mathbf{1}_p + \boldsymbol{\gamma}) \circ \mathbf{x}_2(t))$$

  - Nesta, $\mathbf{1}_p$ é um vetor de 1s com dimensão p.
  - $\boldsymbol{\gamma}$ é o vetor aleatório de dimensão p, cujas componentes são definidas pela equação:

$$\gamma_j = \begin{cases} (2r_j)^{\frac{1}{\eta+1}} & , r_j \leq 0.5 \\ \left(\frac{1}{2(1-r_j)}\right)^{\frac{1}{\eta+1}} & C.C \end{cases}$$

- Em que $r_j \sim U(0,1)$ e $\eta > 0$ é o índice de distribuição.

Algoritmos Genéticos - Recombinação (Representação Em Ponto Flutuante).

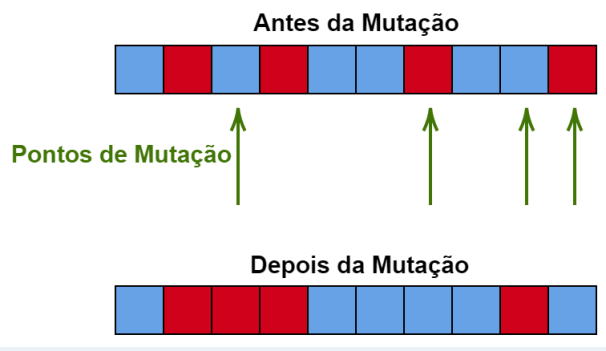
- A operação de recombinação tem como objetivo de conferir certa variabilidade genética em função da troca de material genético entre indivíduos.
- Simultaneamente, pode-se dizer que mantém uma espécie de memória genética da população.
- A intensidade da troca genética, é conduzida pela probabilidade de recombinação.
- Se p_c é muito baixa, então haverá menor troca de informação genética devido à recombinação. Assim, a cada geração, haverá maior inércia (lentidão) com relação a mudanças genéticas na população.
- Por esse motivo p_c deve possui um valor alto como já mencionado.

Algoritmos Genéticos - Mutação.

- O objetivo da mutação é introduzir um novo material genético em um indivíduo existente.
- Ou seja, adicionar diversidade nas características genéticas da população.
- Mutação é utilizada como suporte ao operador de recombinação, de modo a garantir que a faixa de alelos sejam acessíveis para cada gene.
- Assim como a recombinação, a mutação é aplicada com uma certa probabilidade p_m (taxa de mutação) em cada gene da prole.
- Assim, a prole, $\tilde{\mathbf{x}}_i(t)$ após a aplicação do operador de mutação, é $\mathbf{x}'_i(t)$.
- A taxa de mutação deve ser escolhida normalmente com valor pequeno (exemplo: 5%). Isto, pois, garante que as soluções boas não sejam tão distorcidas.

Algoritmos Genéticos - Mutação (Representação Binária).

- Para uma mutação em uma representação binária, podem-se destacar os métodos:
  1. **Mutação Uniforme:** Em que as posições de bits (pontos de mutação) são escolhidas de modo aleatório e em seguida, o bit daquela posição é **negado** (ou do jargão da área, "togglado").



Algoritmos Genéticos - Mutação (Representação Binária).

- Para uma mutação em uma representação binária, podem-se destacar os métodos:
  1. **Mutação Uniforme:** Em que as posições de bits (pontos de mutação) são escolhidas de modo aleatório e em seguida, o bit daquela posição é **negado** (ou do jargão da área, "togglado").

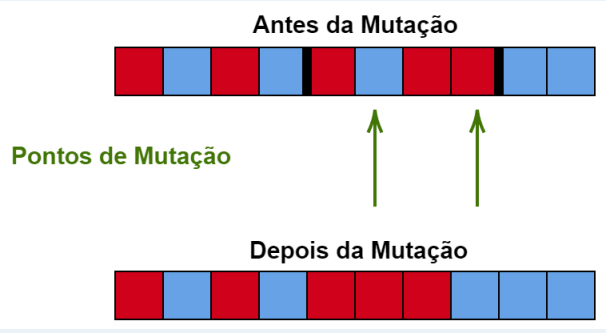
Algorithm 13: Mutação Uniforme.

1: **for** $j = 1$ até n_d **do**
2: **if** $U(0,1) \leq p_m$ **then**
3: $x'_{ij}(t) = \neg \tilde{x}_{ij}(t)$
4: **end if**
5: **end for**
6: FIM.

- Em que $\neg$ representa a negação do $j-$ésimo bit.

Algoritmos Genéticos - Mutação (Representação Binária).

- Para uma mutação em uma representação binária, podem-se destacar os métodos:
  1. **Mutação fora de ordem:** Neste caso, posições na sequência binária são escolhidas de modo aleatório e os bits entre os pontos podem ser alternados como exibe o algoritmo:



Algoritmos Genéticos - Mutação (Representação Binária).

- Para uma mutação em uma representação binária, podem-se destacar os métodos:
  1. **Mutação fora de ordem:** Neste caso, posições na sequência binária são escolhidas de modo aleatório e os bits entre os pontos podem ser alternados como exibe o algoritmo:

Algorithm 14: Mutação Fora de Ordem.

1: Selecione os pontos de mutação, $\xi_1, \xi_2 \sim U(0, n_d - 1)$
2: **for** $j = \xi_1$ até ξ_2 **do**
3: **if** $U(0,1) \leq p_m$ **then**
4: $x'_{ij}(t) = \neg \tilde{x}_{ij}(t)$
5: **end if**
6: **end for**
7: FIM.

- Em que $\neg$ representa a negação do $j-$ésimo bit.

Algoritmos Genéticos - Mutação (Representação Binária).

- Uma outra abordagem, que é amplamente utilizada para representações binárias, bem como para representação em ponto flutuante, é chamada de **Mutação Gaussiana**.
- Para o caso binário, deve-se decodificar a sequência de volta a sua representação em ponto flutuante e assim aplicar a mutação baseada numa perturbação normal (gaussiana).
- Esse procedimento já foi aplicado anteriormente?
- Em seguida, o valor obtido é codificado novamente para sua versão binária.
- Deve-se considerar, nesse caso, que há a possibilidade de estouro da restrição de domínio imposta às variáveis.
- Outra discussão importante, é que os algoritmos exibidos anteriormente (com representação binária), podem acarretar um alto custo computacional se a sequência de bits for muito grande. Uma maneira de contornar esse problema é dividir a sequência em segmentos e assim selecionar de modo aleatório os

Algoritmos Genéticos - Mutação (Representação em ponto flutuante).

- O método de **mutação gaussiana** discutido anteriormente, possui uma melhor desempenho quando aplicado já em uma representação em ponto flutuante, de acordo com resultados obtidos por Hinterding e Michalewicz.
- Assim, a mutação normal pode ser aplicada da seguinte maneira (de modo vetorial):

Algorithm 15: Mutação Gaussiana.

1: Seja $\tilde{\mathbf{X}}(t)$ a prole gerada na $t-$ésima geração.
2: **for** $i = 1$ até n_p **do**
3: **if** $U(0,1) \leq p_m$ **then**
4: $\mathbf{x}_i'(t) = \tilde{\mathbf{x}}_i(t) + \eta(\mathbf{x}^u - \mathbf{x}^l) \circ \mathbf{n}$
5: **end if**
6: **end for**
7: FIM.

Algoritmos Genéticos - Mutação (Representação em ponto flutuante).

- No algoritmo anterior, p_m é a probabilidade de mutação.
- η representa o passo de mutação e deve ser definido entre $0 < \eta \ll 1$.
- $\circ$ representa o operador de produto de Hadamard (ou seja, *element-wise product*).
- $\mathbf{n}$ é um vetor aleatório $p-$dimensional amostrado de uma densidade de probabilidade gaussiana multivariada de vetor médio nulo e matriz de covariância identidade, ou seja, $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_p)$

Algoritmos Genéticos - Mutação **Recapitulando**.

- O operador de mutação tem como objetivo, conferir certa inovação genética aos indivíduos da prole.
- A intensidade da inovação é controlada pela probabilidade de mutação p_m.
- Se p_m é muito alta, haverá excesso de inovação, o que pode destruir a história (memória) genética da população.
- Neste caso, o GA se assemelha a uma busca global com N soluções independentes por geração.
- Desta maneira, p_m deve ser mantida em um valor baixo (por exemplo, $p_m < 5\%$).

Algoritmos Genéticos - Exemplo 1.

- Considere como exemplo inicial, a seguinte problemática:
  1. Encontrar o máximo da função:

$$f(x) = x \cdot \sin(10\pi x) + 1,0$$

  2. Que possui restrição no domínio de x em: $-1,0 \leq x \leq 2,0$.
  3. Para solucionar este problema, considere que serão utilizados no algoritmo genético:
     - I Uma representação cromossômica binária com 20 bits.
     - II Uma população inicial com $N = 20$ indivíduos.
     - III A função aptidão é a própria função objetivo.
     - IV Utilizar o método de seleção do método do torneio (em pares).
     - V Para recombinação, utilizar o operador de um ponto e com probabilidade de 75%.
     - VI A mutação, é aplicada na prole com uma probabilidade de 1%.
     - VII Como critério de parada, defina um número máximo de gerações em 30.

Algoritmos Genéticos - Exemplo 2.

- Considere como exemplo inicial, a seguinte problemática:
  1. Encontrar o mínimo da função:

$$f(x) = -20e^{-0.2|x|} - e^{\cos(2\pi x)} + 20 + e$$

  2. Que possui restrição no domínio de x em: $-5,0 \leq x \leq 5,0$.
  3. Para solucionar este problema, considere que serão utilizados no algoritmo genético:
     - I Uma representação cromossômica em ponto flutuante. Para este caso, qual valor de p?
     - II Uma população inicial com $N = 30$ indivíduos.
     - III A função aptidão é a própria função objetivo.
     - IV Realizar uma análise comparativa entre método do torneio (em pares) e método da roleta.
     - V Para recombinação, utilizar o operador de Recombinação Binária Simulada de um ponto e com probabilidade de 85%.
     - VI A mutação, é aplicada na prole com uma probabilidade de 1%. Neste caso, utilize a perturbação $\eta = 0.01$ e $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_p)$
     - VII Como critério de parada, defina um número máximo de gerações em 30x' .