



FIAP



FIAP GRADUAÇÃO

| Sistemas para internet

# **Inteligência artificial e Machine Learning**

**Prof<sup>a</sup>. Thais Rodrigues Neubauer**  
[profthais.neubauer@fiap.com.br](mailto:profthais.neubauer@fiap.com.br)

2020

- 1. Definições e exemplos*
- 2. Buscas sem informação*
- 3. Buscas com informação*
- 4. Busca local*
- 5. Algoritmos genéticos*

**1. Definições e exemplos**

**2. Buscas sem informação**

**3. Buscas com informação**

**4. Busca local**

**5. Algoritmos genéticos**

## Referências base deste material:

- Russel, S.; Norvig, P. Inteligência Artificial. 2<sup>a</sup> edição. Editora Campus, 2004. Capítulos 3 e 4.
- Koza, J. R. Genetic Programming. MIT Press, 1992. Capítulo: 3.
- Linden, R. Algoritmos Genéticos. Brasport, 2006. Capítulos: 4 a 7 e 9.

\* Este material foi construído com base no da Profa. Dra. Sarajane Marques Peres.



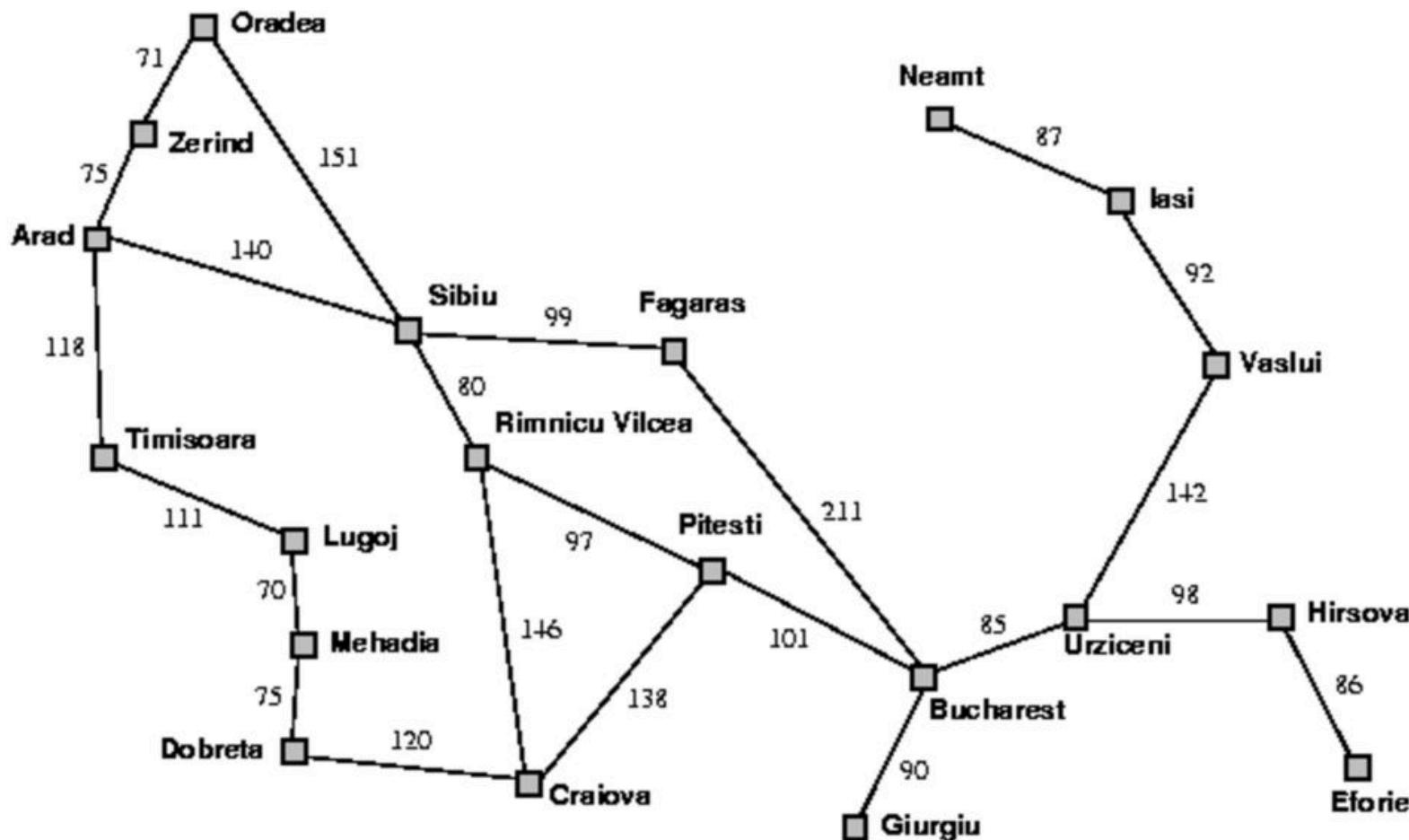
Representação – definição do problema:

- *Estado inicial*
- *Ações possíveis*
- *Teste de objetivo*
- *Função de custo* – custo numérico da solução

**ESPAÇO DE BUSCA = ESTADO INICIAL + REALIZAÇÃO DAS AÇÕES POSSÍVEIS**

# BUSCAS – Exemplo: Chegar em Bucareste

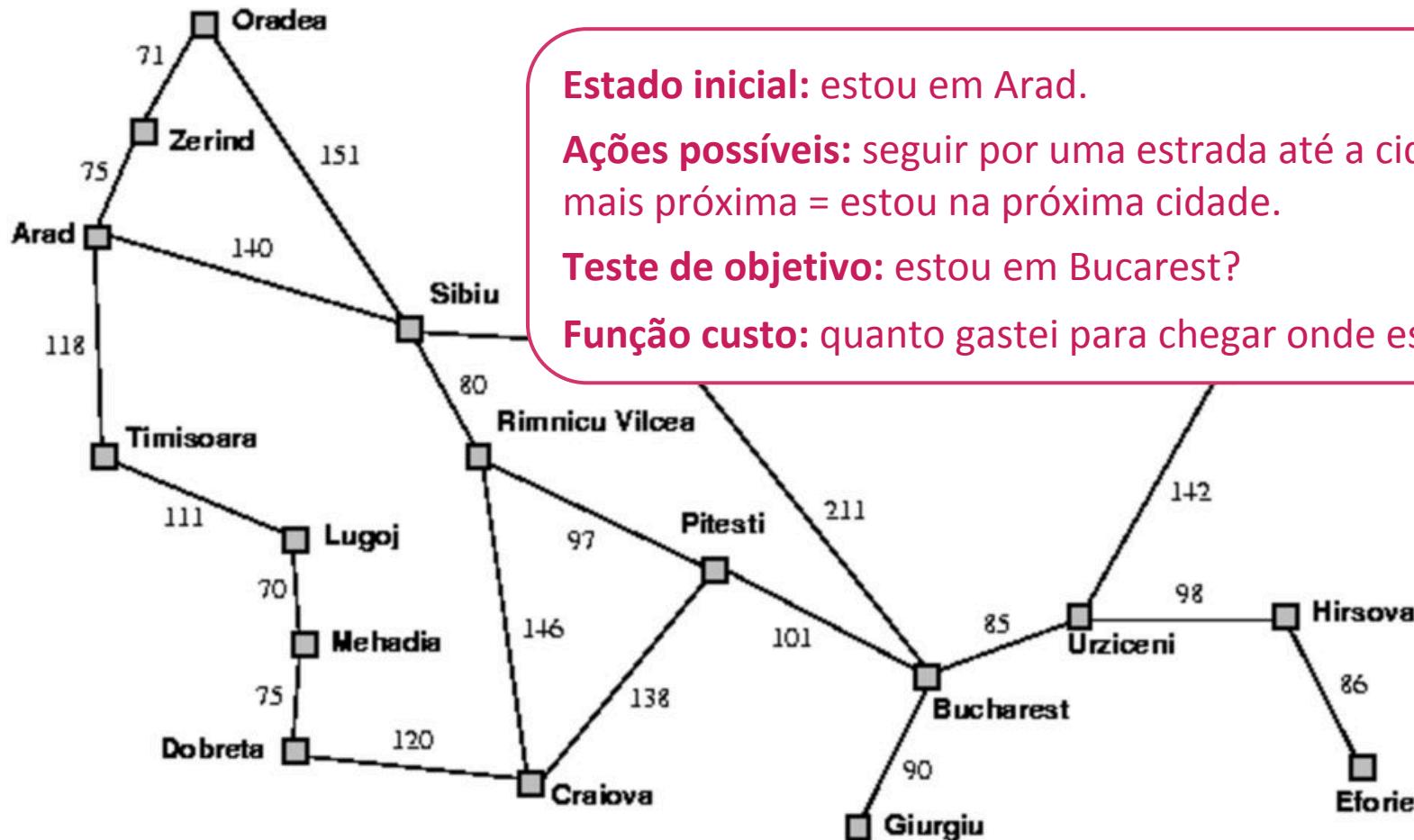
FIAP



**Problema: CHEGAR EM BUCARESTE**  
**Resultado da busca: o CAMINHO**

# BUSCAS – Exemplo: Chegar em Bucareste

FIAP



**Estado inicial:** estou em Arad.

**Ações possíveis:** seguir por uma estrada até a cidade mais próxima = estou na próxima cidade.

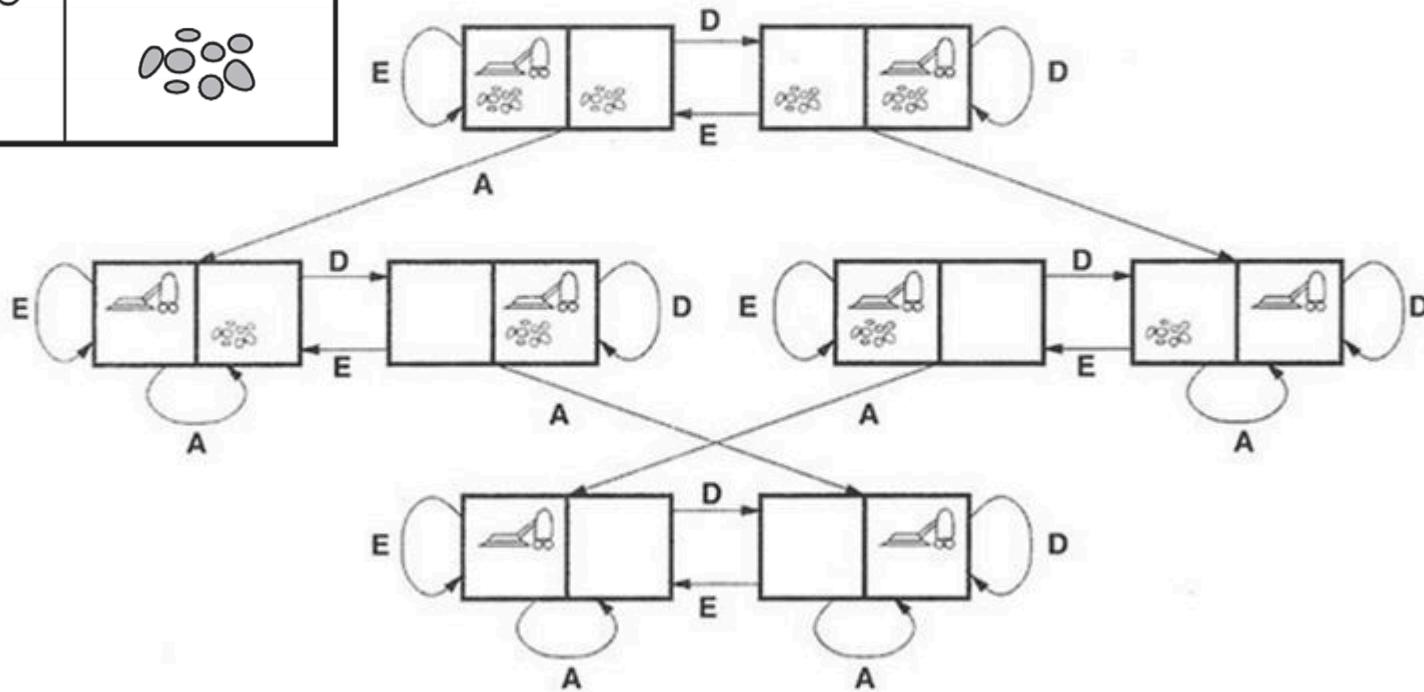
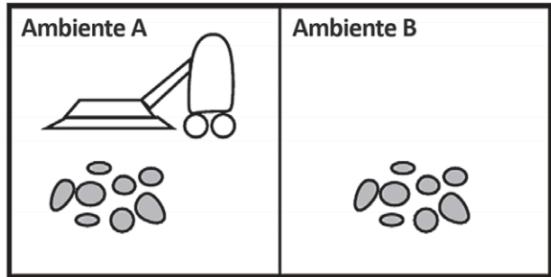
**Teste de objetivo:** estou em Bucareste?

**Função custo:** quanto gastei para chegar onde estou?

**Problema:** CHEGAR EM BUCARESTE  
**Resultado da busca:** o CAMINHO

# BUSCAS – Outros exemplos

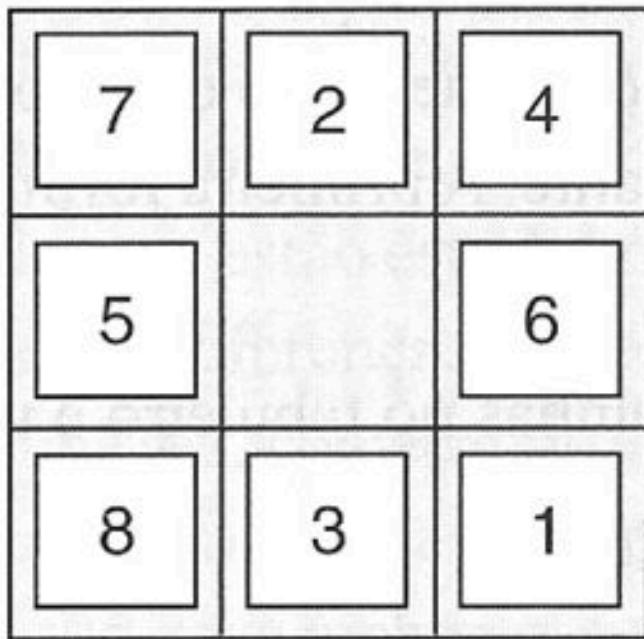
FIAP



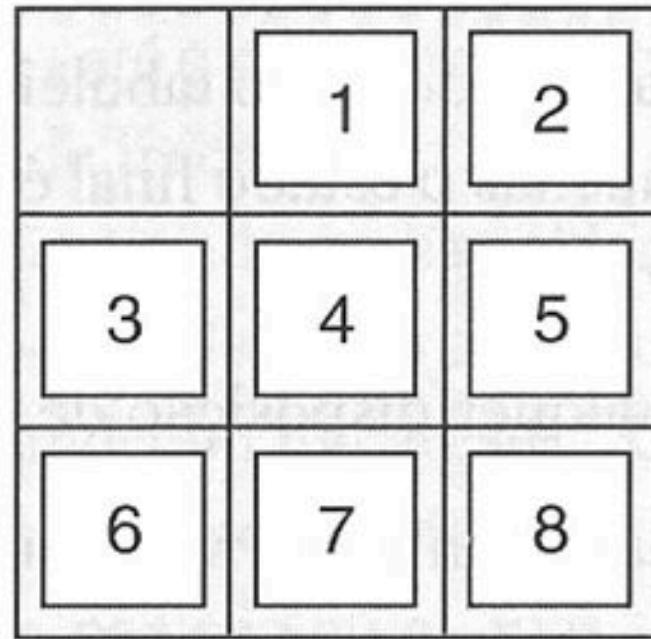
Os arcos denotam ações: E = Esquerda, D = Direita, A = Aspirar

**Problema:** ambientes limpos  
**Resultado da busca:** sequência de ações

## QUEBRA CABEÇA DE 8



Estado inicial

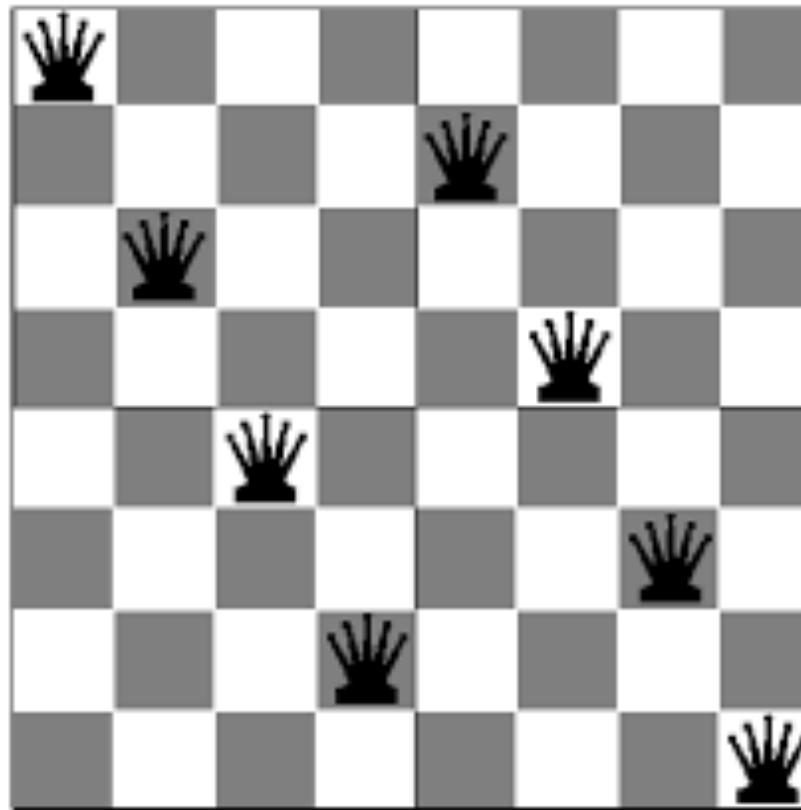


Estado objetivo

**Problema:** alcançar o estado objetivo

**Resultado da busca:** sequência de movimentos

# BUSCAS – Outros exemplos



**Problema:** posicionar 8 rainhas no tabuleiro de xadrez de forma que nenhuma rainha possa atacar outra. \* Uma rainha ataca na mesma linha, coluna ou diagonal.

**Resultado da busca:** estado com as rainhas posicionadas como solicitado.

- **Função SUCESSOR ( $x$ )**: dado o estado  $x$ , retorna um conjunto de pares  $\langle$ ação, sucessor $\rangle$ .
  - Cada ação é uma das ações válidas no estado  $x$ .
  - Cada sucessor é um estado que pode ser alcançado a partir de  $x$  aplicando-se a ação.

SUCESSOR( EM(Arad) ) = {  $\langle$ IR(Sibiu) ,EM(Sibiu) $\rangle$ ,  
 $\langle$ IR(Timissoara) ,EM(Timissoara) $\rangle$ ,  
 $\langle$ IR(Zerind) ,EM(Zerind) $\rangle$  }

- **Espaço de estados:** conjunto de todos os estados acessíveis a partir do estado inicial.
  - *Espaço de estados = estado inicial + função SUCESSOR*

**EM(Arad)** , **EM(Sibiu)** , **EM(Timissoara)** , **EM(Zerind)**

- **Custo do passo:** custo de aplicar a ação  $a$  para ir do estado  $x$  para o estado  $y$ .

**CUSTO (<EM(Arad) , EM(Sibiu)>) = 140**

- **Busca em árvore:** uma solução tradicional

**função BUSCA-EM-ÁRVORE (*problema, estratégia*) retorna** uma solução ou uma resposta sobre a “falha”

Iniciar a árvore de busca usando o estado inicial de *problema*  
**repita**

**se** não existe nenhum candidato para expansão

**então retornar** falha; // (exit)

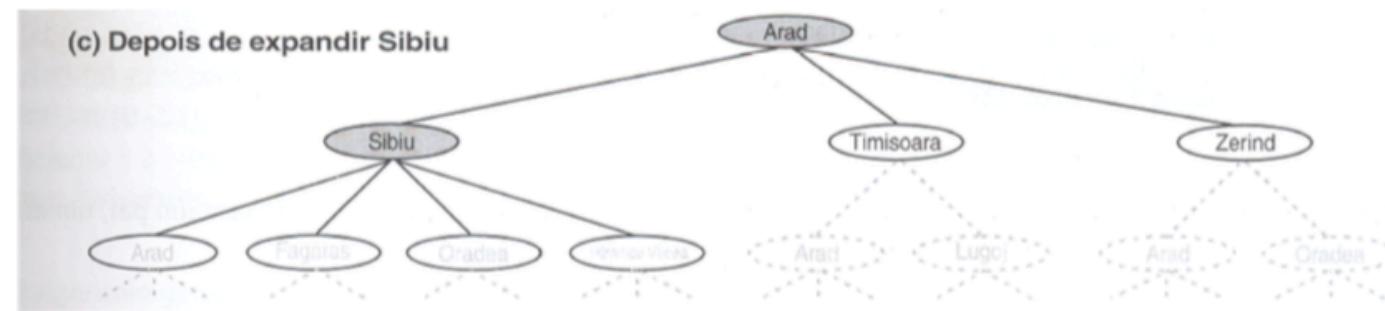
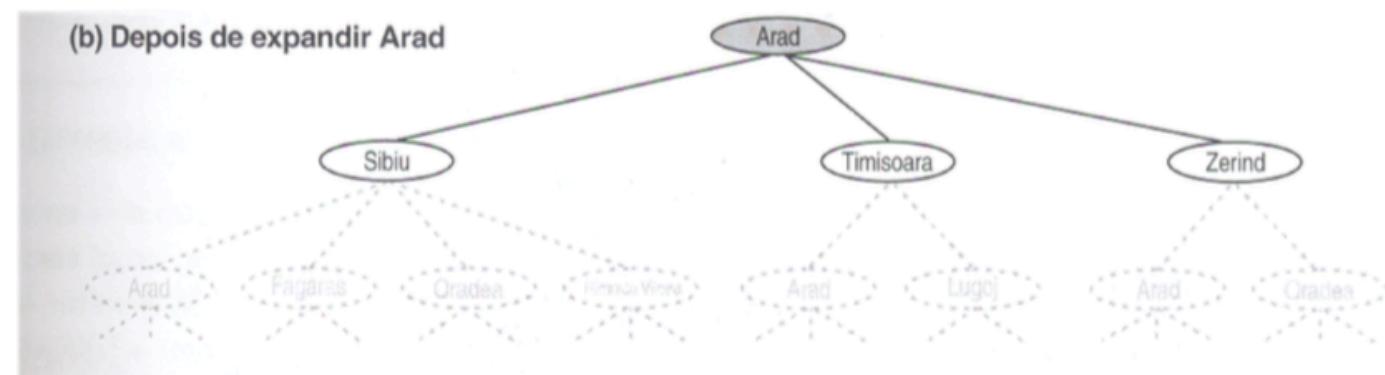
    escolher um nó folha para expansão de acordo com *estratégia*

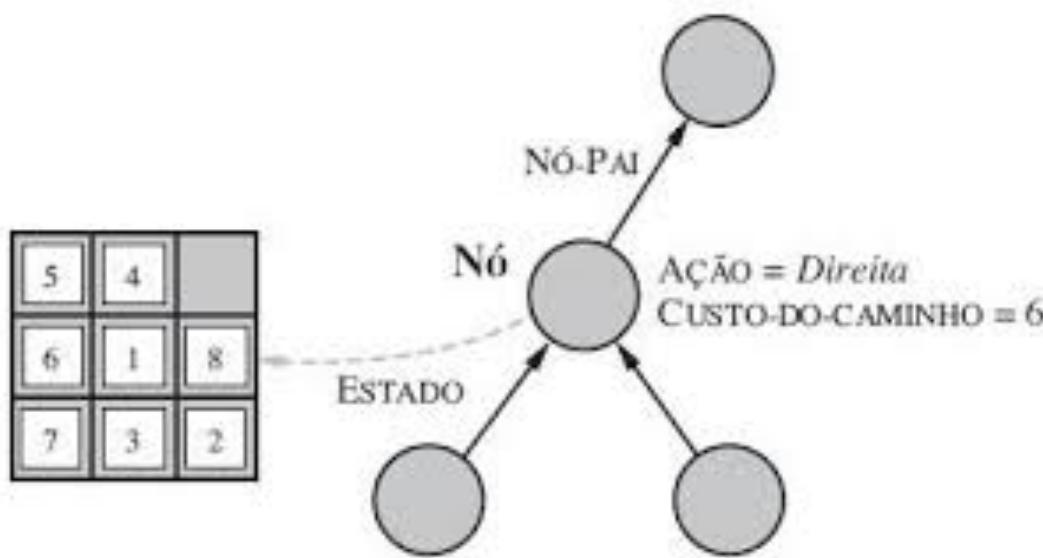
**se** o nó contém um ***estado objetivo*** então retornar a solução correspondente

**senão** expandir o nó e adicionar os nós resultantes à árvore de busca

# BUSCAS – Árvore: Chegar em Bucareste

FIAP





- **Estado:** o estado a que o nó atual corresponde
- **Nó pai:** nó da árvore de busca que gerou o nó atual
- **Ação:** ação que foi aplicada ao pai para gerar o nó atual
- **Custo do caminho:** custo desde o estado inicial
- **Profundidade:** número de passos ao longo do caminho

Como medir o desempenho do algoritmo usado na resolução do problema?

- **Completude:** o algoritmo garante encontrar uma solução quando ela existir?
- **Otimização:** a estratégia encontra a melhor solução? (solução ótima)
- **Complexidade de tempo:** quanto tempo o algoritmo leva para encontrar uma solução?
- **Complexidade de espaço:** quanta memória é necessária para executar a busca?

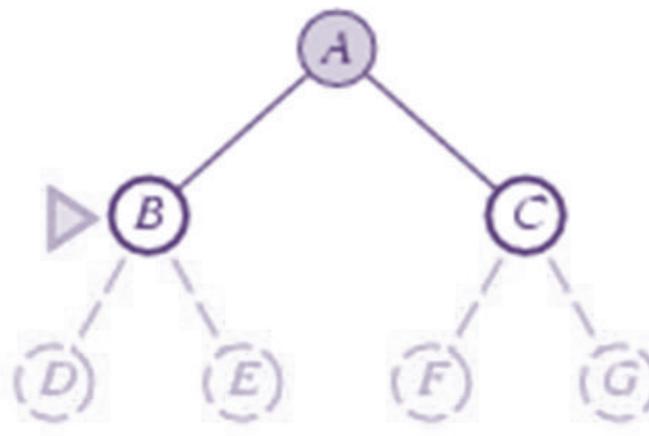
Como medir o desempenho do algoritmo usado na resolução do problema?

- **Completude:** o algoritmo garante encontrar uma solução quando ela existir?
- **Otimização:** a estratégia encontra a melhor solução? (solução ótima)
- **Complexidade de tempo:** quanto tempo o algoritmo leva para encontrar uma solução?
- **Complexidade de espaço:** quanta memória é necessária para executar a busca?

- **Busca cega ou busca sem informação:** estratégias que não usam informação adicional sobre os estados além das fornecidas na definição do problema – só gera sucessores e distingue um estado objetivo dos outros.
  - Busca em extensão
  - Busca de custo uniforme
  - Busca em profundidade
  - Busca em profundidade limitada
  - Busca de aprofundamento iterativo
- **Busca com informação e heurística:** estratégias que sabem se um estado não-objetivo é mais promissor do que outro.

- 1. Definições e exemplos***
- 2. Buscas sem informação***
- 3. Buscas com informação***
- 4. Busca local***
- 5. Algoritmos genéticos***

- **Busca em extensão:** a cada passo, são expandidos os nós para seus sucessores.



- **Busca de custo uniforme:** expande-se o nó n com o caminho de custo mais baixo.
  - Se todos os “custos de passos” forem iguais, essa busca é igual à busca em extensão.

- **Busca em profundidade:** expande o nó mais profundo na borda da árvore que ainda não foi expandido.



- **Busca em profundidade limitada:** nós na profundidade  $p$  são tratados como se não tivessem sucessores.
  - Resolve o problema de caminhos infinitos, mas pode não chegar na solução.
- **Busca em profundidade, com aprofundamento interativo:** aumenta gradualmente o limite, até alcançar o objetivo.

- 1. Definições e exemplos***
- 2. Buscas sem informação***
- 3. Buscas com informação***
- 4. Busca local***
- 5. Algoritmos genéticos***

- **Busca com informação:** usa conhecimento específico sobre o problema, além da definição do problema.
- **Busca heurística ou busca pela melhor escolha:** a seleção do nó a ser expandido é feita com base em uma avaliação do nó, aplicando uma função de avaliação  $f(n)$ .

A **função de avaliação**, em geral, mede de alguma forma a **distância** do nó  $n$  até o objetivo (por exemplo, usando algum tipo de custo). Assim, o **nó selecionado** é o com avaliação de **valor mais baixo**.

Cuidado “busca pela melhor escolha” não é uma marcha direta ao objetivo, é a escolha do nó que PARECE ser o melhor de acordo com a função de avaliação!

- **Busca gulosa:** guiada pela melhor escolha, expande o nó mais próximo à meta, supondo que isso provavelmente levará a uma solução rápida e eficiente.

Avalia os nós usando apenas a **função heurística**:  $f(n) = h(n)$ .

## Função heurística $h(n)$ :

- Custo **estimado** do caminho mais econômico do nó  $n$  até um nó objetivo.
- Avaliação do próprio nó como uma solução para o problema.
- Função arbitrária específica para um problema, com uma restrição:  
Para um nó objetivo,  $h(n) = 0$ .

Voltando ao problema de chegar a Bucareste...

**Heurística:** distância em linha reta até Bucareste –  $H_{DLR}$

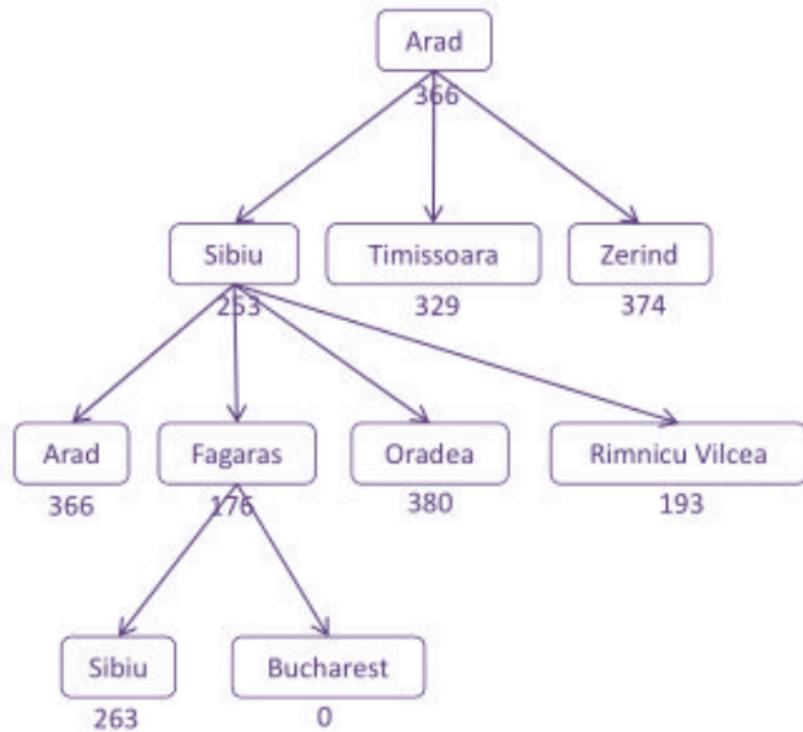
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374
Hirsova	151	Urziceni	80

Os valores  $H_{DLR}$  não fazem parte dos dados na definição do problema!

É necessário experiência no **contexto** do problema para saber que  $H_{DLR}$  é uma heurística útil por estar relacionada com distâncias reais.

Voltando ao problema de chegar a Bucareste...

Heurística: distância em linha reta até Bucareste –  $H_{DLR}$



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374
Hirsova	151	Urziceni	80

A heurística não é ótima: o Caminho passando por Sibiu e Fagaras é 32 quilômetros mais longo que o caminho por Rimmicu Vilcea e Pitesti.

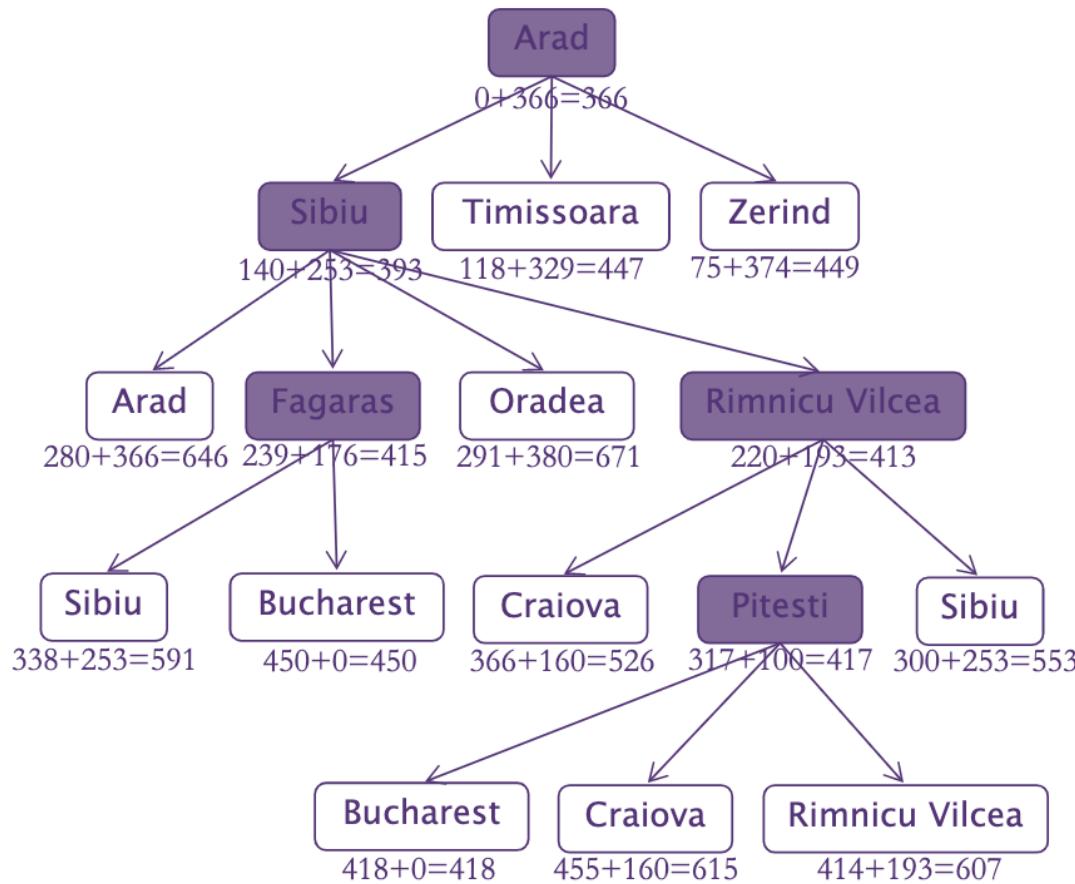
- **Busca A\***: também guiada pela melhor escolha, mas minimiza o custo total estimado da solução.
  - Função de avaliação  $f(n) = g(n) + h(n)$ .
  - $g(n)$  é o custo para alcançar cada nó e  $h(n)$  é o custo para ir do nó atual até o objetivo.
  - $h(n)$  deve ser admissível: nunca superestimar o custo para alcançar o objetivo.
  - Permite voltar uma decisão.

# BUSCAS – Com informação

FIAP

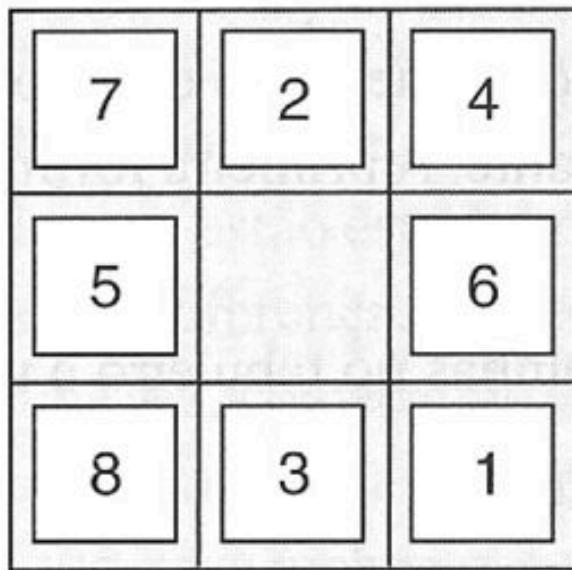
Voltando ao problema de chegar a Bucareste...

Heurística: distância em linha reta até Bucareste –  $H_{DLR}$

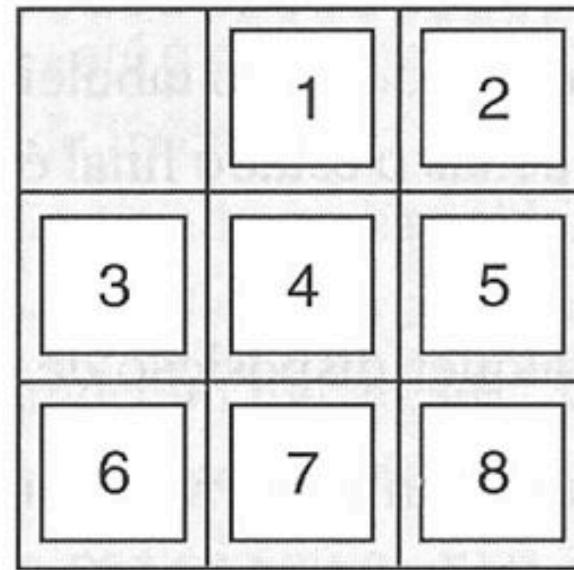


# BUSCAS – Com informação

Para o problema do **quebra-cabeças de 8 peças**, qual poderia ser a **heurística** adotada?



Estado inicial

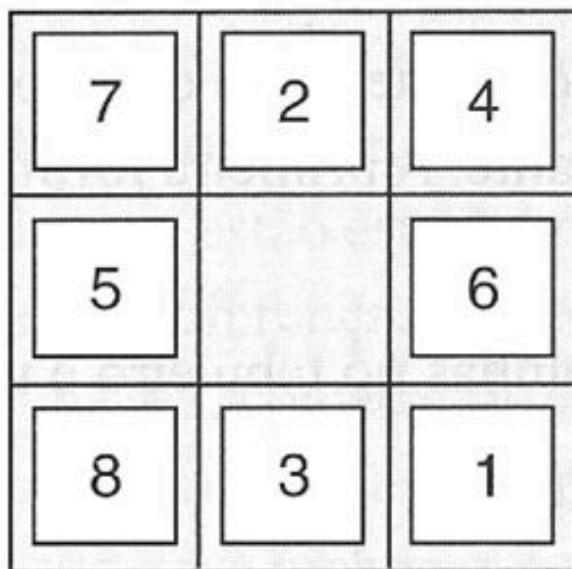


Estado objetivo

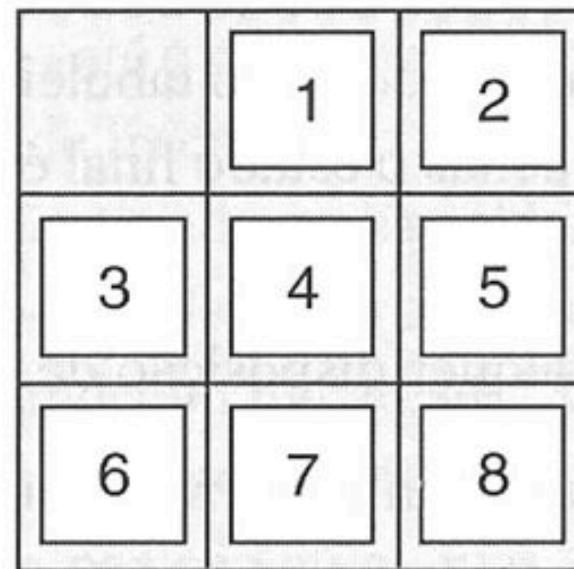
Para usarmos a busca A\*, precisamos definir uma função heurística que **nunca superestime o número de passos até o objetivo** – pode envolver o *relaxamento das regras*

# BUSCAS – Com informação

Para o problema do **quebra-cabeças de 8 peças**, qual poderia ser a **heurística** adotada?



Estado inicial



Estado objetivo

**Uma heurística possível:** retirar todos os números que estão em local errado e só colocá-los no local correto.

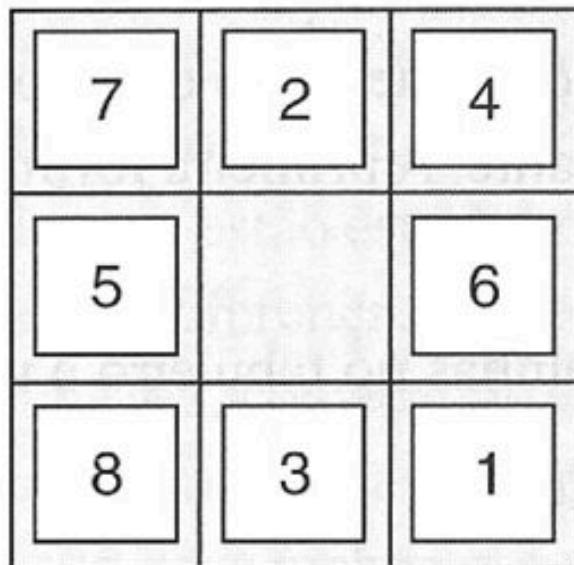


Relaxamos a regra do problema e certamente, seguindo a regra, não haverá forma de solução menos custosa.

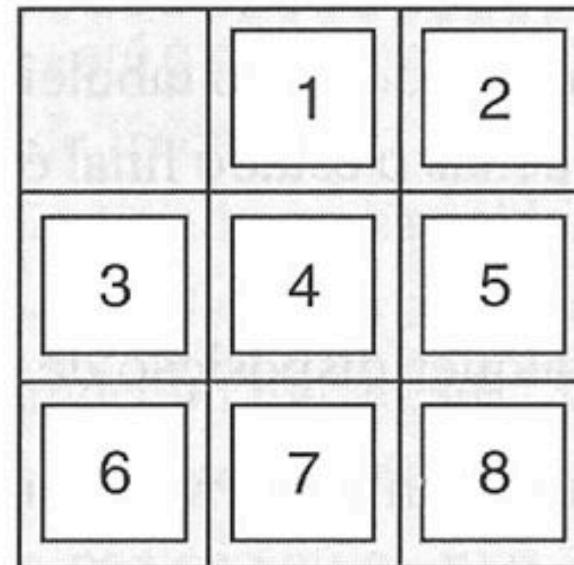
# BUSCAS – Com informação

No vídeo abaixo, é ilustrada a aplicação da busca gulosa para encontrar a solução para esse problema:

<https://www.youtube.com/watch?v=e6njY7rVe3E>



Estado inicial

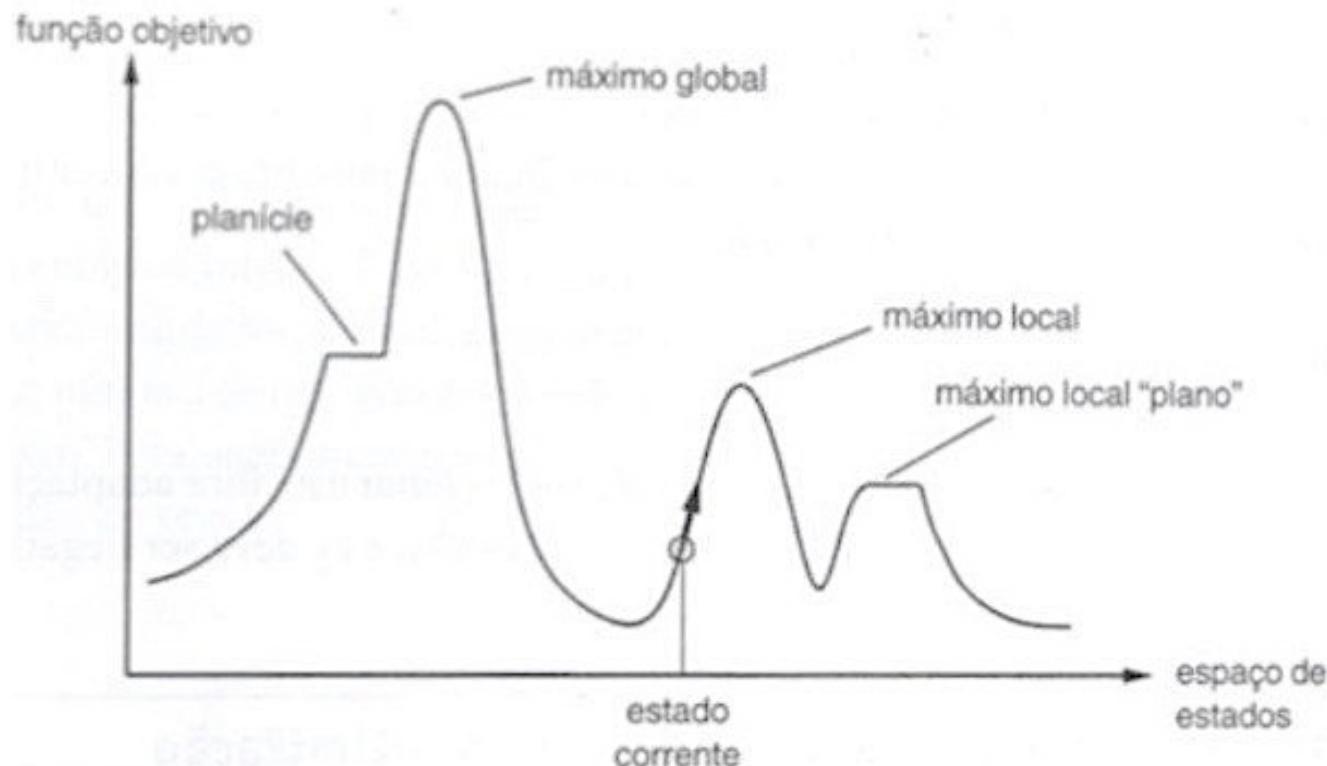


Estado objetivo

- 1. Definições e exemplos***
- 2. Buscas sem informação***
- 3. Buscas com informação***
- 4. Busca local***
- 5. Algoritmos genéticos***

# BUSCAS – Busca Local

- **Busca Local:** opera usando um único estado corrente e em geral se move apenas para os vizinhos desse estado.
  - Normalmente, os caminhos seguidos não são guardados.
  - Podemos considerar essa classe de algoritmos quando não precisamos guardar o caminho para chegar ao nó objetivo.



- **Algoritmo subida de encosta** (*hill climbing*): laço para mover continuamente no sentido do valor crescente (encosta acima).

função SUBIDA-DE-ENCOSTA (problema) retorna um estado que é um máximo local

  entradas: problema //um problema

  variáveis locais: *corrente*, vizinho // nós

*corrente*  $\leftarrow$  CRIAR-NÓ (ESTADO-INICIAL[problema]);

  repita

*vizinho*  $\leftarrow$  um sucessor de *corrente*;

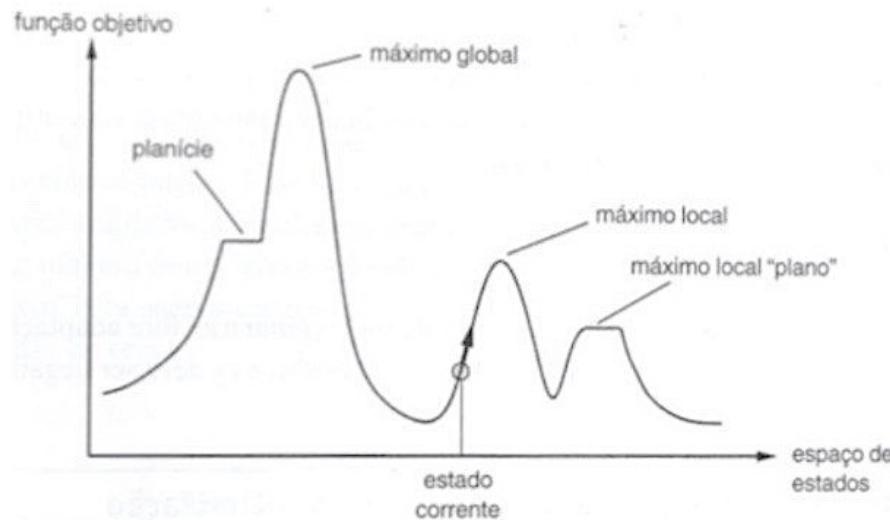
    se VALOR[i]  $\leq$  VALOR[i]

      então retornar ESTADO[i]; // possibilidade de término

    senão *corrente*  $\leftarrow$  *vizinho*;

- \* Problemas de minimização: invertemos o raciocínio para buscar valores menores do que o atual.

- **Algoritmo subida de encosta (*hill climbing*):** geralmente, queremos parar a busca quando encontramos o máximo global, mas o algoritmo também é encerrado quando encontra:
  - Máximo local: é o pico mais alto entre os estados vizinhos, embora seja mais baixo que o máximo global.
  - Picos: sequência de máximos locais que torna difícil a navegação.
  - Platôs: área da topologia do espaço de estados na qual a função de avaliação é plana (planície ou máximo local plano).



- **Algoritmo Simulated Annealing:** combina subida de encosta com percurso aleatório e permite pioras no resultado.

## Modificação no algoritmo de subida de encosta:

- Melhor movimento a cada passo → escolhe-se um movimento aleatório:
  - se o movimento **melhorar a situação**, ele será **aceito**;
  - se o movimento **não melhorar a situação**, ele será aceito com **probabilidade menor que  $l$** .
- A **probabilidade  $l$**  diminui exponencialmente de acordo com a má qualidade do movimento.
- A **probabilidade  $l$**  também diminui à medida que a **temperatura  $T$**  se reduz (movimentos “ruins” com maior probabilidade de aceite no início).

- **Algoritmo Simulated Annealing:**

- Annealing = Recozimento.
- Recozimento: processo térmico para obter estados de baixa energia de um sólido.
  - Na fase líquida (alta temperatura), o arranjo das partículas é aleatório.
  - Ao diminuir a temperatura cuidadosamente, as partículas se arranjam no estado de mínima energia do sólido.



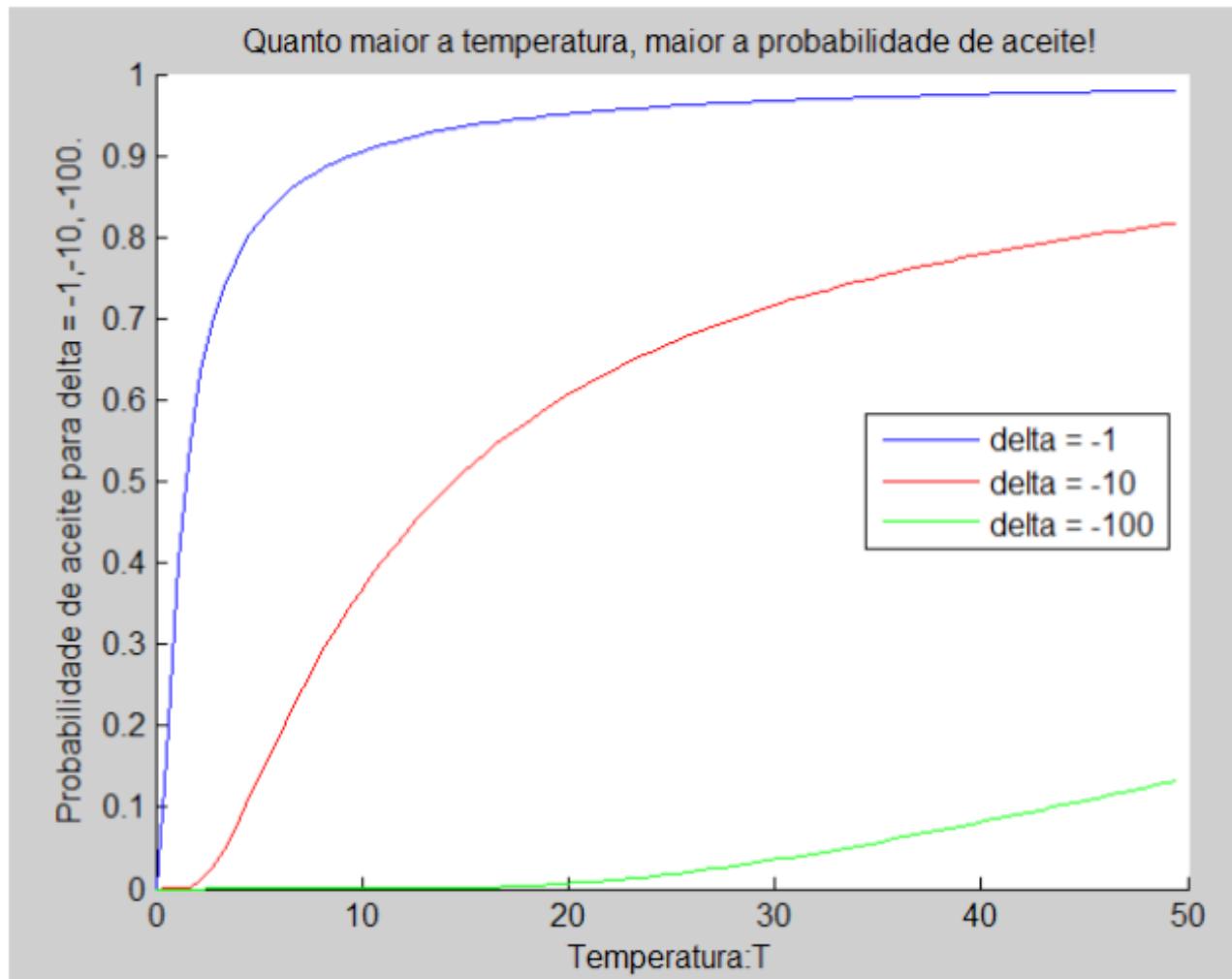
O custo de solução (valor da função objetivo) é equivalente à energia de um sólido.

- **Algoritmo Simulated Annealing:**

```
função TEMPERA-SIMULADA (problema, escalonamento) retorna um estado solução
    entradas: problema // um problema
    escalonamento // um mapeamento de tempo para “temperatura”
    variáveis locais: corrente, próximo // dois nós
    T // uma “temperatura” que controla a probabilidade de passos descendentes

    corrente ← CRIAR-NÓ (ESTADO-INICIAL[problema]);
    Para t ← 1 até ∞ faça
        T ← escalonamento[t]; // um looping
        se T = 0 então retorna corrente; // término
        próximo ← um sucessor de corrente selecionado ao acaso;
        ΔE ← VALOR[próximo] – VALOR[corrente];
        se ΔE > 0 então corrente ← proximo;
        senão corrente ← próximo somente com probabilidade  $e^{\Delta E/T}$ ;
```

- **Algoritmo Simulated Annealing:**
  - Estudo da probabilidade de aceitação de um movimento de piora.



- **Algoritmo Simulated Annealing:**

- Como determinar a temperatura? Duas sugestões:

- 1. Selecionar uma solução aleatória;
    - 2. Calcular o custo de várias soluções da vizinhança dessa solução;
    - 3. Tomar como temperatura inicial o maior custo obtido no passo anterior.

- 1. Selecionar uma solução aleatória;
    - 2. Gerar um conjunto de soluções vizinhas dessa solução;
    - 3. Iniciar com uma temperatura baixa e contar quantas soluções vizinhas seriam aceitas com essa temperatura;
    - 4. Aumentar a temperatura de forma a aceitar uma quantidade “grande” de vizinhos (tão grande quanto fizer sentido no contexto).

- 1. Definições e exemplos***
- 2. Buscas sem informação***
- 3. Buscas com informação***
- 4. Busca local***
- 5. Algoritmos genéticos***

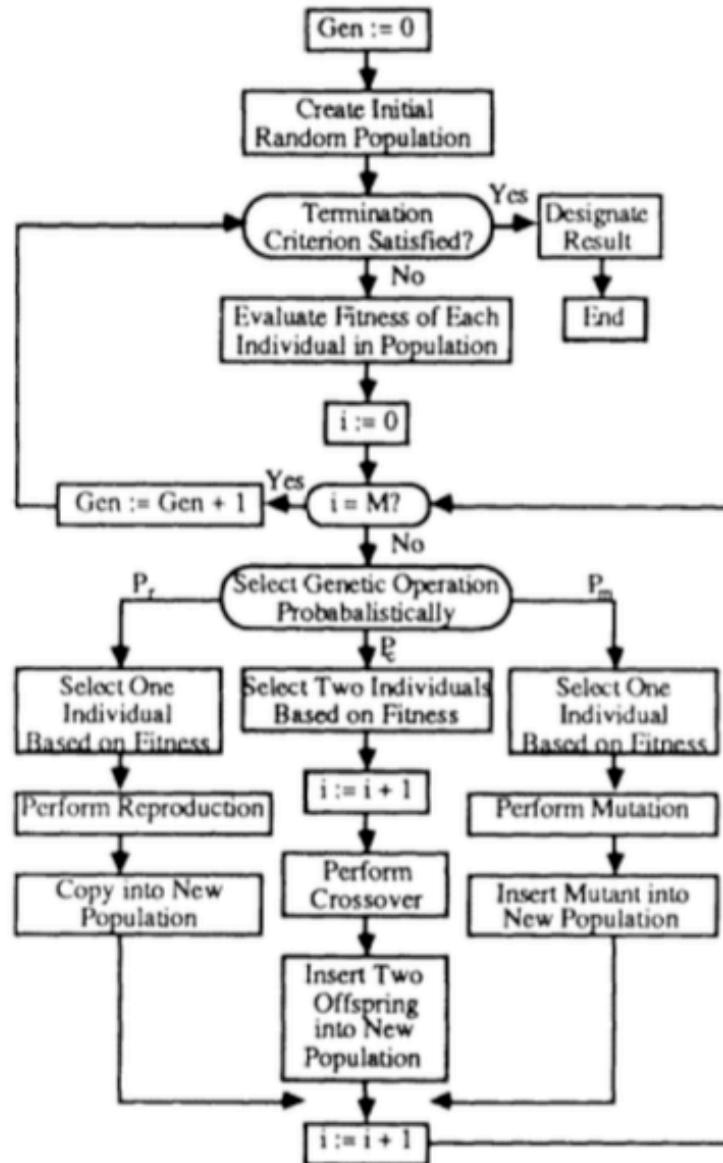
- **Algoritmo genético:** algoritmo matemático paralelo que transforma um conjunto (população) de objetos matemáticos individuais (tipicamente uma string de caracteres de tamanho fixo – indivíduos representados por **cromossomo**), cada um associado a um valor de **fitness**, em uma nova população usando operações baseadas no princípio Darwiniano de reprodução (operações genéticas) e sobrevivência do mais bem adaptado.

## Processo evolutivo natural (base do algoritmo genético):

- Uma entidade tem a habilidade de se reproduzir
- Existe uma população formada por essas entidades
- Existe alguma diversidade entre essas entidades
- Algumas diferenças na habilidade de sobrevivência nesse ambiente está associada com a diversidade entre as entidades.

# BUSCAS – Algoritmo Genético

FIAP

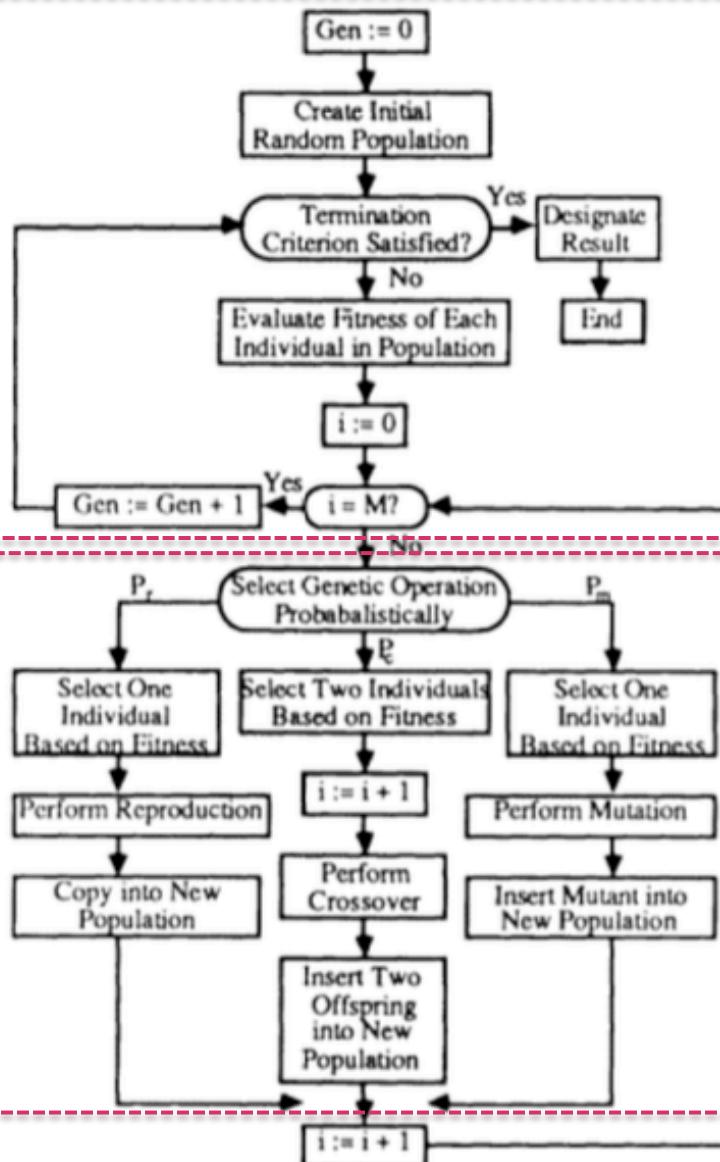


Algoritmo básico retirado de Koza (1992)

# BUSCAS – Algoritmo Genético

FIAP

Parte 1



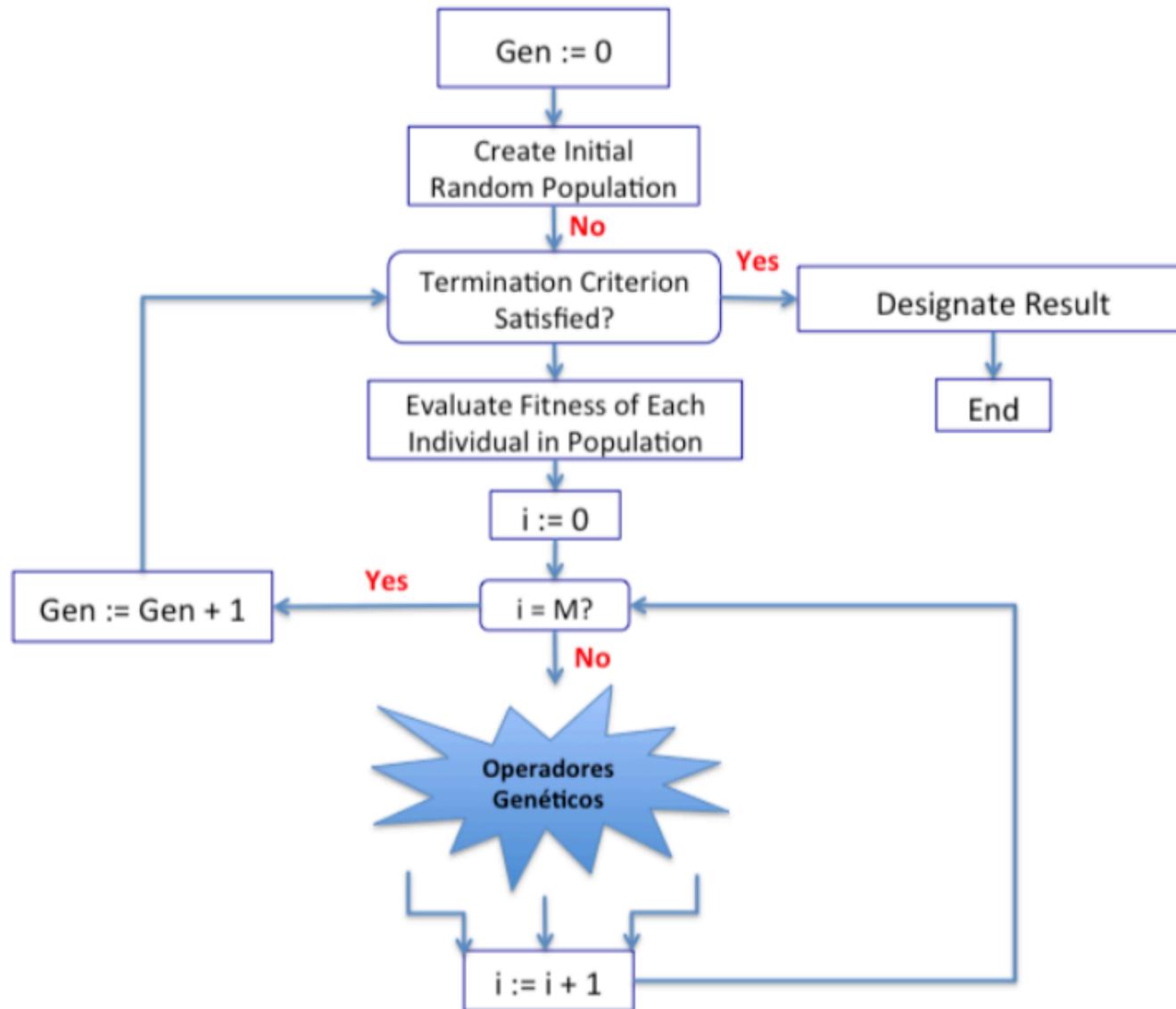
Parte 2

Algoritmo básico retirado de Koza (1992)

# BUSCAS – Algoritmo Genético

FIAP

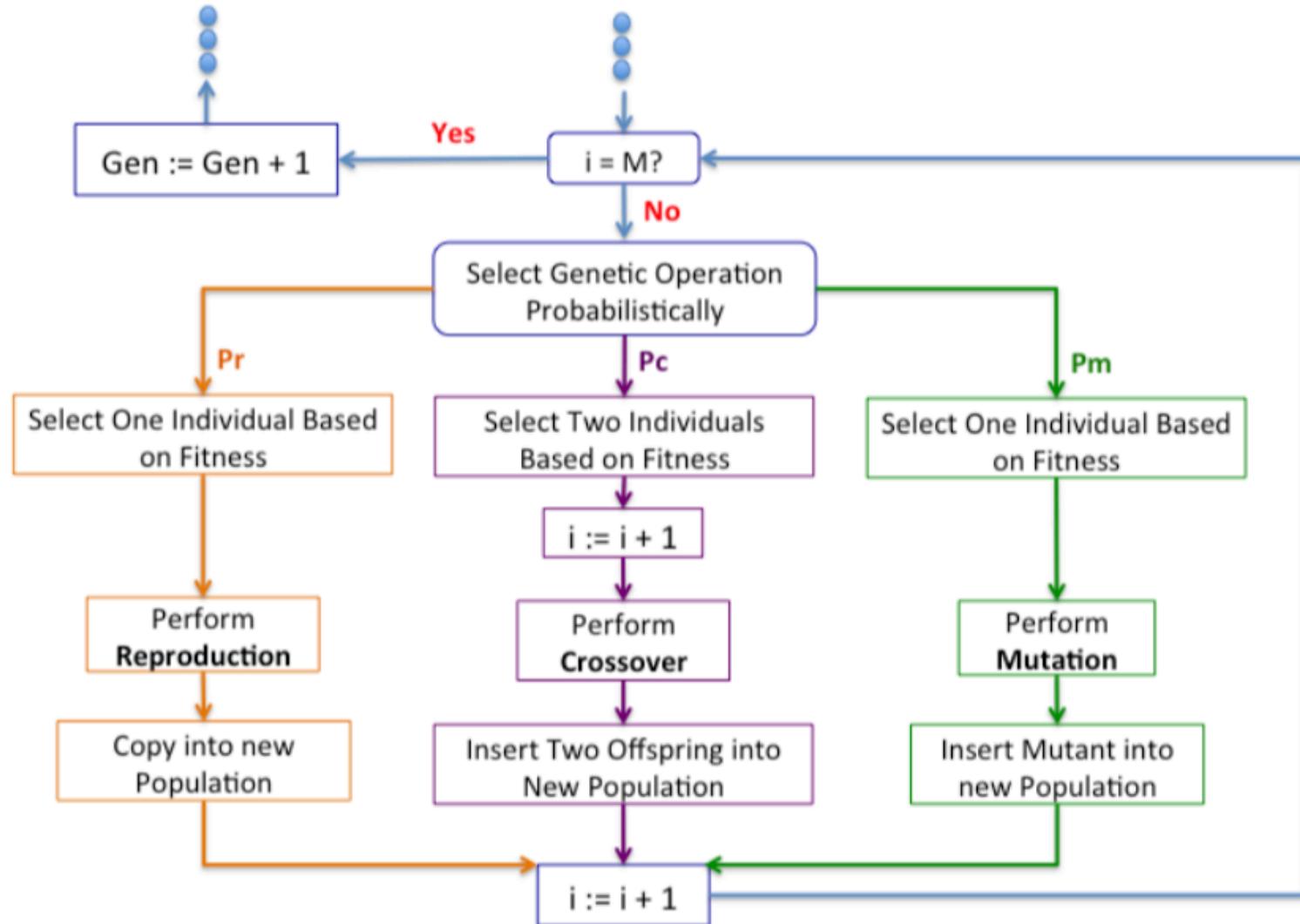
Parte 1 aproximada:



# BUSCAS – Algoritmo Genético

FIAP

Parte 2 aproximada - operadores genéticos:



- **Vamos trabalhar com um exemplo:**

## Restaurante Hambúrguer

- Problema: encontrar a melhor estratégia de negócio para uma rede de 4 restaurantes.
- Estratégias baseadas em 3 decisões binárias:
  - Preço – 50 centavos ou 10 dólares?
  - Bebida – vinho ou cola?
  - Serviço – lento e calmo, com garçons vestidos com ternos, ou rápido e divertido, com garçons vestidos com uniforme de poliéster branco?
- Meta: encontrar a combinação das decisões estratégicas que maximize o lucro dos restaurantes.

# BUSCAS – Algoritmo Genético

- **Representação dos restaurantes – cromossomo:** string de comprimento  $L = 3$  com alfabeto de tamanho  $K = 2$ .

	Preço	Bebida	Serviço
<b>Cromossomo:</b>	Alto = 0 Baixo = 1	Vinho = 0 Cola = 1	Lento = 0 Rápido = 1

- Para cada decisão de estratégia, o valor 0 ou 1 é associado à posição correspondente no cromossomo;
- Espaço de busca:  $2^3 = 8$  possíveis estratégias de negócio.

Representação dos 4 restaurantes da rede (população):

Id	Preço	Bebida	Serviço	Cromossomo
1	0 (alto)	1 (cola)	1 (rápido)	011
2	0 (alto)	0 (vinho)	1 (rápido)	001
3	1 (baixo)	1 (cola)	0 (lento)	110
4	0 (alto)	1 (cola)	0 (lento)	010

Cada linha:  
um indivíduo  
(um restaurante)

- **Função fitness:** a avaliação dos indivíduos (cromossomo que representa um restaurante) será o valor decimal equivalente à string binária.

Id	Cromossomo	Fitness
1	011 4 2 1	3
2	001	1
3	110	6
4	010	2
Total		12
Pior		1
Média		3
Melhor		6



Valores para  
a população  
dessa geração!

- **Operadores genéticos básicos:**

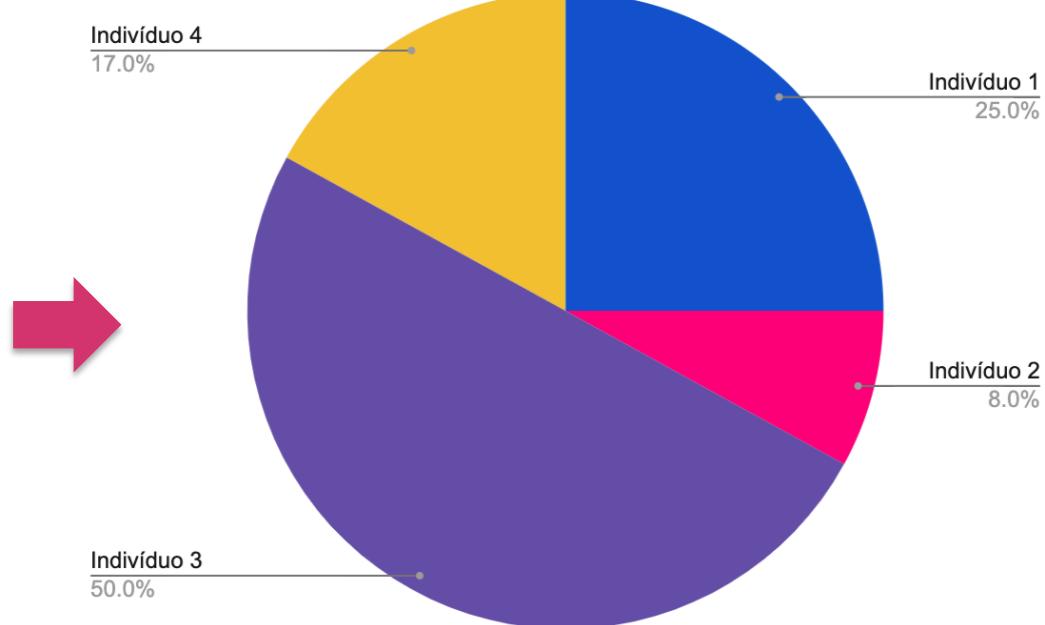
- **seleção:** seleciona um indivíduo da população de acordo com uma probabilidade relacionada ao valor de fitness, para sofrer a aplicação dos outros operadores;
- **reprodução:** realiza uma cópia do indivíduo na população atual e o coloca na próxima população (geração);
- **crossover:** também denominado como *operador de recombinação sexuada*, realiza a combinação genética de dois indivíduos criando indivíduos descendentes;
- **mutação:** realiza uma modificação aleatória em um ou mais genes do indivíduo.

- **Funcionamento do operador de seleção - roleta:**
  1. calcule o fitness total da população;
  2. calcule a distribuição do fitness dos indivíduos considerando o fitness total da população;
  3. construa a roleta atribuindo uma parte dela para cada indivíduo considerando a sua probabilidade (fitness) de sobrevivência na população;
  4. sorteie um número de 1 a 100;
  5. posicione o número sorteado na roleta e selecione o indivíduo que ocupa essa parte da roleta.

# BUSCAS – Algoritmo Genético

- Funcionamento do operador de seleção - roleta:

Id	Cromossomo	Fitness	%fitness
1	011	3	$3/12 = 0,25$
2	001	1	$1/12 = 0,08$
3	110	6	$6/12 = 0,5$
4	010	2	$2/12 = 0,17$
Total		12	1,0



\* Se o número sorteado entre 0 e 100 estiver no intervalo de 33 a 83, o indivíduo selecionado será o 3.

- Seleção + reprodução:**

## Exemplo de seleção + reprodução

- Considere a geração 0 na tabela abaixo.
- Se selecionarmos o indivíduo 3 e aplicarmos o operador de reprodução:

<b>Id</b>	<b>Cromossomo</b>	<b>Fitness</b>
1	011	3
2	001	1
<b>3</b>	<b>110</b>	<b>6</b>
4	010	2
<b>Total</b>		<b>12</b>
<b>Média</b>		<b>3</b>
<b>Melhor</b>		<b>6</b>
<b>Pior</b>		<b>1</b>



<b>Id</b>	<b>Cromossomo</b>	<b>Fitness</b>
1	011	3
<b>2</b>	<b>110</b>	<b>6</b>
<b>3</b>	<b>110</b>	<b>6</b>
4	010	2
<b>Total</b>		<b>17</b>
<b>Média</b>		<b>4.25</b>
<b>Melhor</b>		<b>6</b>
<b>Pior</b>		<b>2</b>

- **Operador de crossover:** novos indivíduos são criados a partir dos “pais” (indivíduos selecionados para sofrer o crossover).
  - Na sua forma básica, a partir do material genético dos pais, são produzidos 2 novos indivíduos (descendentes).
  - Os descendentes criados possuem material genético dos pais, mas são (preferencialmente) diferentes dos pais e diferentes entre si.
- **Funcionamento do operador de crossover:**
  1. Selecione dois indivíduos com o operador de seleção.
  2. Selecione um ponto de crossover: escolha um número entre 1 e  $L-1$  aleatoriamente.
  3. Separe cada um dos cromossomos pais em duas partes, de acordo com o ponto de crossover.
  4. Combine as partes dos pais criando os dois novos cromossomos.

# BUSCAS – Algoritmo Genético

- Funcionamento do operador de crossover de um ponto:

## Exemplo de seleção + crossover

	Id	Cromossomo	Fitness	%fitness
PAI 1	1	011	3	$3/12 = 0,25$
	2	001	1	$1/12 = 0,08$
PAI 2	3	110	6	$6/12 = 0,5$
	4	010	2	$2/12 = 0,17$
Total		12	1,0	

Ponto de crossover sorteado:  $L = 2$

Fragments PAI 1 (01|1): **01** \_ e \_ **1**  
Fragments PAI 2 (11|0): **11** \_ e \_ **0**

Filho 1 = **010**  
Filho 2 = **111**

\* Considere que há uma **probabilidade** de ocorrência de crossover!  
Ou seja, nesse exemplo, a probabilidade de ocorrência de crossover levou a somente um par de indivíduos sofrer crossover.

- **Operador mutação:** opera sobre um único indivíduo e sua função é inserir diversidade genética na população.
  - Deve ser usado com moderação!
- **Funcionamento do operador de mutação:**
  1. Selecione um indivíduo na população.
  2. Selecione o ponto de mutação: escolha um número entre 1 e  $L$  aleatoriamente.
  3. Escolha um caracter do alfabeto aleatoriamente e insira no ponto de mutação selecionado.

# BUSCAS – Algoritmo Genético

- **Escolha dos indivíduos que farão parte da próxima geração**
  - Mantendo a mesma quantidade de indivíduos em todas as gerações, podemos, por exemplo, excluir da próxima geração os indivíduos com menores valores de fitness:

Geração 0

Cromossomo	Fitness
011	3
001	1
110	6
010	2

Reprodução no 2º  
+  
Crossover com 1º e 3º



Geração 1

Cromossomo	Fitness
011	3
110	6
110	6
111	7

Seleção dos maiores fitness

Cromossomo	Fitness
011	3
001	1
110	6
010	2
110	6
010	2
111	7

Gerado pela reprodução

Gerados pelo crossover

- **Critérios de parada do algoritmo genético:**
  - Número máximo de gerações.
  - Convergência da população.
  - Alcance da solução ótimo (se esse valor de fitness for conhecido – no exemplo dos restaurantes, sabemos que é 7).  
o valor que representará preço a 50 centavos, bebida cola e serviço rápido
- **Adaptação de parâmetros:**
  - Determinística: são aplicadas mudanças nos parâmetros seguindo alguma regra. Exemplo: aumentar a probabilidade de mutação em 0,01% a cada geração.
  - Adaptativa: é usado algum feedback da execução do algoritmo para determinar o valor do parâmetro. Exemplo: se mais de 1/5 das mutações forem bem sucedidas, aumenta-se a taxa de mutação, caso contrário, diminui-se.
  - Auto-adaptativa: codifica os parâmetros dentro do cromossomo e deixa-os evoluir junto com a solução.

- **Variações de operadores:**

- Seleção por torneio: o torneio seleciona o melhor (mais bem adaptado, maior fitness) entre  $k$  indivíduos aleatoriamente selecionados para participar.
  - Se todos os indivíduos da população participarem, o melhor indivíduo da população é selecionado.
  - O pior indivíduo da população só será selecionado se  $k = 1$  e ele for o sorteado a participar.
- Seleção por ranking: antes de executar a roleta, ordena os indivíduos de acordo com seu valor fitness e aplica um procedimento para minimizar as diferenças entre seus valores. Com a minimização das diferenças, os indivíduos terão chances mais parecidas de serem selecionados.
- Seleção truncada: apenas os X% melhores indivíduos podem ser selecionados.

- **Variações de operadores:**

- Crossover de dois pontos: dois pontos de corte são selecionados; o primeiro filho recebe a parte de fora do corte do primeiro pai e a parte de dentro do corte do segundo pai e o segundo, recebe as partes restantes.
- Crossover uniforme: para cada posição, sortea-se 0 ou 1. Se o valor sorteado for 0, o filho recebe o valor que consta no primeiro pai para a posição em questão; se for 1, recebe do segundo pai.
- Crossover baseado em maioria: sorteia-se n pais e faz-se com que cada posição do filho seja igual ao valor mais comum nos pais selecionados. Acelera a convergência genética.

- Para o problema das **N rainhas** apresente:
  - Uma representação para indivíduos com cromossomo;
  - Uma função fitness;
  - Um fenótipo e seu valor fitness;
  - A aplicação de operadores evolutivos (crossover e mutação) pode gerar algum cronomosso infactível? Se sim, como você proporia tratar esses casos (operadores alterativos ou penalização na função fitness)?

O problema: dado um tabuleiro de NxN, encontrar uma distribuição de N rainhas tal que nenhuma delas esteja ameaçada por outra. Duas rainhas se ameaçam quando estão posicionadas na mesma linha, mesma diagonal ou mesma coluna.

Copyright © 2019 Profa. Thais Rodrigues Neubauer

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).