

Busca Heurísticas e Meta-Heurística

Prof. Dr. Clodoaldo Lima

Escola de Artes, Ciências e Humanidades – Universidade de São Paulo

Introdução

- ▶ **Estratégias de buscas heurísticas** utilizam algum **conhecimento específico** sobre o problema a fim de encontrar soluções de maneira eficiente.
- ▶ **Organização:**
 - ▶ Resolução de problemas por meio de busca
 - ▶ Estratégias de busca com informação (heurísticas)
 - ▶ Construção de funções heurísticas
 - ▶ Algoritmo de busca local
 - ▶ Algoritmos de busca distribuída
 - ▶ Meta-Heurística

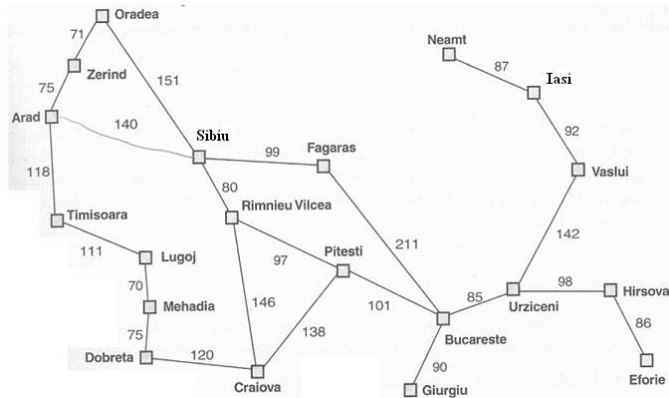


Resolução de problemas por meio de busca

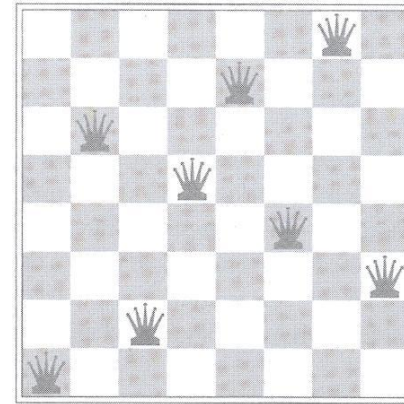
- ▶ Definição de um problema (representação):
 - ▶ Estado inicial
 - ▶ Ações possíveis
 - ▶ Estado inicial + realização de ações possíveis = espaço de estados (espaço de busca)
 - ▶ Teste de objetivo: determina se um dado estado é um estado objetivo (alcance do objetivo buscado na resolução do problema)
 - ▶ Pode ser um único objetivo (expresso de uma única forma ou de diferentes formas)
 - ▶ Ou um conjunto de vários objetivos
 - ▶ Função de custo: medida de desempenho da busca dentro do processo de resolução do problema



Exemplos de problemas



A solução é o caminho entre uma cidade e outra.



A solução é um posicionamento específico das 8 rainhas.

7	2	4
5		6
8		1

Estado inicial

	1	2
3	4	5
6	7	8

Estado objetivo

A solução é um posicionamento específico das pedras do tabuleiro.

CUSTOS DE CAMINHOS NO MAPA RODOVIÁRIO DA ROMÊNIA.

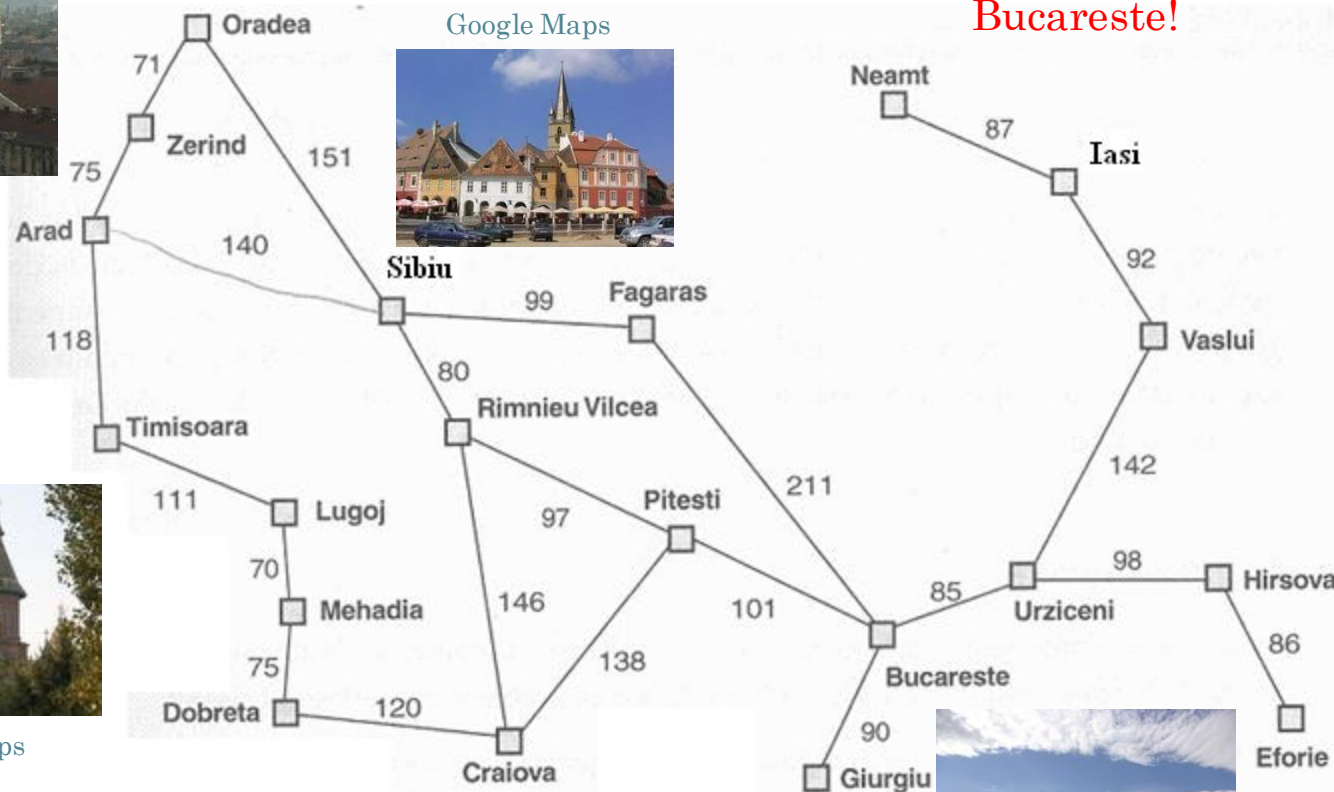
Achar o melhor caminho
para ir de Arad até
Bucareste!



tennisforum.com



Google Maps



Google Maps

Busca em árvore

função BUSCA-EM-ÁRVORE (*problema*, *estratégia*) **retorna** uma solução ou falha

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

se não existe nenhum candidato para expansão

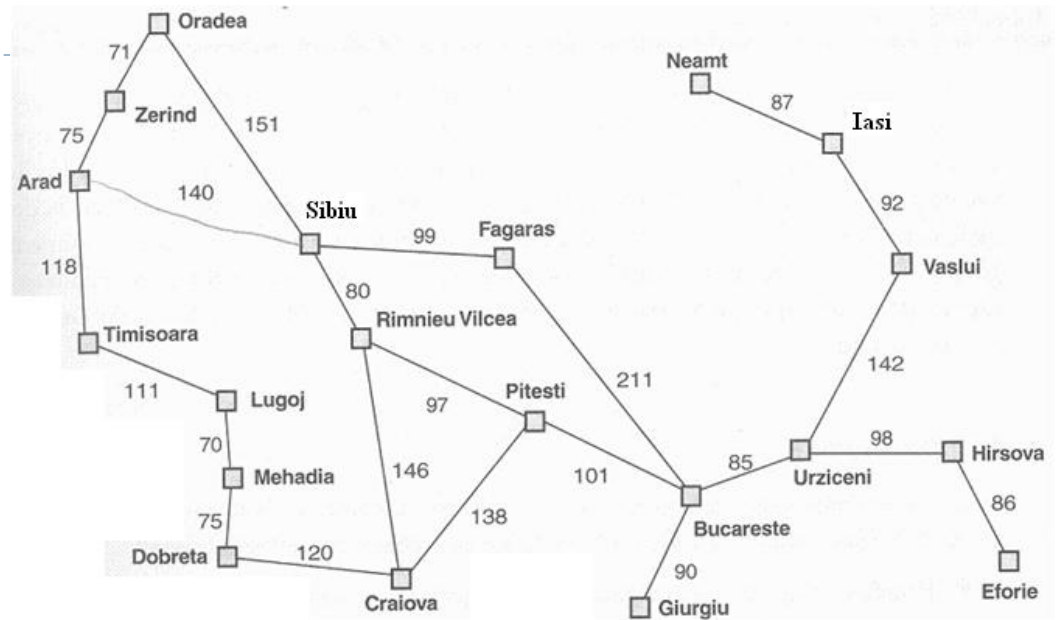
então retornar falha; // (exit)

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

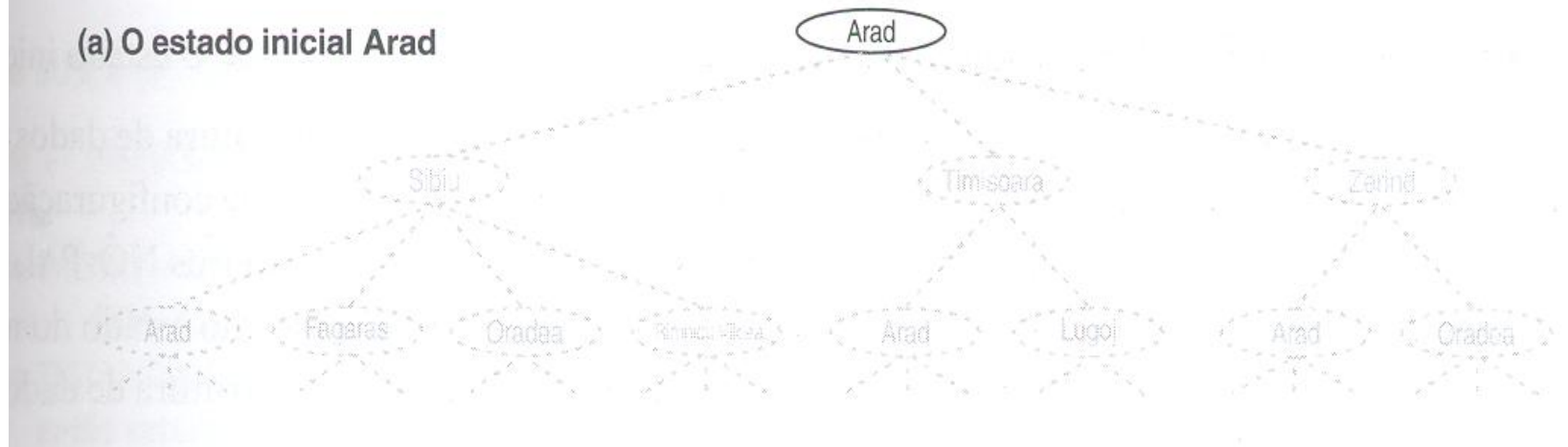
se o nó contém um *estado objetivo* então retornar a **solução**
correspondente

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

OBSERVAÇÃO: PROCESSO DE BUSCA EM ÁRVORE

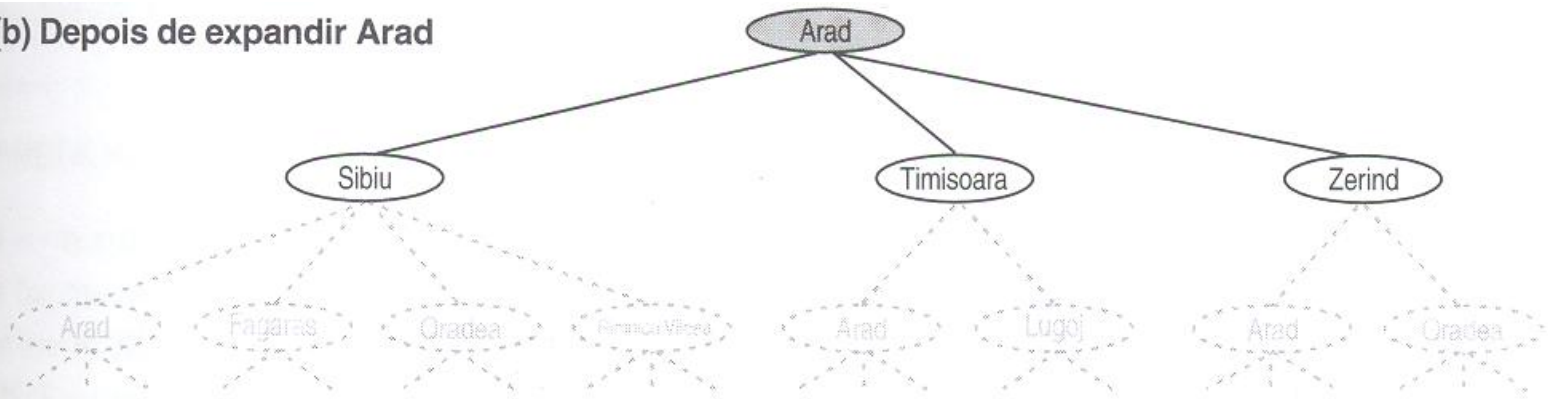


(a) O estado inicial Arad

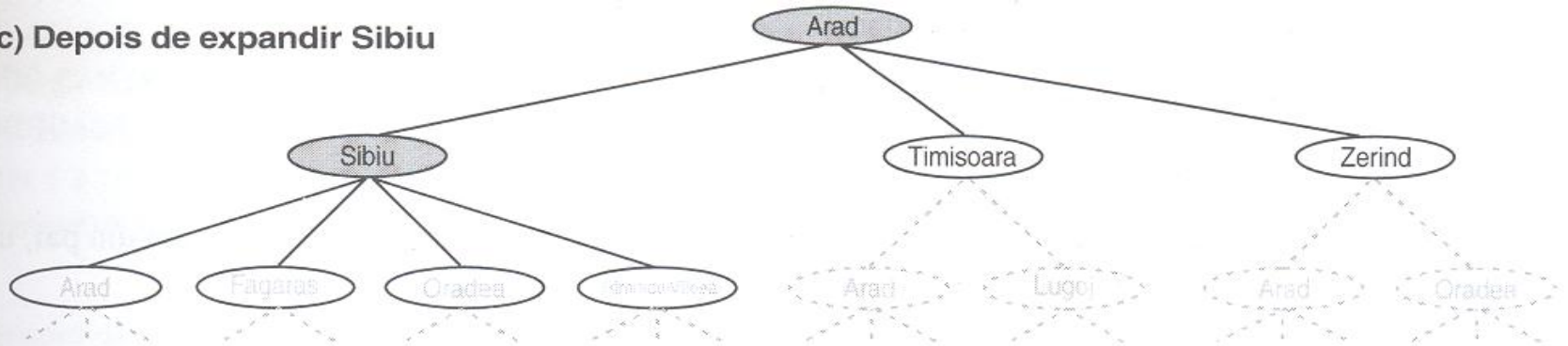


OBSERVAÇÃO: PROCESSO DE BUSCA EM ÁRVORE

(b) Depois de expandir Arad



(c) Depois de expandir Sibiu



E assim por diante

Estratégia de busca com informação (heurística)

▶ Busca com informação:

- ▶ Uma estratégia que **usa conhecimento específico** sobre um problema, além da definição do próprio problema.
- ▶ Pode encontrar soluções de forma mais eficiente que uma estratégia sem informação.

▶ Busca pela melhor escolha:

- ▶ Trata-se de um tipo de busca em árvore ou um tipo de busca em grafo no qual um nó é selecionado para expansão (ou exploração) com base em uma **função de avaliação** $f(n)$.
 - ▶ Tradicionalmente, o nó com a avaliação **mais baixa** é selecionada para expansão porque a **avaliação** mede a **distância** dele até o objetivo (de alguma forma – usando por exemplo, algum tipo de custo)



Estratégia de busca com informação (heurística)

- ▶ Cuidado com a interpretação do nome “**busca pela melhor escolha**”!!
 - ▶ Se pudéssemos realmente expandir o **melhor nó** primeiro, isso não seria uma busca e sim uma “**marcha direta**” ao objetivo.
 - ▶ O que fazemos é escolher o nó que **parece ser** o melhor de acordo com a função de avaliação.
 - ▶ Se a função de avaliação for (exatamente) precisa, esse será de fato o melhor nó!

- ▶ Componente fundamental: a FUNÇÃO HEURÍSTICA

Exemplo 1 : A estimativa do custo do caminho mais econômico de uma cidade a outra pode ser a distância em linha reta entre estas duas cidades.

Exemplo 2 : A avaliação de uma grade de distribuição de horário é a soma de condições desejáveis que ela atende, uma vez que ela atende a todas as restrições.

- ▶ Denotada por $h(n)$:
 1. Custo estimado do caminho mais econômico do nó n até um nó objetivo, ou
 2. Avaliação do próprio nó como uma solução para o problema
- ▶ Funções arbitrárias, específicas para um problema, como uma restrição: **se n é o nó objetivo então $h(n) = 0$.**
- ▶ Seu valor depende do estado (condições) do nó que lhe é passado como entrada.



Busca Gulosa (pela melhor escolha)

- ▶ Estratégia que expande o nó mais próximo à meta, na suposição de que isso provavelmente levará a uma solução rápida e eficiente.
 - ▶ Avalia os nós usando apenas a função heurística:
$$f(n) = h(n)$$
(função de avaliação = função heurística)
- ▶ Para o problema de planejamento de rotas na Romênia:
 - ▶ Heurística da distância em linha reta (h_{DLR})
 - ▶ Se o objetivo é chegar em Bucareste, é necessário conhecer as distâncias em linha reta até Bucareste.



CUSTOS DE CAMINHOS NO MAPA RODOVIÁRIO DA ROMÊNIA.

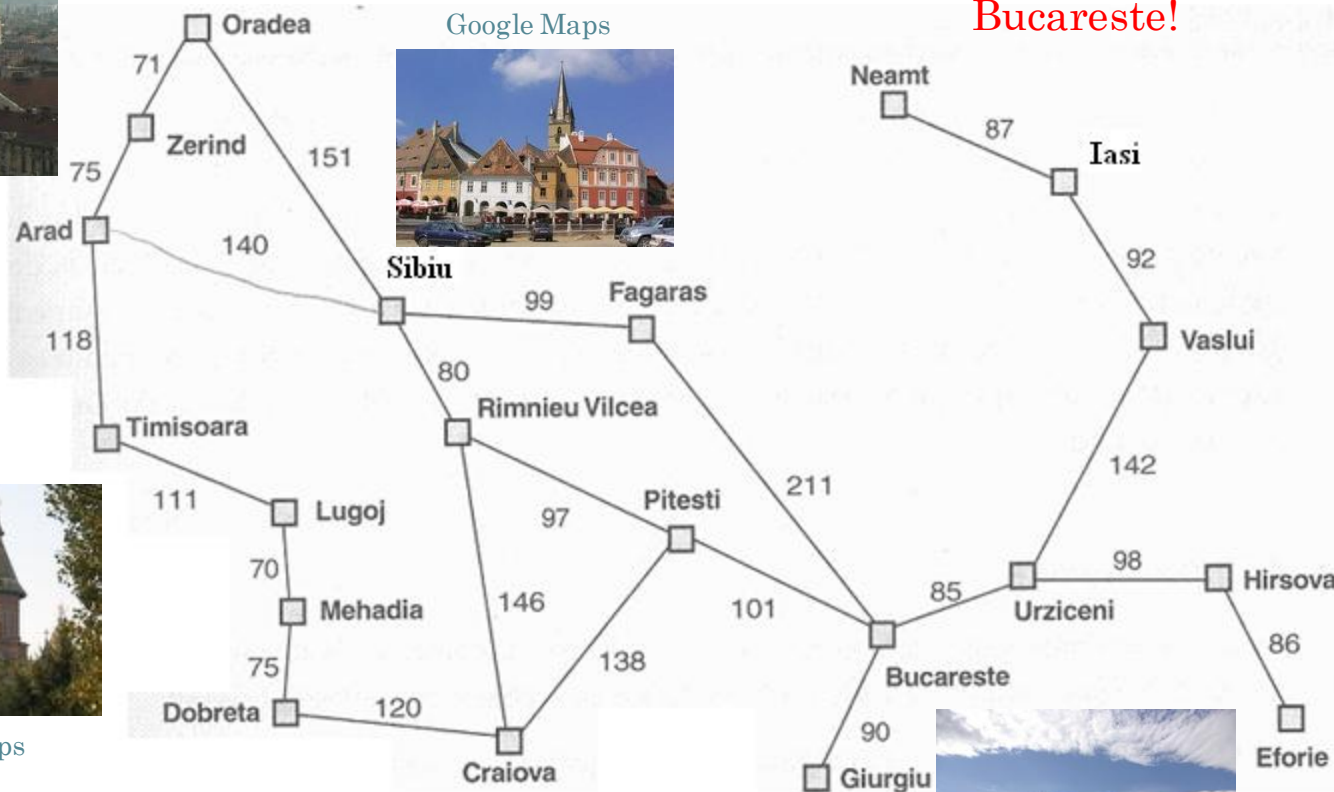
Achar o melhor caminho
para ir de Arad até
Bucareste!



tennisforum.com



Google Maps



Google Maps

VALORES DE h_{DLR}

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

Os valores h_{DLR} não são dados na definição do problema e é necessário **experiência no contexto do problema** para saber que h_{DLR} está relacionada com distâncias rodoviárias reais e que, portanto, é uma heurística útil.

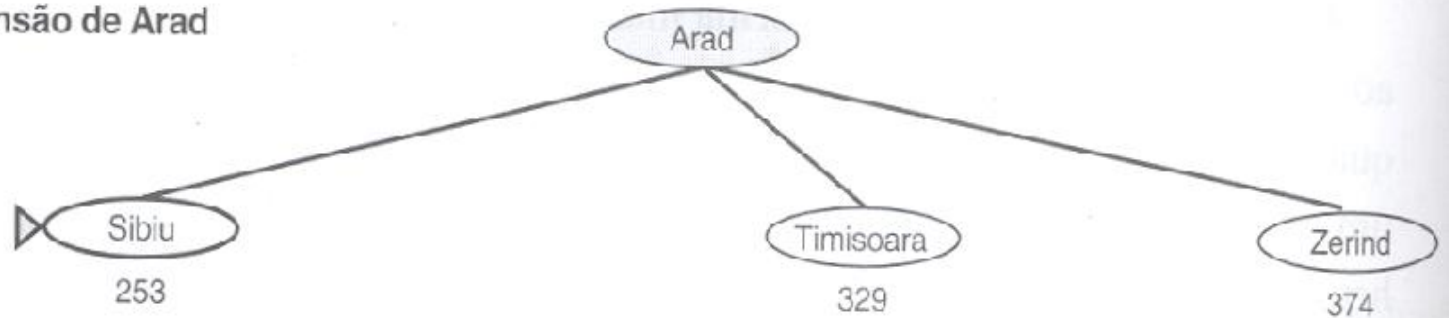


O PROCESSO EXECUTADO POR UMA BUSCA GULOSA

(a) O estado inicial

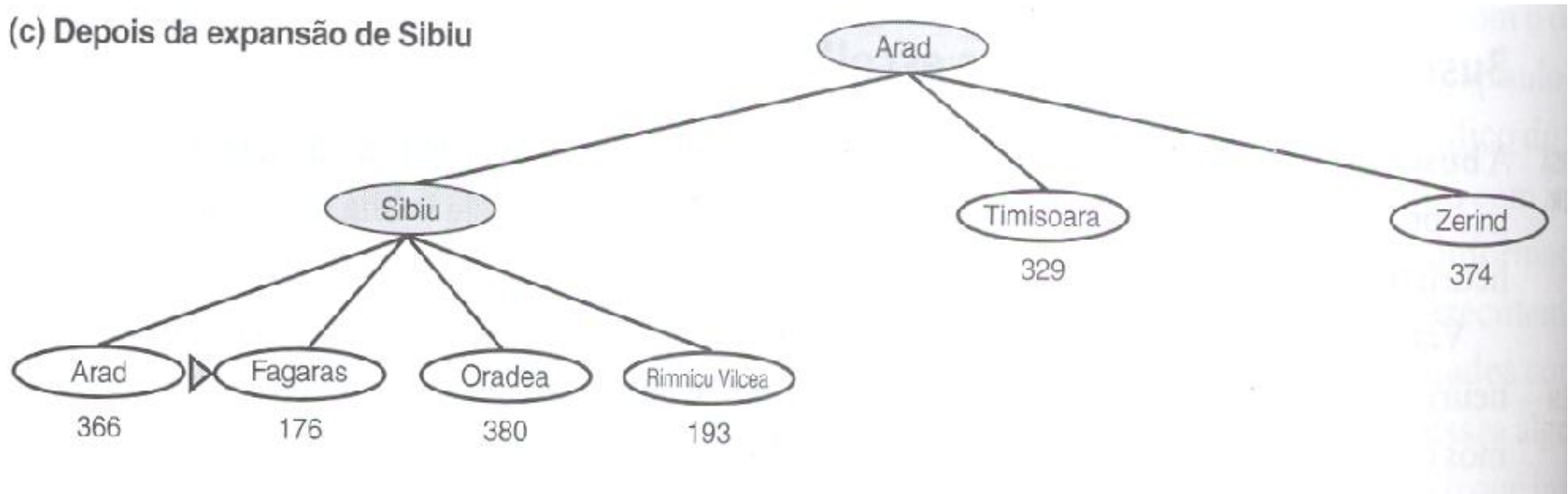


(b) Depois da expansão de Arad



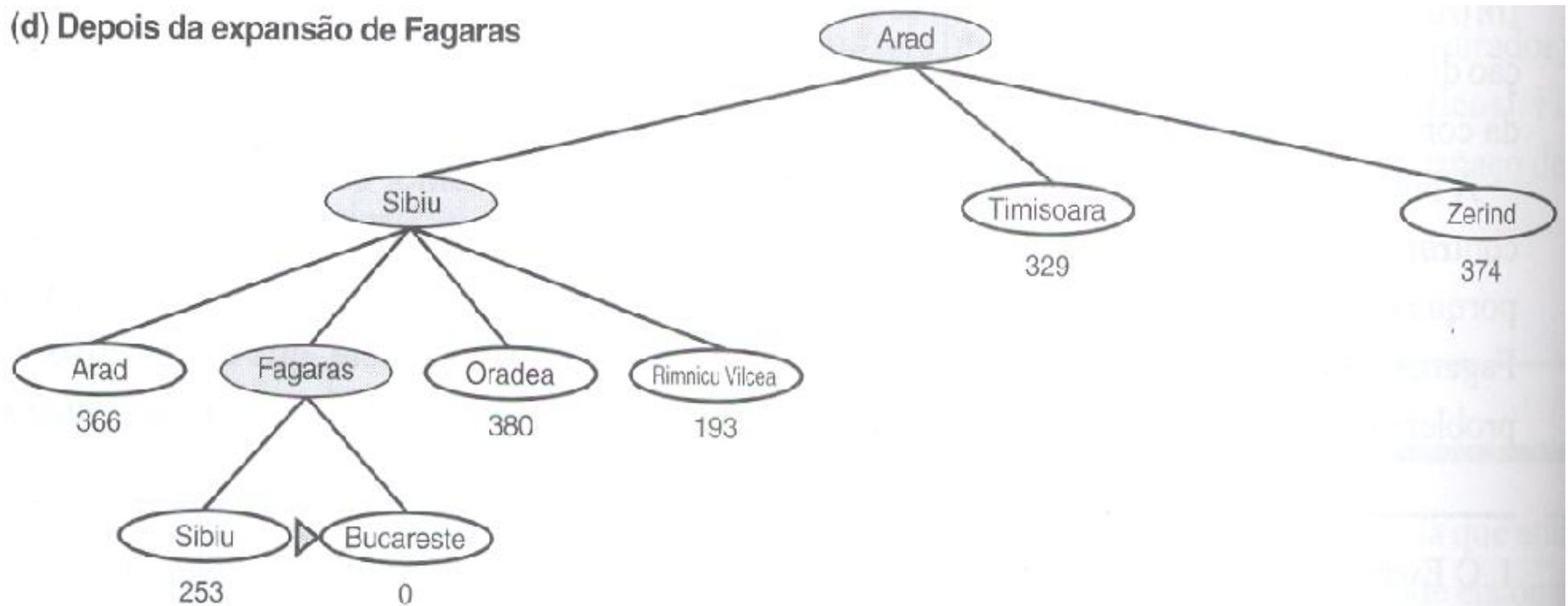
O PROCESSO EXECUTADO POR UMA BUSCA GULOSA

(c) Depois da expansão de Sibiu



O PROCESSO EXECUTADO POR UMA BUSCA GULOSA

(d) Depois da expansão de Fagaras



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

Busca A*

- ▶ É um tipo de busca pela melhor escolha que minimiza o **custo total** estimado da solução.
- ▶ Ela avalia os nós combinando:
 - ▶ $g(n)$ – o custo para alcançar cada nó
 - ▶ $h(n)$ - o custo para ir do nó até o objetivo

$$f(n) = g(n) + h(n)$$

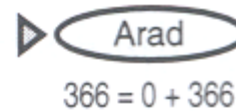
(função de avaliação = custo até o nó + custo para ir do nó até o objetivo)

- ▶ Sabendo que $g(n)$ fornece o custo do caminho desde o nó inicial até o nó n , e que $h(n)$ é o custo estimado do caminho de custo mais baixo desde n até o objetivo, tem-se
- ▶ $f(n)$ = custo **estimado** da solução de custo mais baixo passando por n .
- ▶ **A heurística deve ser admissível:**
 - ▶ $h(n)$ é admissível se ela nunca superestimar o custo para alcançar o objetivo.

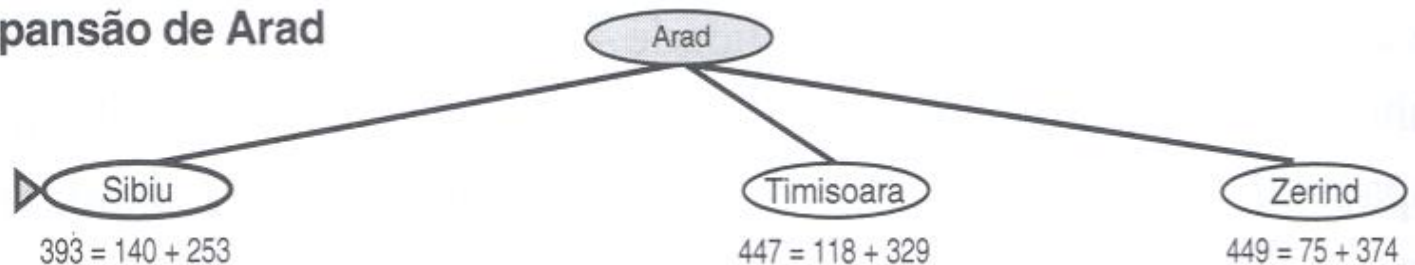


O PROCESSO EXECUTADO POR UMA BUSCA A*

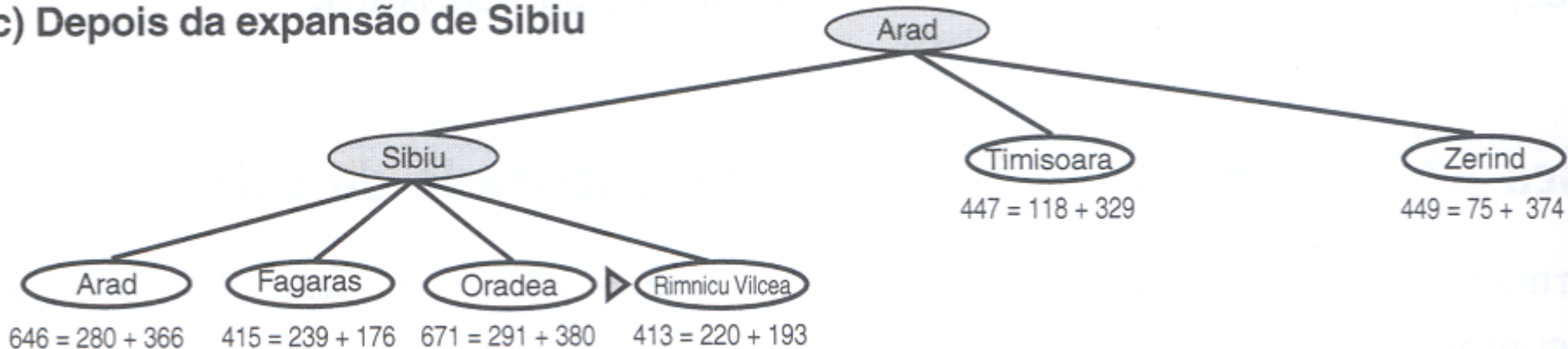
(a) O estado inicial



(b) Depois da expansão de Arad

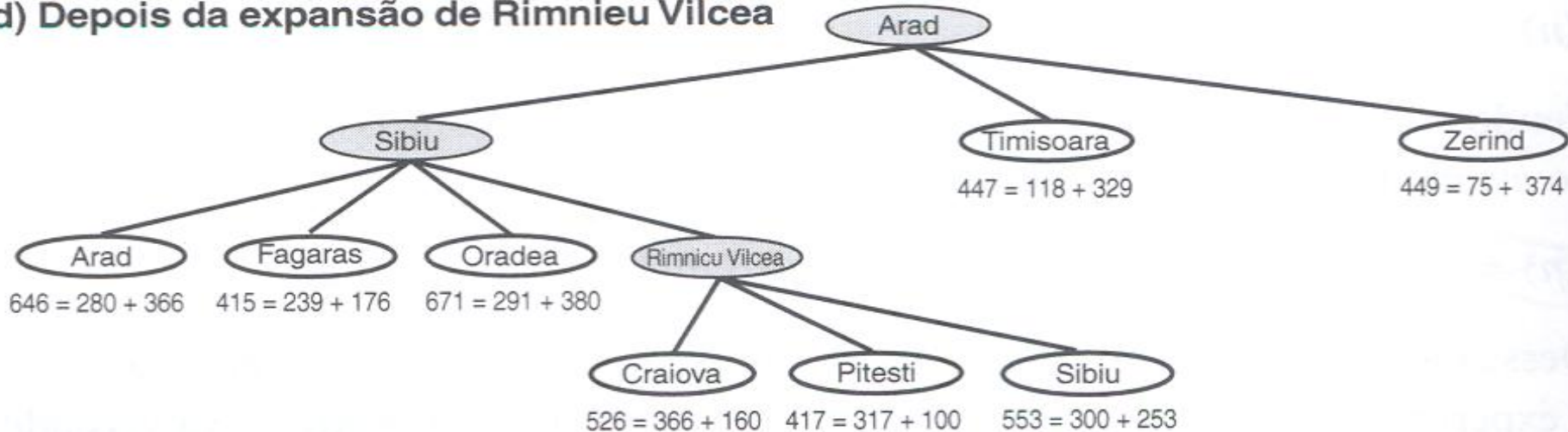


(c) Depois da expansão de Sibiu

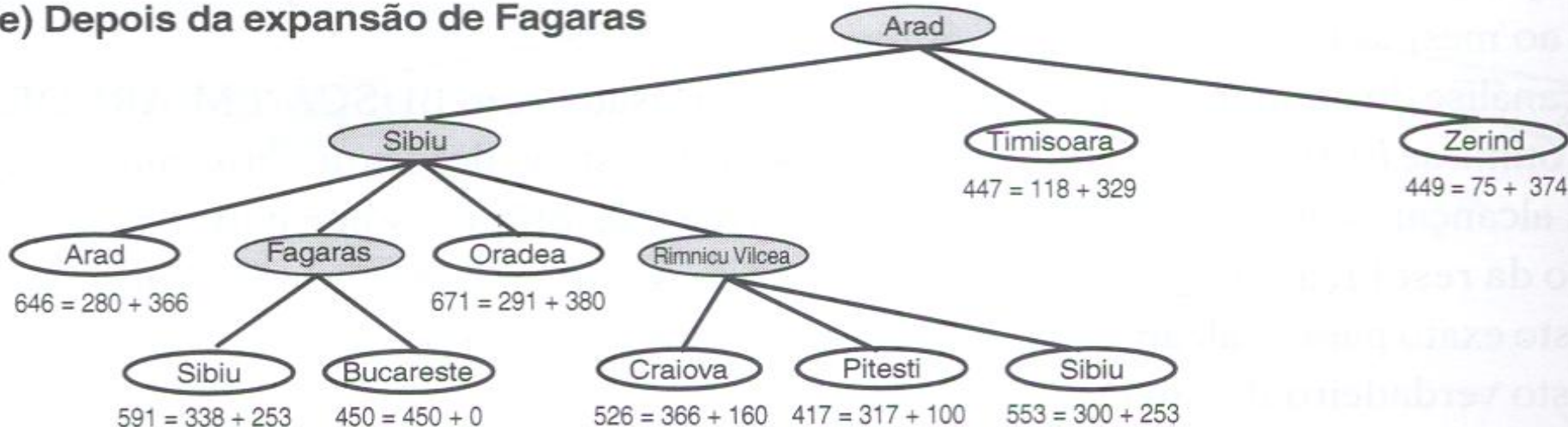


O PROCESSO EXECUTADO POR UMA BUSCA A*

(d) Depois da expansão de Rimniew Vilcea

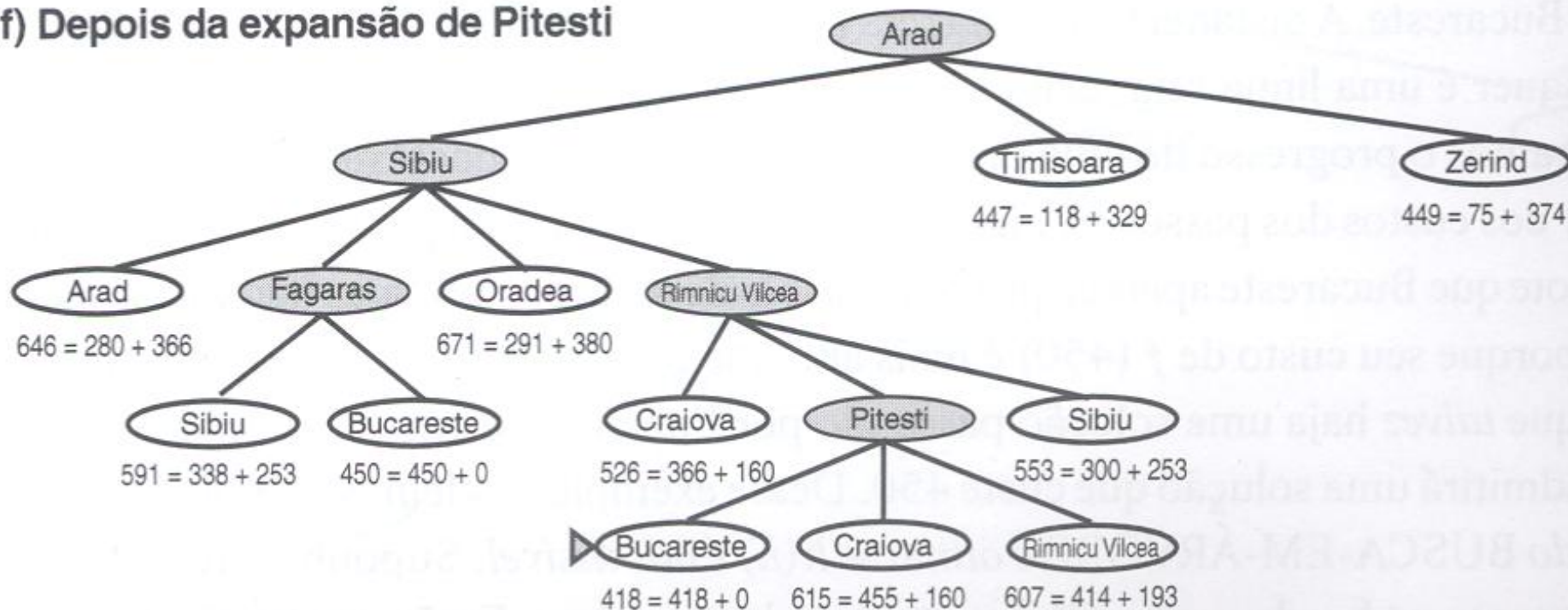


(e) Depois da expans3o de Fagaras



O PROCESSO EXECUTADO POR UMA BUSCA A*

(f) Depois da expansão de Pitesti



A* usando BUSCA-EM-ÁRVORE é ótima se $h(n)$ é admissível.

Otimidade de A^*



Pense!!

Suponha que um **nó-objetivo não-ótimo** G_2 aparece nas folhas (nó gerado e ainda não expandido) e que C^* é o custo da solução ótima.

Então, como G_2 não é ótimo e $h(G_2) = 0$, sabe-se que:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

Agora considere um nó folha n que está em *um caminho* de solução ótimo

(Pitesti na figura anterior (e) – sempre haverá tal nó se existe uma solução).

Se $h(n)$ não **superestimar** o custo de completar o caminho até a solução, então:

$$f(n) = g(n) + h(n) \leq C^*$$

Assim vemos que $f(n) \leq C^* < f(G_2)$ e portanto G_2 não será escolhido para expansão e A^* deve retornar uma solução ótima.

Funções Heurísticas

- ▶ Considere o problema do quebra-cabeças de 8 peças:

7	2	4
5		6
8		1

Estado inicial

	1	2
3	4	5
6	7	8

Estado objetivo

- ▶ **Objetivo:**
 - ▶ deslizar os blocos no sentido horizontal ou vertical para o espaço vazio, até a configuração coincidir com a configuração objetivo.
- ▶ O custo médio de solução para uma instância do quebra-cabeça de 8 peças, **gerada ao acaso é cerca de 22 passos.**
 - ▶ O fator de ramificação é aproximadamente igual a 3.
- ▶ Quando o espaço vazio está no meio, há quatro movimentos possíveis
 - ▶ Quando o espaço vazio está em um canto, são possíveis dois movimentos;
 - ▶ Quando o espaço vazio está em uma borda, há três movimentos;
 - ▶ Tem-se então, em uma busca exaustiva, 322 estados para exame.

Brute Force (Exhaustive Search)

Shaun Gause & Yu Cao

CSE Department
University of South Carolina

Problem	Nodes	Brute-Force Search Time (10 million nodes/second)
▶ 8 Puzzle:	10^5	.01 seconds
▶ 15 Puzzle:	10^{13}	6 days
▶ 24 Puzzle:	10^{25}	12 billion years
▶ 48 Puzzle:	10^{48}	why dinosaurs really went extinct



Funções Heurísticas

- ▶ Se quisermos descobrir soluções usando A^* , precisaremos de uma função heurística que nunca superestime “o número de passos até o objetivo”.
- ▶ Candidatas:
 - ▶ h_1 = o **número de blocos em posições erradas**. Se todos os blocos estão em posições erradas então $h_1=8$. h_1 é uma heurística admissível porque é claro que qualquer bloco que esteja fora do lugar deve ser movido pelo menos uma vez.
 - ▶ h_2 = **a soma das distâncias dos blocos de suas posições objetivo**. Como os blocos não podem se mover em diagonal, a distância que levaremos em conta é distância de Manhattan.
 - ▶ h_2 é admissível porque o resultado de qualquer movimento é deslocar um bloco para uma posição mais próxima do objetivo. No exemplo:

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

- ▶ O custo da solução é 26

7	2	4
5		6
8		1

Estado inicial

	1	2
3	4	5
6	7	8

Estado objetivo

FUNÇÕES HEURÍSTICAS

Pense!!

Uma maneira de caracterizar a qualidade de uma heurística é o *fator de ramificação efetiva*, b^* .

Se o número total de nós gerados pelo A^* para um determinado problema é N , e se a profundidade da solução é d , então b^* é o fator de ramificação que uma árvore uniforme de profundidade d precisaria ter para com conter $N + 1$ nós.

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Por exemplo, se A^* encontrar uma solução na profundidade 5 usando 52 nós, o fator de ramificação efetiva será 1,92.

b^* é, em geral, relativamente constante para problemas suficientemente difíceis. Então, medidas experimentais de b^* em um pequeno conjunto de problemas podem fornecer uma boa orientação sobre a utilidade geral da heurística.

Uma heurística bem projetada teria o valor de b^* próximo a 1, permitindo a resolução de problemas bastante extensos.

FUNÇÕES HEURÍSTICAS

Pense!!

A tabela abaixo mostra o número médio de nós expandidos por cada estratégia e o fator de ramificação efetivo.

d	Número médio de nós expandidos			Fator de ramificação efetiva		
	s/ heurística	A*(h1)	A*(h2)	s/	A*(h1)	A*(h2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	-	539	113	-	1,44	1,23
16	-	1301	211	-	1,45	1,25
18	-	3056	363	-	1,46	1,26

d = tamanho da solução

h_2 domina h_1

CRIAÇÃO DE FUNÇÕES HEURÍSTICAS ADMISSÍVEIS

h_1 e h_2 são estimativas do comprimento de caminho restante para o quebra-cabeça de 8 peças, mas também são comprimentos de caminho perfeitamente precisos para versões simplificadas do quebra-cabeça.

Se as regras do quebra-cabeça fossem alteradas de forma que um bloco pudesse se deslocar para qualquer lugar, e não apenas para o quadrado vazio adjacente, então h_1 daria o número exato de passos da solução mais curta.

De modo semelhante, se um bloco pudesse se mover para um quadrado em qualquer direção, então h_2 forneceria o número exato de passos na solução mais curta.

Um problema com menos restrições sobre as ações é chamado **problema relaxado** e o custo de uma solução ótima para um problema relaxado é uma heurística admissível para o problema original.



EXEMPLIFICANDO ...

- Problema original:
 - Um bloco pode se mover do quadrado A para o quadrado B se
 - A é horizontal ou verticalmente adjacente a B e B é vazio.
- Problemas relaxados:
 - Um bloco pode se mover do quadrado A para o quadrado B se
 - A é adjacente a B; (h_2)
 - Um bloco pode se mover do quadrado A para o quadrado B se
 - B está vazio; (heurísticas de Gaschnig)
 - Um bloco pode se mover do quadrado A para o quadrado B.
(h_1)



Busca Local

- ▶ Algoritmos de busca do tipo que estudamos até agora, realizam uma exploração sistemática do espaço de busca.
 - ▶ Este caráter sistemático é alcançado mantendo-se um ou mais caminhos na memória e registrando-se as alternativas que foram exploradas em cada ponto ao longo do caminho e quais delas não foram exploradas;
 - ▶ Quando um objetivo é encontrado, o caminho até esse objetivo também constitui uma solução para o problema.
- ▶ **Em alguns casos, o caminho até o objetivo não é relevante.**
 - ▶ Veja o caso do problema das 8 rainhas: o que importa é a configuração final.
 - ▶ Outros exemplos:
 - Layout de instalações industriais, projeto de circuitos integrados, **escalonamento de jornadas de trabalho**, roteamento de veículos, etc.



Busca local

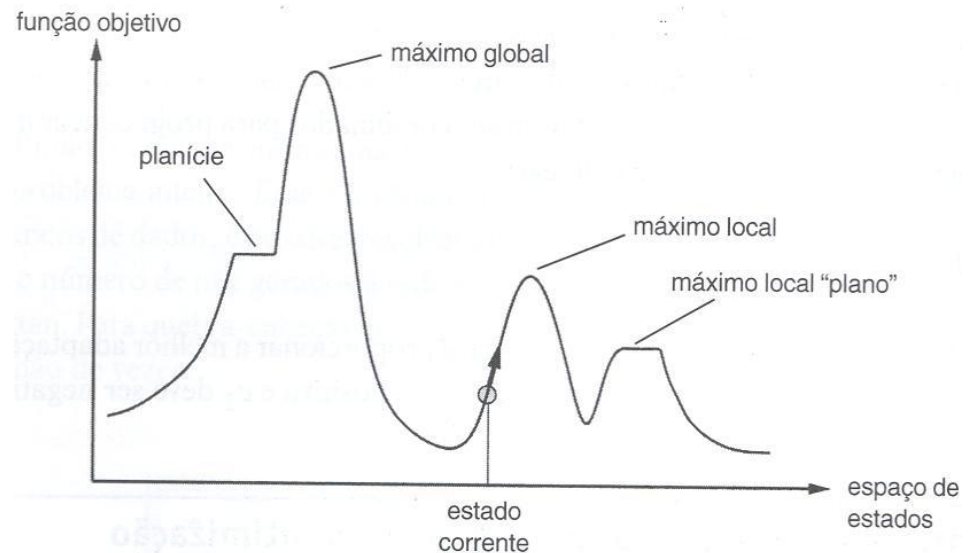
- ▶ Os algoritmos de busca local operam usando **um único estado corrente** (em vez de vários caminhos) e em geral se movem apenas para os **vizinhos desse estado**.
- ▶ Vantagens;
 - ▶ Usam pouquíssima memória – quase sempre um valor constante;
 - ▶ Frequentemente podem encontrar soluções razoáveis em espaços de estados grandes ou infinitos.
- ▶ São úteis para resolver **problemas de otimização puros**, nos quais o objetivo é encontrar o melhor estado de acordo com uma função objetivo.



Conceitos Relacionados

- ▶ Topologia do espaço de estados:

- ▶ Mínimo local e global
- ▶ Máximo local e global

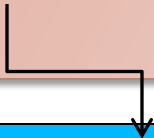


- ▶ Um algoritmo de busca local completo sempre encontra um objetivo, caso ele exista.
- ▶ Um algoritmo de busca local ótimo sempre encontra um mínimo/máximo global.
- ▶ Efeitos de uma busca de **subida de encosta** e uma busca de **têmpera simulada** (ou simulated annealing)

Busca de **subida de encosta** (busca gulosa local)

- ▶ Um laço repetitivo que se move de forma contínua no sentido do valor crescente, isto é, encosta acima.

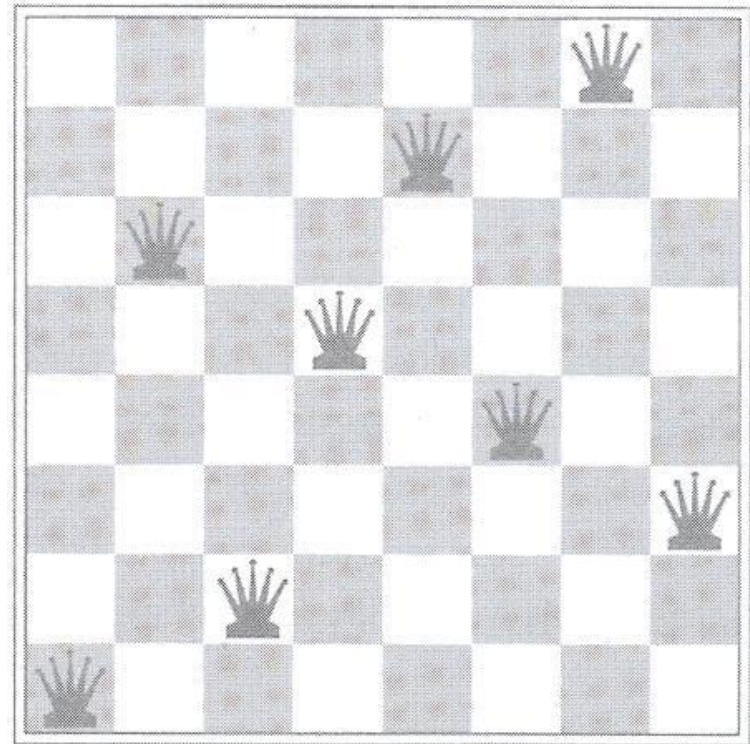
- **função** SUBIDA-DE-ENCOSTA (problema) **retorna** um estado que é um máximo local
- **entradas:** problema //um problema
- **variáveis locais:** *corrente*, *vizinho* // dois nós
- $corrente \leftarrow \text{CRIAR-NÓ}(\text{ESTADO-INICIAL}[\text{problema}]);$
- **repita**
- $vizinho \leftarrow \text{um sucessor de } corrente;$
- **se** VALOR[vizinho] <= VALOR[corrente]
- **então retornar** ESTADO[corrente]; // possibilidade de término
- **senão** $corrente \leftarrow vizinho;$



- Primeiro melhor
- Melhor dos Melhores

Contextualizando – 8 rainhas

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♚	13	16	13	16
♚	14	17	15	♚	14	16	16
17	♚	16	18	15	♚	15	♚
18	14	♚	15	15	14	♚	16
14	14	13	17	12	14	12	18



- Melhor dos melhores;
- Primeiro melhor

Busca de subida de encosta

- ▶ Razões que paralisar a busca:
 - ▶ **Máximos locais:** é um pico mais alto que cada um de seus estados vizinhos, embora seja mais baixo que o máximo global. O estado da última figura do slide anterior é um mínimo local → todo movimento de uma única rainha piora a situação.
 - ▶ **Picos:** resultam em uma sequência de máximo locais que torna muito difícil a navegação.
 - ▶ **Platôs:** área da topologia de espaço de estados em que a função de avaliação é plana. Pode ser um máximo local plano ou uma planície.



Têmpera simulada (Simulated Annealing)

- ▶ Um algoritmo de subida de encosta que **nunca faz movimentos “encosta abaixo”** em direção a estados com valor mais baixo (ou de custo mais alto) sem dúvida é incompleto, **porque pode ficar paralisado em um máximo (ou mínimo) local**.
- ▶ O algoritmo de Têmpera Simulada combina a subida de encosta com um percurso aleatório, resultando, de algum modo, em eficiência e completeza.
- ▶ Introduz-se uma modificação no algoritmo de subida de encosta:
 - ▶ em vez de escolher o melhor movimento, **escolhe-se um movimento aleatório**;
 - ▶ Se o movimento melhorar a situação, ele será aceito;
 - ▶ Caso contrário, o algoritmo **aceitará o movimento com alguma probabilidade menor que 1**;
 - ▶ A probabilidade diminui exponencialmente de acordo com a má qualidade do movimento;
 - ▶ A probabilidade também diminui à medida que a temperatura T se reduz (movimentos ruins tem maior probabilidade de serem aceitos no início);

Têmpera simulada

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

A dark blue vertical bar on the left side of the slide.

Computação Evolutiva

A light blue vertical bar on the left side of the slide.

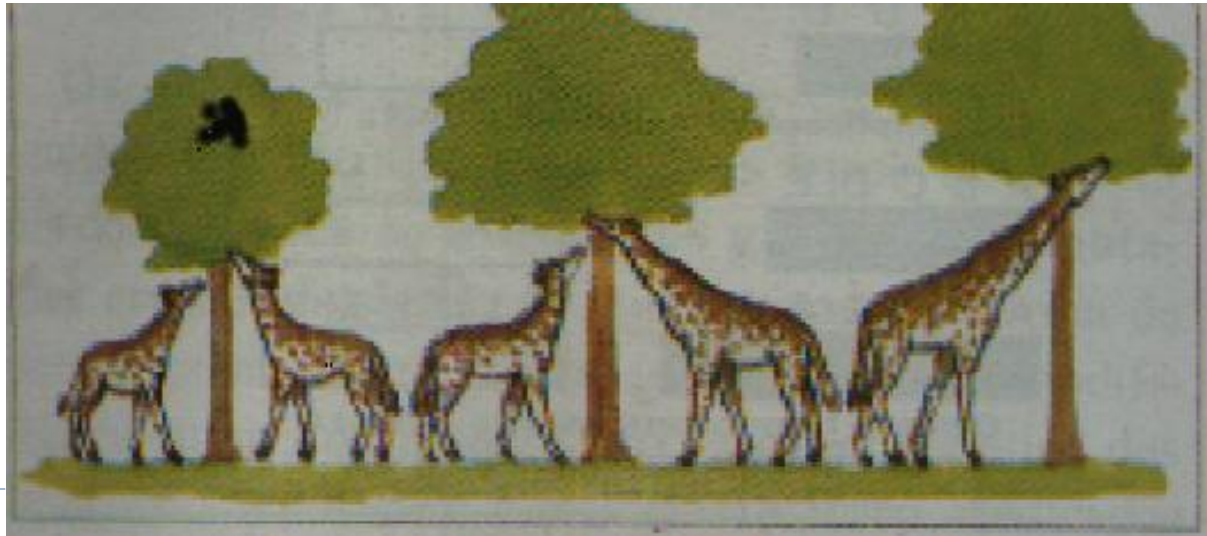
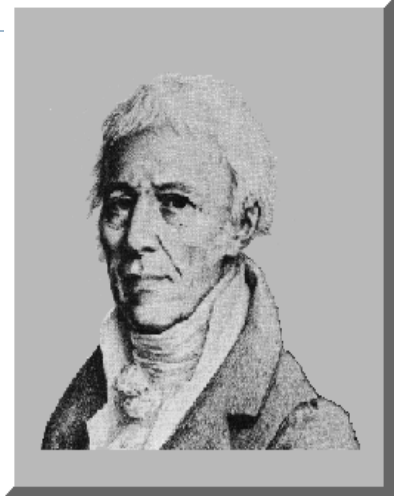
Indagações de alguns séculos atrás...



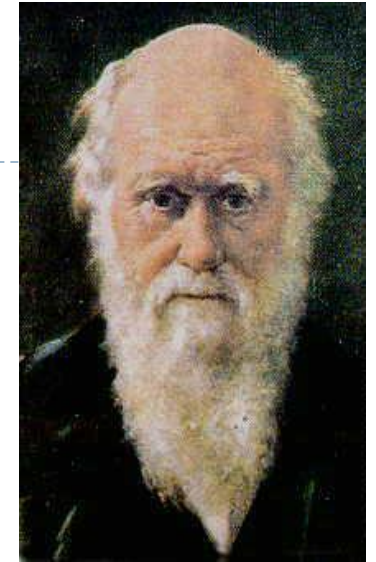
- ▶ Como explicar a diversidade de animais?
- ▶ Como explicar sua evolução?
 - ▶ Qual é a influência dos antepassados?
 - ▶ Qual é a influência do meio ambiente?

História da Teoria da Evolução

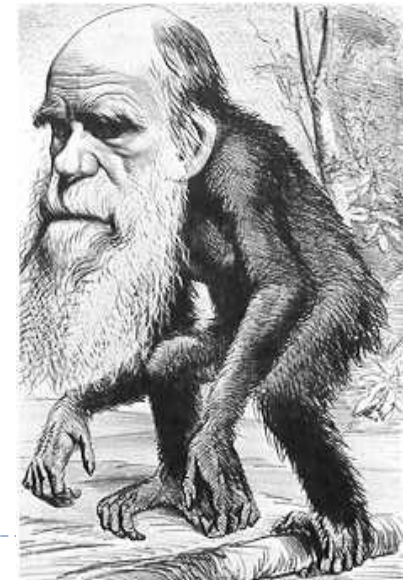
- ▶ 1809: Jean-Baptiste Lamarck
 - ▶ Lei do uso e do desuso
 - ▶ pelo uso e desuso de suas aptidões, a natureza força os seres a se adaptarem para sobreviverem.
 - ▶ Lei dos caracteres adquiridos.
 - ▶ Os seres mais fortes são mais capazes de “transmitir” suas aptidões às novas gerações



História da Teoria da Evolução



- ▶ 1859: Charles Darwin
 - ▶ Existe uma diversidade de seres devido aos contingentes da natureza (comida, clima, ...) e é pela lei da **Seleção Natural** que os seres mais adaptados ao seus ambientes sobrevivem
 - ▶ contra lei do uso de desuso
 - ▶ Os caracteres adquiridos são herdados pelas gerações seguintes
 - ▶ o homem vem do macaco...
- ▶ Na época, que isto tudo foi polêmico...



História da Teoria da Evolução

▶ 1865: Gregor Mendel

- ▶ Formalizou a “herança de características”, com a teoria do DNA (ervilhas)

▶ 1901: Hugo De Vries

- ▶ Só a seleção natural não é responsável pela produção de novas (mais adaptadas) espécies. Tem de haver uma mudança genética!
- ▶ Formalizou o processo de geração de diversidade: **Teoria da Mutação**

Fundamentos de computação evolutiva

- ▶ A computação evolutiva é uma meta-heurística populacional baseada no princípio da **seleção natural de Darwin**.
- ▶ Definição de algoritmos evolutivos:
 - ▶ São procedimentos computacionais para a solução de problemas ou modelagem de processos evolutivos, resultantes da aplicação de técnicas heurísticas baseadas na seguinte sequência básica comum: **realização de reprodução, imposição de variações aleatórias, promoção de competição e execução de seleção de indivíduos de uma dada população**.



Fundamentos de computação evolutiva

- ▶ Logo, **a evolução** é caracterizada basicamente por um processo constituído de 3 passos:
 - ▶ **1. Reprodução com herança genética;**
 - ▶ **2. Introdução de variação aleatória em uma população de indivíduos;**
 - ▶ **3. Aplicação da “seleção natural” para a produção da próxima geração.**
- ▶ **A variação cria diversidade na população**, a diversidade é transmitida por herança e a seleção elimina indivíduos menos aptos, quando comparados aos demais indivíduos da população.



Fundamentos de computação evolutiva

- ▶ Quatro operações são fundamentais junto à população de indivíduos: **reprodução; variação; atribuição de um valor ou grau de adaptação relativa (denominado fitness); e escolha de indivíduos que irão compor a próxima geração.**
- ▶ A disponibilidade de uma quantidade significativa de recursos computacionais é um requisito para viabilizar a implementação computacional de algoritmos evolutivos.
- ▶ A seleção natural em si não tem um propósito definido, mas em computador **é possível impor um propósito matematicamente** definido, por exemplo, para implementar metaheurísticas de otimização.
- ▶ Neste sentido, os algoritmos evolutivos caracterizam-se como meta-heurísticas que compreendem metodologias de busca em espaços de soluções candidatas, capazes de gerenciar operadores computacionais de busca local e de busca global.

Fundamentos de computação evolutiva

- ▶ Como praticamente todas as meta-heurísticas, os algoritmos evolutivos geralmente **NÃO apresentam as seguintes propriedades:**
 - ▶ Garantia de obtenção da solução ótima;
 - ▶ Garantia de convergência;
 - ▶ Garantia de custo máximo para se chegar a uma solução.



Formalização matemática

- ▶ **Representação genotípica:**
 - ▶ **Codificação dos candidatos à solução** (um subconjunto deles irá compor a população a cada geração). Podem existir vários tipos de representação para o mesmo problema, mas qualquer uma delas deve ser capaz de representar todas as soluções-candidatas, mesmo que nem sempre de forma única.
- ▶ **Representação fenotípica:**
 - ▶ Interpretação do código.
- ▶ **Espaço de busca:**
 - ▶ Tem como elementos **todos os possíveis candidatos à solução do problema** e é definido a partir da representação genotípica. Dependendo da existência ou não de restrições, pode conter regiões **factíveis e infactíveis**.



Formalização matemática

- ▶ **Função de adaptação ou fitness:**
 - ▶ Atribui a cada elemento do espaço de busca um valor de adaptação, que será usado como medida relativa de desempenho. Representa a pressão do ambiente sobre o fenótipo dos indivíduos.
 - ▶ **Operadores de inicialização:**
 - ▶ Produzem a primeira geração de indivíduos (população inicial), tomando elementos do espaço de busca.
 - ▶ **Operadores genéticos:**
 - ▶ Implementam o mecanismo de introdução de variabilidade aleatória no genótipo da população.
 - ▶ **Operadores de seleção:**
 - ▶ Implementam o mecanismo de “seleção natural”.
 - ▶ **Índice de diversidade:**
 - ▶ Geralmente representa a distância média entre os indivíduos da população corrente. Pode ser medido no genótipo, no fenótipo ou levando-se em conta apenas o fitness (medida aproximada).
 - ▶ A **codificação genética** (representação genotípica) e a **definição da função de adaptação** ou fitness são as etapas mais críticas, embora o desempenho efetivo do processo evolutivo dependa das decisões tomadas em todas as etapas de definição do algoritmo.
-

Formalização matemática

- ▶ É necessário atribuir valores aos parâmetros envolvidos: **tamanho da população; probabilidade de aplicação dos operadores genéticos; argumentos dos operadores de seleção; e argumentos do critério de parada.**

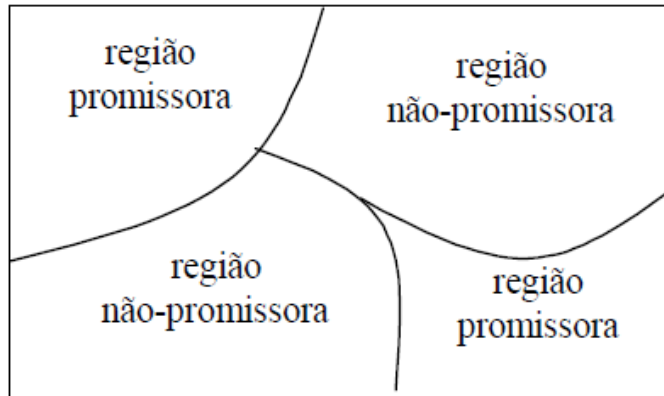
Termos Biológicos	Termos Computacionais
Cromossomo	Indivíduo
Gene	Caractere ou atributo
Alelo	Valor do atributo
Lócus	Posição do atributo
Genótipo	Vetor de atributos que representa o indivíduo
Fenótipo	Interpretação do vetor de atributos

- ▶ **Genótipo: representa o conjunto específico de genes do genoma.** Neste caso, indivíduos com o mesmo genoma são ditos terem o mesmo genótipo.
- ▶ **Fenótipo: é a manifestação do genótipo no comportamento, fisiologia e morfologia do indivíduo, como um produto de sua interação com o ambiente.**
- ▶ A seleção natural opera somente na expressão fenotípica do genótipo.

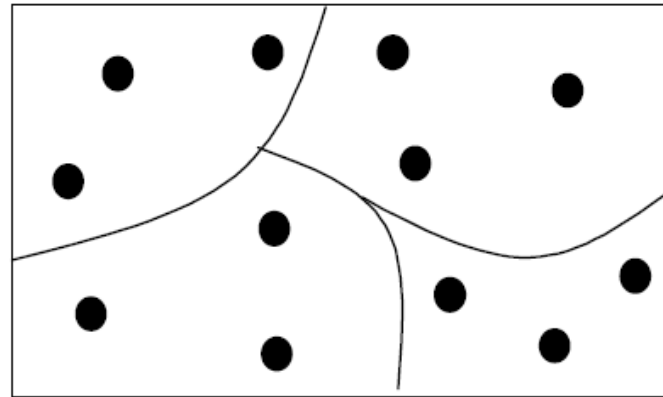


Visão pictórica da força evolutiva

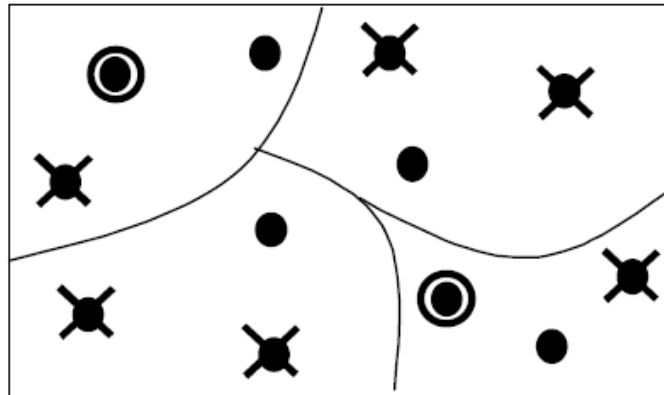
visão geral do espaço de busca



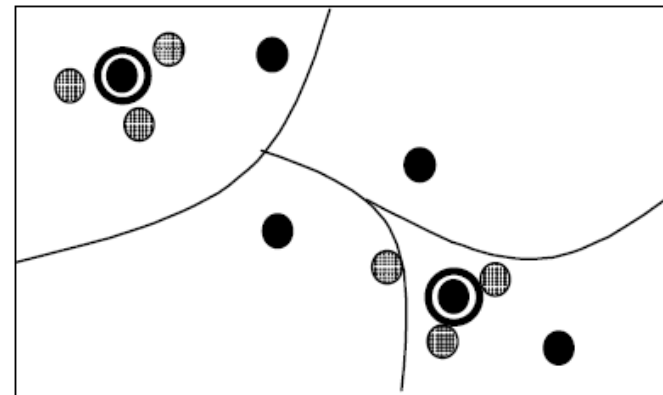
geração atual



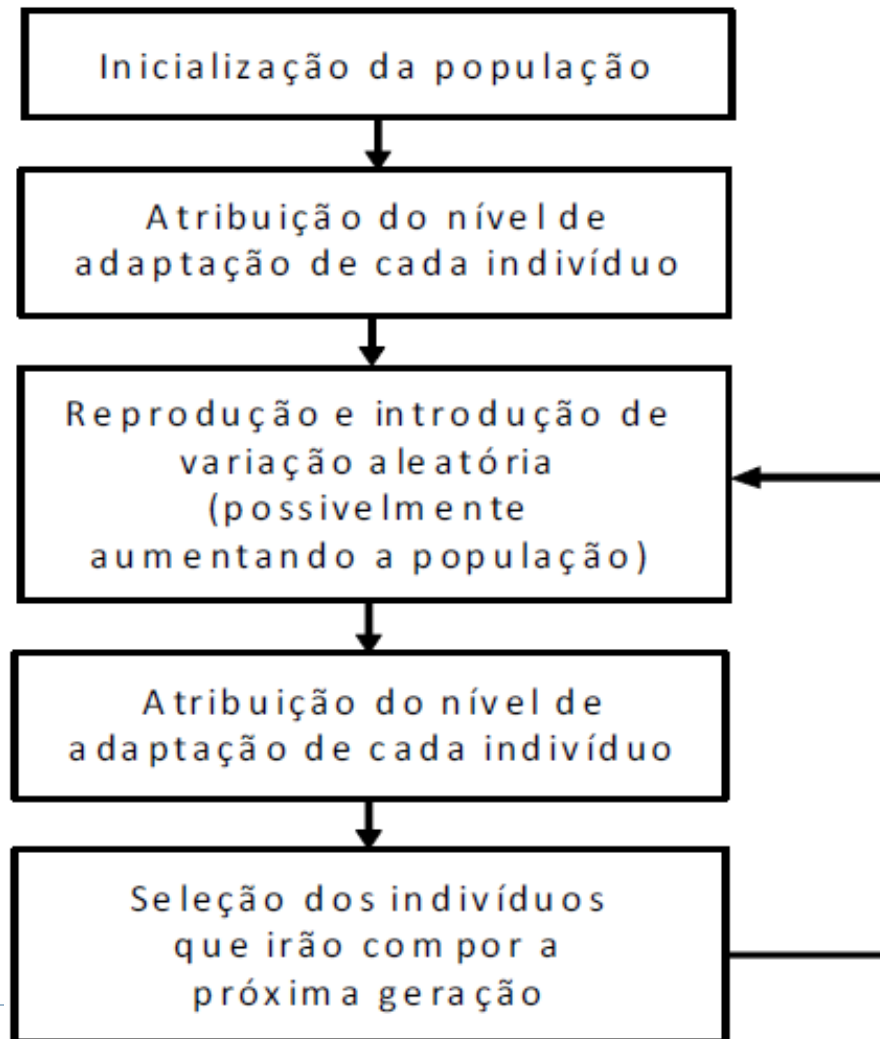
escolha dos indivíduos que irão se reproduzir e aqueles que irão ser substituídos



próxima geração



Fluxograma de um algoritmo evolutivo



Pseudo-código de um algoritmo evolutivo

Procedimento $[P] = \text{algoritmo_evolutivo}(N, pc, pm)$

$P'' \leftarrow \text{inicializa}(N)$

$fit \leftarrow \text{avalia}(P'')$

$t \leftarrow 1$

Enquanto condição_de_parada for FALSO **faça**,

$P \leftarrow \text{seleciona}(P'', fit)$

$P' \leftarrow \text{reproduz}(P, fit, pc)$

$P'' \leftarrow \text{varia}(P', pm)$

$fit \leftarrow \text{avalia}(P'')$

$t \leftarrow t + 1$

Fim Enquanto

Fim Procedimento

- ▶ Quais são as diferenças entre este pseudo-código e o fluxograma do slide anterior?



As várias faces dos algoritmos evolutivos

- ▶ Anos 50 e 60: Cientistas da computação já estudavam sistemas evolutivos com a ideia de que o mecanismo de evolução poderia ser utilizado como uma ferramenta de otimização para problemas de engenharia.
- ▶ FRASER (1959) – “Simulation of Genetic Systems by Automatic Digital Computers” – Australian Journal of Biological Science, 10:484-499.
- ▶ FRIEDBERG (1958) – “A Learning Machine: part I” – IBM Journal, pp. 2-13, Jan.
- ▶ ANDERSON (1953) – “Recent Advances in Finding Best Operating Conditions” – Journal of American Statistic Association, 48, pp. 789-798.
- ▶ BREMERMAN (1962) – “Optimization Through Evolution and Recombination” – in M.C. Yovits, G.T. Jacobi and D.G. Goldstein, editors – *Self-organizing Systems*, Spartan, Washington, D.C., pp. 93-106.

As várias faces dos algoritmos evolutivos

- ▶ RECHENBERG (1973) introduziu, nos anos 60, as **estratégias evolutivas**, *as quais foram* utilizadas para otimizar **parâmetros de valor real em sistemas dinâmicos**. As estratégias evolutivas foram aperfeiçoadas por SCHWEFEL (1975; 1977).
- ▶ FOGEL, OWENS e WALSH (1966) desenvolveram a **programação evolutiva**, *método* em que os candidatos à solução de um dado problema são representados por **máquinas de estado finito**, as quais evoluem pela mutação aleatória de seus diagramas de transição de estados, seguida pela seleção da mais bem adaptada. Uma formulação mais ampla da programação evolutiva pode ser encontrada em FOGEL (1999).



Pseudo-código de uma estratégia evolutiva

Procedimento $[P] = \text{estrategia_evolutiva}(\mu, \lambda)$

inicialize μ indivíduos do tipo: $\mathbf{v}_i = (\mathbf{x}_i, \boldsymbol{\sigma}_i, \boldsymbol{\theta}_i)$, $i = 1, \dots, \mu$

avale os indivíduos

$t \leftarrow 1$

Enquanto condição_de_parada for FALSA **faça**,

gere λ indivíduos (clonagem + mutação) a partir dos μ indivíduos

avale os indivíduos gerados

selecione os μ melhores indivíduos de λ ou $\mu + \lambda$

Fim Enquanto

Fim Procedimento

- ▶ Sugere-se o emprego da seguinte relação: $1 \leq \mu \leq \lambda$
 - ▶ Repare que cada indivíduo contém instruções sobre como aplicar a mutação aos seus descendentes e estes devem herdar estas instruções, com alguma variabilidade. A mutação emprega uma distribuição normal multivariada
-

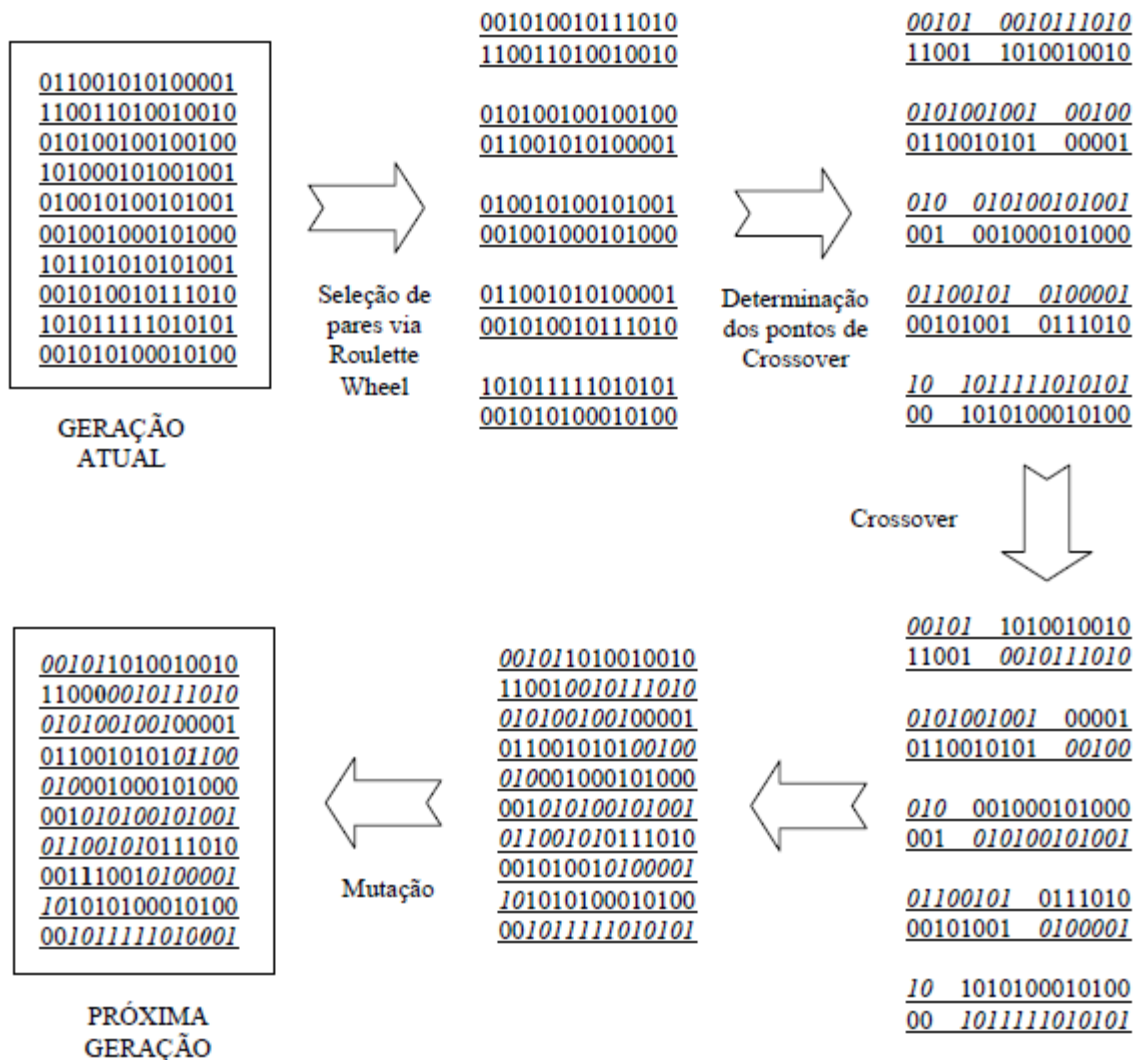


As várias faces dos algoritmos evolutivos

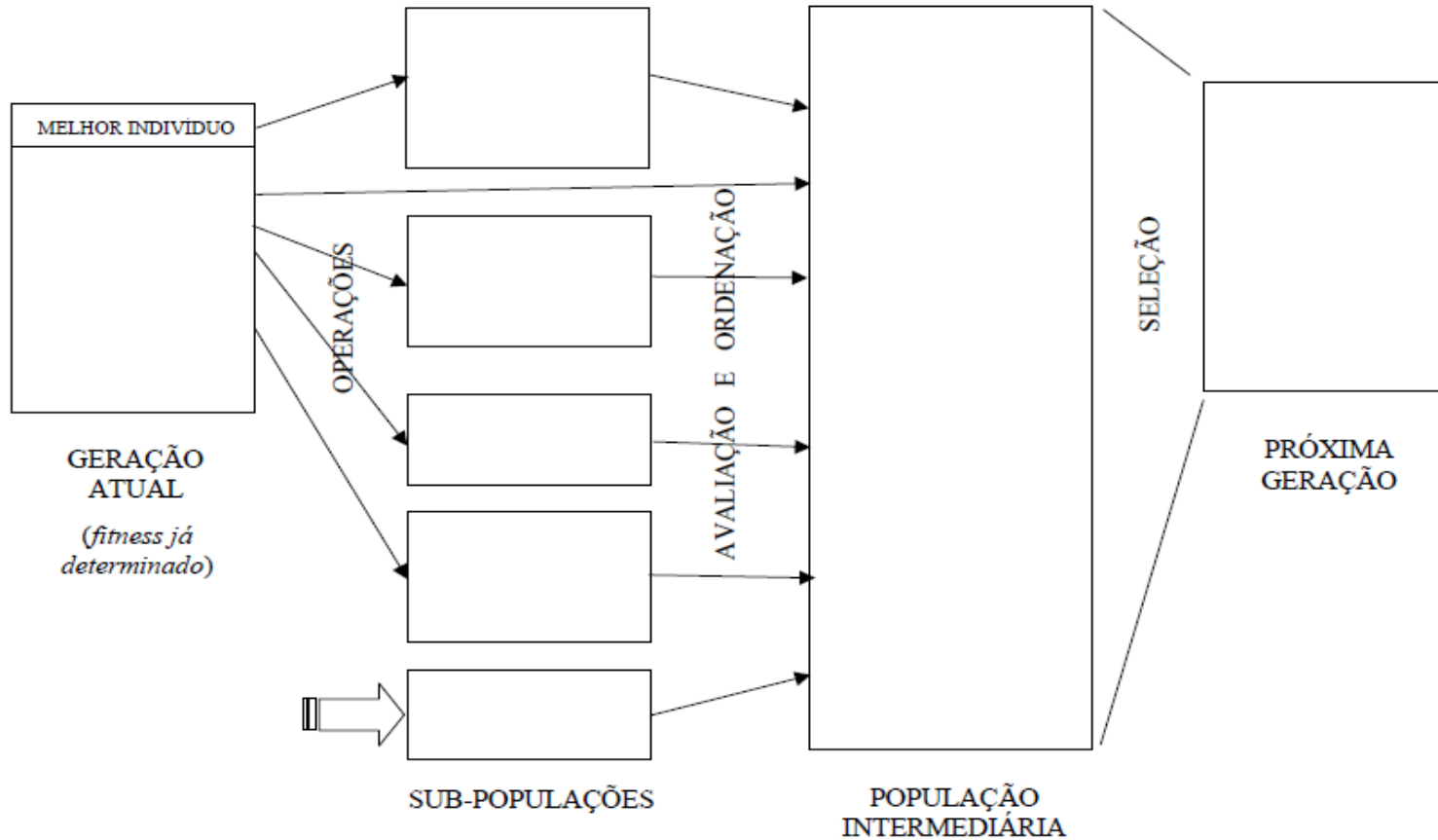
- ▶ Os **algoritmos genéticos** foram **propostos por Holland nos anos 60**, tendo sido desenvolvidos até meados dos anos 70 por seu grupo de pesquisa na Universidade de Michigan. Em contraste com as estratégias evolutivas e a programação evolutiva, o objetivo original de Holland não foi o de desenvolver algoritmos para a solução de problemas específicos, mas sim **estudar formalmente os fenômenos de adaptação, naturais ou artificiais, com o propósito de importar estes mecanismos de adaptação para ambientes computacionais.**
 - ▶ HOLLAND (1975/1992) apresentou os algoritmos genéticos **como uma abstração da evolução biológica**, tendo como inovações significativas a utilização conjunta de operadores de recombinação e inversão (além de operadores de mutação) e de um número elevado de indivíduos em cada geração.
-



Um fluxograma de um algoritmo genético



Algoritmo Genético Modificado



Uma proposta de extensão de um algoritmo genético

As várias faces dos algoritmos evolutivos

- ▶ HOLLAND (1975/1992) também apresentou os **sistemas classificadores**, especificamente implementados para operarem em ambientes não-estacionários, como exigido no caso de agentes autônomos.
 - ▶ Sistemas **Classificadores** representam metodologias para criação e atualização evolutiva de regras (denominadas classificadores) em um sistema de tomada de decisão.
 - ▶ Dadas as características de um ambiente em um determinado instante e levando-se em conta a “energia” de cada classificador (regra), alguns classificadores podem ser ativados. Eles codificam alternativas de ações específicas, as quais são submetidas a um processo de competição para selecionar aquela que será executada.
 - ▶ Dependendo do efeito de cada ação (ou sequência de ações) no atendimento dos objetivos (os quais podem estar implícitos), os classificadores responsáveis pelas ações serão recompensados ou punidos (ganhando ou perdendo “energia”).
 - ▶ Periodicamente, o elenco de classificadores é submetido a um processo evolutivo, que toma basicamente como medida de *fitness* a “energia” dos classificadores.
-



As várias faces dos algoritmos evolutivos

- ▶ Já no início dos anos 90, KOZA (1992) estendeu as técnicas de algoritmo genético para o espaço de **programas computacionais**, resultando na **programação genética**.
- ▶ Em programação genética, os indivíduos que constituem a população sujeita ao processo evolutivo, ao invés de apresentarem cadeias cromossômicas de comprimento fixo, **são programas que devem ser executados**. A programação genética representa uma iniciativa de se desenvolver métodos para a geração automática de programas computacionais genéricos.



Introdução

- ▶ Um algoritmo evolutivo **pode ser entendido como um procedimento iterativo de busca (otimização) inspirado nos mecanismos evolutivos biológicos, sendo assim modelos simplificados de processos biológicos.**
- ▶ Baseado na teoria neo-Darwiniana da evolução, é possível propor um *algoritmo evolutivo básico ou padrão com as seguintes características:*
 - ▶ *Uma população de candidatos à solução (denominados indivíduos ou cromossomos) que se reproduz com herança genética: cada indivíduo corresponde a uma estrutura de dados que representa ou codifica um ponto em um espaço de busca. Estes indivíduos devem se reproduzir (de forma sexuada ou assexuada), gerando descendentes que herdam algumas das características de seu(s) progenitor(es). No caso de reprodução sexuada, há troca de material genético (crossover ou recombinação) entre dois ou mais indivíduos progenitores;*



Introdução

- ▶ *Variação genética: durante o processo reprodutivo os descendentes não apenas herdam características de seu(s) progenitor(es), como eles também podem sofrer uma mutação genética responsável pela alteração de seu código genético;*
- ▶ *Seleção natural: a avaliação dos indivíduos em seu ambiente – através de uma função de avaliação ou fitness – resulta em um valor correspondente à adaptabilidade ou qualidade deste indivíduo. A comparação dos valores individuais de fitness resultará em uma competição pela sobrevivência e reprodução no ambiente, devendo ser promovida uma vantagem seletiva daqueles indivíduos com valores elevados de fitness.*
- ▶ Quando todos os passos acima (reprodução, variação genética e seleção) tiverem sido executados, diz-se que ocorreu uma *geração*.



Introdução

- ▶ Em cada iteração, sejam P uma população contendo N indivíduos $P = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ (**estruturas de dados**), $f_i, i = 1, \dots, N$, o valor do fitness de cada indivíduo da população, e p_c e p_m as probabilidades de crossover e mutação, respectivamente.
- ▶ Com isso, é possível propor um algoritmo evolutivo padrão da seguinte forma:

```
procedimento [P] = AE_padrao(N, pc, pm)
    P'' ← inicializa(N)
    fit ← avalia(P'')
    t ← 1
    enquanto NÃO condição_de_parada faça,
        P ← seleciona(P'', fit)
        P' ← reproduz(P, fit, pc)
        P'' ← varia(P', pm)
        fit ← avalia(P'')
        t ← t + 1
    fim enquanto
fim procedimento
```

Introdução

- ▶ O critério de parada pode variar bastante, sendo os mais simples baseados em **um número fixo de gerações ou na detecção de que se atingiu uma solução com desempenho acima de um limiar pré-estabelecido**



Os principais Algoritmos Evolutivos

▶ Tipos:

- ▶ Algoritmos Genéticos (AG) – Genetic Algorithms (GA)
- ▶ Estratégias Evolutivas (EE) – Evolution Strategies (ES)
- ▶ Programação Evolutiva (PE) – Evolutionary Programming (EP)
- ▶ Programação Genética (PG) – Genetic Programming (GP)
- ▶ Sistemas Classificadores (SC) – Classifier Systems (CS)

▶ Quais são as diferenças entre eles?

- ▶ Representação (codificação): qual estrutura de dados?
- ▶ Operadores genéticos: crossover e/ou mutação?
- ▶ Operadores de seleção: determinística ou probabilística?



Outras Características Principais

- ▶ *Algoritmos Genéticos: formalizados por HOLLAND (1975) com o nome de planos adaptativos, **ênfatisam a recombinação como o principal operador de busca e aplicam mutação com baixas probabilidades** (operador secundário). Operam com representação binária de indivíduos e possuem seleção probabilística proporcional ao fitness (implementada com um procedimento denominado de Roulette Wheel).*
 - ▶ *Estratégias Evolutivas: desenvolvidas por RECHENBERG (1973) e SCHWEFEL (1975, 1977), **utilizam mutações com distribuição normal para modificar vetores de números em ponto flutuante e enfatizam a mutação e a recombinação como operadores essenciais ao processo de busca no espaço de busca e no espaço de parâmetros.** O operador de seleção é determinístico e o tamanho da população de progenitores e de descendentes pode ser distinto.*
-



Outras Características Principais

- ▶ *Programação Evolutiva: formalizada por FOGEL et al. (1966), enfatiza a mutação e não incorpora a recombinação.* Assim como as EEs, quando aplicada a problemas de otimização de valores reais, a PE também emprega mutações com distribuição normal e estende o processo evolutivo ao espaço de parâmetros. O operador de seleção é probabilístico e a maioria das aplicações atuais emprega vetores de números em ponto flutuante como codificação, embora ela tenha sido originalmente desenvolvida para evoluir máquinas de estado finito.
- ▶ *Programação Genética: KOZA (1992) estendeu os algoritmos genéticos para o espaço de programas computacionais.* As estruturas de dados são representadas utilizando árvores, e os operadores de crossover e mutação (adaptados para operarem com estruturas do tipo árvore) são empregados. O processo de seleção segue aquele dos algoritmos genéticos, ou seja, a seleção é probabilística e proporcional ao fitness.



Resumo Geral

	AG	EE	PE	PG
Representação	Cadeias binárias	Vetores de números em ponto flutuante	Vetores de números em ponto flutuante	Árvores
Auto-adaptação	Nenhuma	Desvio padrão e co-variâncias	Desvio padrão e coeficiente de correlação	Nenhuma
O fitness é	Valor escalonado da função objetivo	Valor da função objetivo	Valor (escalonado) da função objetivo	Valor escalonado da função objetivo
Mutação	Operador secundário	Principal operador	Único operador	Um dos operadores
Recombinação	Principal operador	Diferentes variações, importante para a auto-adaptação	Nenhuma	Um dos operadores
Seleção	Probabilística	Determinística	Probabilística	Probabilística

Eventos Importantes

- ▶ Os seguintes eventos foram importantes para o estabelecimento da área de computação evolutiva.
 - ▶ Encontro de pesquisadores em algoritmos evolutivos na conferência *Parallel Problem Solving from Nature (PPSN)*, realizada em Dortmund em 1990.
 - ▶ Organização da primeira *International Conference on Genetic Algorithms (ICGA'91)* em 1991.
 - ▶ Chegada em um consenso para o nome da área – *computação evolutiva* – e o estabelecimento de um jornal homônimo pelo MIT Press em 1993.
 - ▶ Inclusão da área na primeira *World Conference on Computational Intelligence (WCCI)* em 1994, que incluiu: redes neurais artificiais, sistemas nebulosos e algoritmos evolutivos.



Algumas Possíveis Aplicações da Computação Evolutiva

- ▶ Aplicações de computação evolutiva podem ser encontradas em diversas áreas.
- ▶ Por conveniência, elas serão divididas aqui em cinco grandes áreas não exaustivas e não mutuamente exclusivas (BÄCK, 1994):
 - ▶ Planejamento
 - ▶ Projeto (*Design*)
 - ▶ Identificação e Controle
 - ▶ Classificação



Aplicações em Planejamento

▶ *Roteamento:*

- ▶ *Caixeiro viajante: qual é a melhor ordem de cidades a serem visitadas, dado o custo entre cidades e considerando o retorno à primeira cidade visitada?*
- ▶ *Roteamento de veículos: generalização do caixeiro viajante para mais de um veículo (caixeiro).*
- ▶ *Robótica: um caminho factível, seguro e sem colisões deve ser percorrido por um robô.*

▶ *Sequenciamento de tarefas (scheduling):*

- ▶ *desenvolver um plano para executar uma determinada quantidade de tarefas em um período de tempo, onde os recursos são limitados, existem restrições e pode haver mais do que um objetivo a ser otimizado.*



Aplicações em Planejamento

▶ *Job shop scheduling:*

- ▶ *um número finito de processos (jobs) deve ser processado por um número finito de máquinas. Cada processo consiste de uma sequência pré-determinada de tarefas (tasks), cada qual requerendo uma dada máquina por um certo intervalo de tempo, sem interrupção.*
- ▶ *As tarefas do mesmo processo não podem ser executadas concorrentemente e cada processo deve utilizar cada máquina exatamente uma vez. Um sequenciamento factível é uma atribuição de tarefas em janelas de tempo (time slots) das máquinas, sem violar as restrições do job shop. O makespan é definido como o tempo consumido para se completar todos os processos. O objetivo, portanto, é encontrar um sequenciamento que minimize o makespan. Um bom sequenciamento é aquele que minimiza o tempo total em que as máquinas ficam ociosas.*



Projeto (*Design*)

- ▶ *Filtros: projeto de sistemas eletrônicos ou digitais que implementam uma determinada resposta em frequência, tanto resposta ao impulso finita (FIR) quanto resposta ao impulso infinita (IIR).*
 - ▶ *Processamento de sinais: otimização do projeto de sistemas de processamento de sinais e desenho de circuitos integrados.*
 - ▶ *Sistemas inteligentes: definição de arquitetura e/ou parâmetros de redes neurais artificiais, sistemas nebulosos, autômatos celulares, e sistemas imunológicos artificiais, dentre outros.*
 - ▶ *Aplicações em engenharia: redes de telecomunicações, projetos estruturais, projeto de aeronaves, projeto de estruturas espaciais, projeto de reatores químicos, teste e diagnóstico de falhas, etc.*
-



Identificação e Controle

- ▶ *A identificação envolve a determinação de parâmetros de modelos a partir de um comportamento desejado.*
- ▶ Duas abordagens distintas para controle de processos: *on-line* e *off-line*.
 - ▶ *Off-line: um algoritmo evolutivo é utilizado para projetar um controlador que é depois utilizado para controlar um sistema.*
 - ▶ *On-line: um algoritmo evolutivo é utilizado como uma parte ativa do controlador.*
- ▶ Uma vantagem de um controlador evolutivo é que ele pode ser utilizado em ambientes dinâmicos, embora com limitações (evolução × adaptação).



Aplicações em Classificação

- ▶ *Sistemas classificadores têm sido utilizados como partes de outros sistemas como, por exemplo, sistemas de controle.*
- ▶ Algoritmos evolutivos também têm sido aplicados a problemas de jogos (*game playing*).
- ▶ Muitas outras aplicações de AEs existem em classificação como, por exemplo, processamento de imagens e mineração de dados



A dark blue vertical bar on the left side of the slide.

Algoritmo Genético

A light blue vertical bar on the left side of the slide.

Algoritmo Genético Tradicional

- ▶ População de tamanho fixo e estrutura de dados do tipo cadeias binárias;
- ▶ Seleção natural proporcional ao fitness via algoritmo Roulette Wheel;
- ▶ Crossover simples (crossover de um ponto);
- ▶ Mutação pontual.



Otimização - Dificuldades

- ▶ Alguns problemas podem ter espaços de busca muito grandes
- ▶ Muitos algoritmos não são capazes de localizar ótimo global na presença de múltiplos ótimos locais
 - ▶ Ex.: Hill Climbing



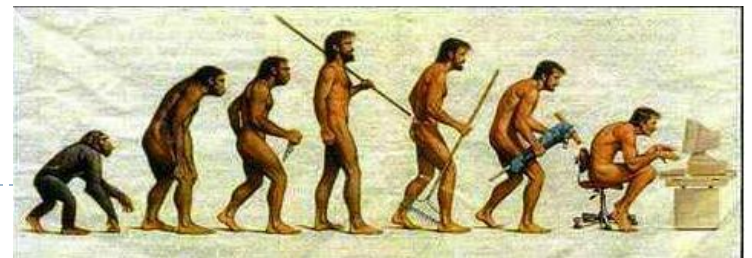
Algoritmos Genéticos (Goldberg) (Linden)

- ▶ São algoritmos de busca baseados no mecanismo de seleção natural e na genética natural.
 - ▶ Desenvolvidos por John Holland e sua equipe na Universidade de Michigan.
 - ▶ A pesquisa em AGs possui duas metas:
 - ▶ Abstrair e explicar o processo adaptativo dos sistemas naturais;
 - ▶ Projetar sistemas de software artificiais que emulem os mecanismos dos sistemas naturais.
- ▶ É um sistema robusto:
 - ▶ Faz o balanço entre a eficiência e a eficácia necessária para sobrevivência em diferentes ambientes.
- ▶ Faz parte de uma área de estudo que usa modelos computacionais dos processos naturais de evolução como uma ferramenta para resolver problemas – Algoritmos Evolucionários ou Computação Evolutiva
 - ▶ Outras técnicas: Estratégias Evolucionárias e Programação Genética



Algoritmos Genéticos (Linden)

- ▶ Técnica de busca baseada em uma metáfora do processo biológico de evolução natural
- ▶ São técnicas heurísticas de otimização global (pode ser considerada uma meta-heurística).
- ▶ Utilizam:
 - ▶ População de estruturas, denominadas indivíduos ou cromossomos, sobre as quais são aplicados operadores genéticos. Cada indivíduo recebe uma avaliação que é uma quantificação de sua qualidade como solução do problemas em questão.
 - ▶ Operadores genéticos: aproximações computacionais de fenômenos vistos na natureza, como reprodução sexuada, mutação genética etc...



Algoritmos Genéticos

- ▶ Passo 1: Geração de uma população inicial com indivíduos escolhidos aleatoriamente
- ▶ Passo 2: Avaliação dos indivíduos
 - ▶ Cálculo da função de **fitness** (usando função objetivo)
- ▶ Passo 3: Seleção de indivíduos mais aptos
- ▶ Passo 4: Geração de uma nova população a partir dos indivíduos selecionados e ir para Passo 2
 - ▶ Operadores de busca (**crossover e mutação**)



Algoritmos Genéticos

Seja $S(t)$ a população de cromossomos na geração t .

$t \leftarrow 0$

inicializar $S(t)$

avaliar $S(t)$

enquanto o critério de parada não for satisfeito **faça**

$t \leftarrow t + 1$

 selecionar $S(t)$ a partir de $S(t-1)$

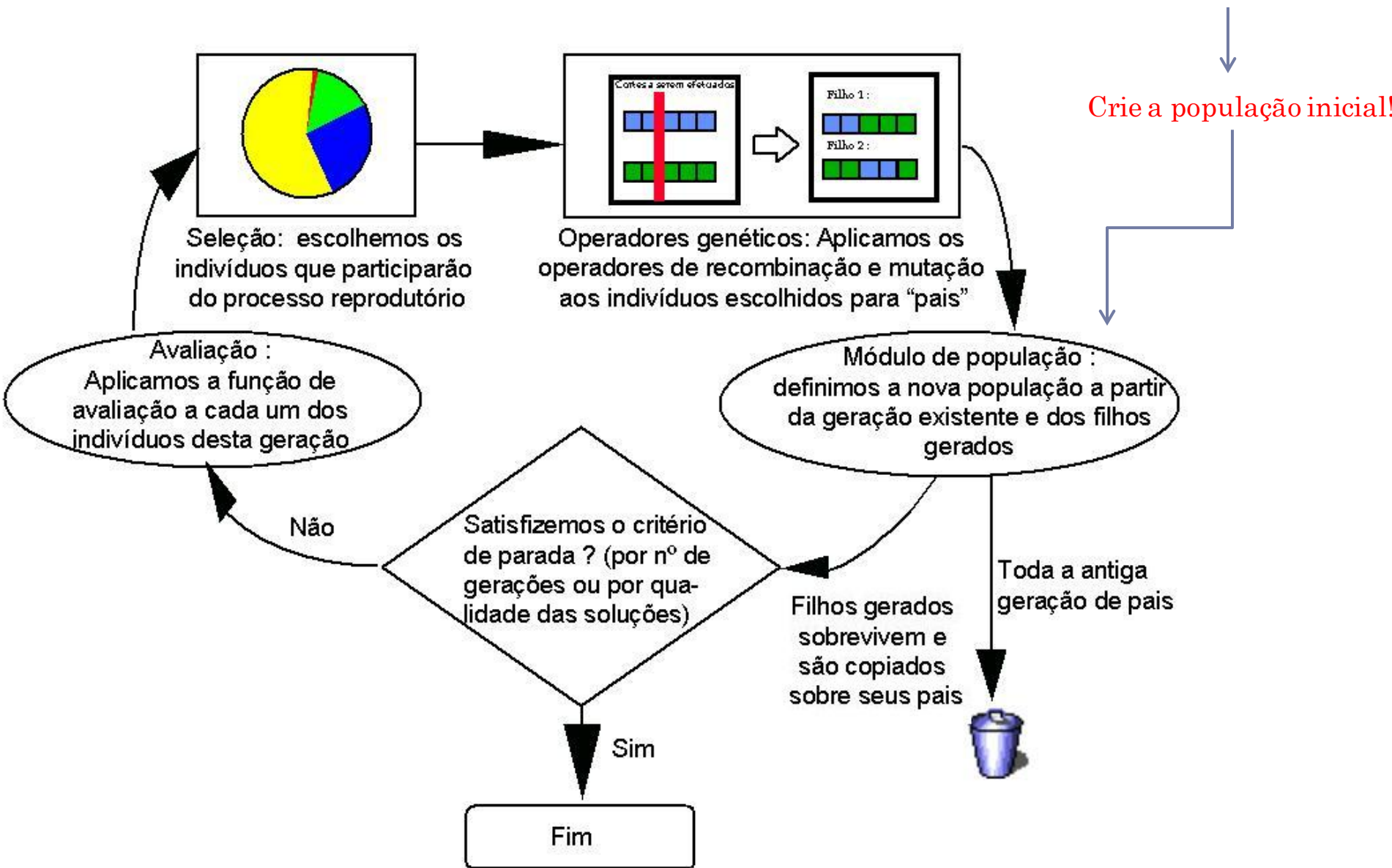
 aplicar *crossover* sobre $S(t)$

 aplicar mutação sobre $S(t)$

 avaliar $S(t)$

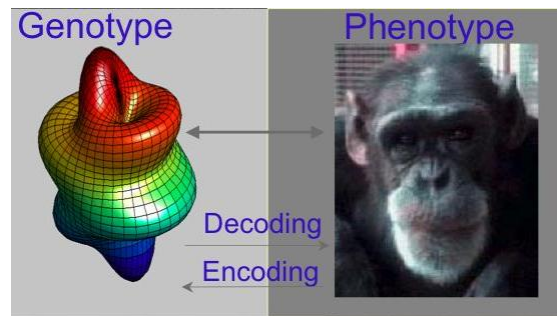
fim enquanto





Terminologia

- ▶ Cromossomo X indivíduo X string de bits
- ▶ Um cromossomo é formado por *genes*, que podem ter um determinado valor entre vários possíveis (*alelos*). A posição do gene é o *locus*.
- ▶ Genótipo X Fenótipo
 - ▶ Genótipo: é a estrutura do cromossomo, e pode ser identificada na área de GA como o termo *estrutura*.
 - ▶ Fenótipo: é a interação do conteúdo genético com o ambiente.



www.lania.mx/~asanchez/IA/GAapuntos/gen50.html

- ▶ Função fitness: mede a adaptabilidade de uma solução a um problema (é a medida de avaliação).
-

Algoritmos Genéticos

► Representação de conhecimento

► Representando o problema

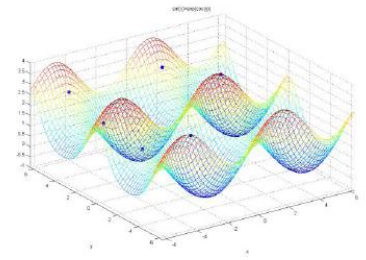
► O cromossomo

- A representação deve ser o mais simples possível;
- Se houver soluções infactíveis, é interessante que elas não sejam representadas
- Restrições do problema devem estar preferencialmente implícitas na representação

► Representando a avaliação no problema

► A função fitness (ou objetivo) – de avaliação

- Caso soluções infactíveis sejam permitidas na representação, ou restrições não estejam representadas, tente usar a função fitness para “matar” tais indivíduos.



Representação de Indivíduos

- ▶ Um cromossomo representa (codifica) um conjunto de parâmetros da função objetivo
 - ▶ E.g., na função $f(x) = x \sin(10\pi x) + 1$, um cromossomo codifica um valor do parâmetro x
- ▶ A representação de uma solução do espaço de busca é dependente do problema de otimização
 - ▶ Porém, alguns esquemas de representação podem ser reaproveitados



Representação Binária

- ▶ Cromossomo representado por uma cadeia de bits (0 ou 1)
 - ▶ Cada sequência de bits é mapeada para uma solução do espaço de busca
- ▶ Representação tradicional, fácil de manipular através de operadores de busca



Representação Binária - Exemplo

- ▶ Codificação de $-1 \leq x \leq 2$ com 22 bits
 - ▶ 2^{22} valores possíveis (tamanho do espaço)
- ▶ $S_1 = 10001011110110101000111$ na base 10 seria igual a 2288967
- ▶ Mapeado para intervalo $[-1; 2]$ representaria a solução:
 - ▶ $x_1 = \min + (\max - \min) * b_{10} / (2^{22} - 1) =$
 $-1 + (2 + 1) * 228896 / (2^{22} - 1) = 0,637197$



Representação Real

- ▶ Para otimização de parâmetros contínuos a representação binária não é adequada
 - ▶ Muitos bits para obter boa precisão numérica
- ▶ Parâmetros numéricos podem ser codificados diretamente nos cromossomos
 - ▶ Ex.: $S_1 = 0,637197$



Algoritmo Genético (Linden)

- ▶ **Convergência genética:**

- ▶ Se traduz em uma população com baixa diversidade genética que, por possuir genes similares, não consegue evoluir, a não ser pela ocorrência de mutações aleatórias que sejam positivas.

- ▶ **Perda de diversidade:**

- ▶ Pode ser definida como sendo o número de indivíduos que nunca são escolhidos pelo método de seleção de pais.



Seleção

- ▶ AGs selecionam indivíduos aptos de uma população para gerar novos indivíduos
 - ▶ **Cromossomos filhos** (novas soluções)
- ▶ Em geral, indivíduos pais são selecionados com uma probabilidade proporcional a seus valores de fitness
 - ▶ Probabilidade de seleção

$$p_i = \frac{f_i}{\sum_{i=1}^N f_i}$$



Seleção – Roda da Roleta

Ind. i	Aptidão f_i	Aptidão Acumulada $\Sigma(f_i)$
•	2,0	2,0
•	1,6	3,6
3	1,4	5,0
4	0,7	5,7
5	0,3	6,0

1. Ordenar aptidões da população
2. Calcular aptidões acumuladas
3. Gerar número aleatório entre [0; Última aptidão acumulada]
4. Indivíduo selecionado é o primeiro com aptidão acumulada maior que o número aleatório gerado

Exemplo: gerar número aleatório entre [0; 6]. Se 4.2 for o número gerado selecione indivíduo 2



Seleção – Roda da Roleta

- ▶ Observação importante:

- ▶ Não funciona para valores negativos da função de objetivo
- ▶ Nesse caso, deve-se função aptidão para valores positivos ou realizar ***Seleção por Torneio***



Seleção por Torneio

- ▶ Passo 1: Escolher inicialmente com a mesma probabilidade n indivíduos
- ▶ Passo 2: Selecionar cromossomo com maior aptidão dentre os n escolhidos
- ▶ Passo 3: Repetir passos 1 e 2 até preencher população desejada



Operadores Genéticos

- ▶ A etapa de seleção, gera uma população intermediária de potenciais cromossomos pais
- ▶ Na nova geração, escolhe-se aleatoriamente dois pais para aplicação de operadores genéticos (**crossover e mutação**)
- ▶ Produção de filhos é feita até completar o tamanho da população desejada



Operador Crossover – Representação Binária

- ▶ Aplicado a um par de cromossomos retirados da população intermediária para gerar filhos
 - ▶ Filhos herdam características dos pais
- ▶ Crossover de um ponto
 - ▶ Cortar pais em uma posição aleatória e recombinar as partes geradas

pai_1	(0010101011 10000011111)
pai_2	(0011111010 010010101100)
$filho_1$	(0010101011 010010101100)
$filho_2$	(0011111010 10000011111)

Operador Crossover – Representação Binária

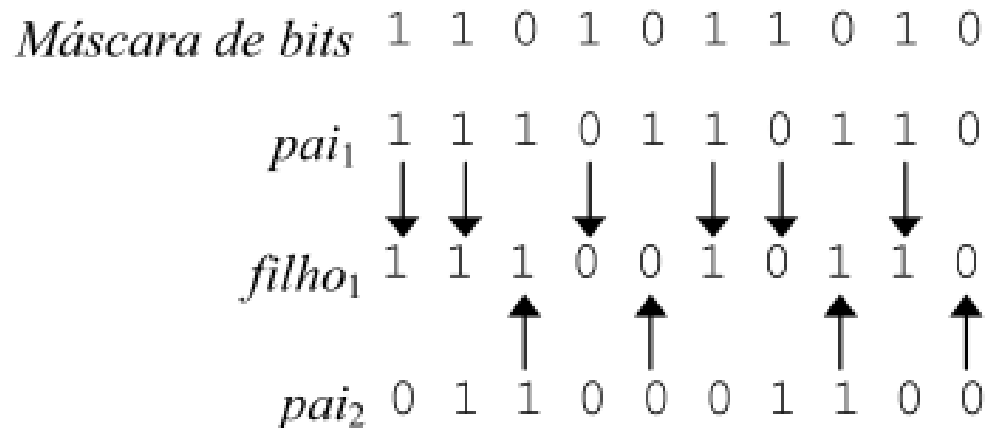
- ▶ Crossover de dois pontos
 - ▶ Cortar pais em duas posições aleatórias e recombinar as partes geradas

<i>pai</i> ₁	010	011000	101011
<i>pai</i> ₂	001	001110	001101
<i>filho</i> ₁	010	001110	101011
<i>filho</i> ₂	001	011000	001101

Operador Crossover – Representação Binária

► Crossover uniforme

- Gerar uma máscara de bits aleatórios e combinar os bits dos pais de acordo com a máscara gerada



Operador Crossover – Representação Real

- ▶ Na representação real, crossover é obtido através de operações aritméticas sobre os pais
- ▶ Crossover média aritmética
 - ▶ Filho = $(\text{pai1} + \text{pai2})/2$
- ▶ Crossover média geométrica
 - ▶ Filho = $\text{raiz}(\text{p1} * \text{p2})$



Operador Crossover – Representação Real

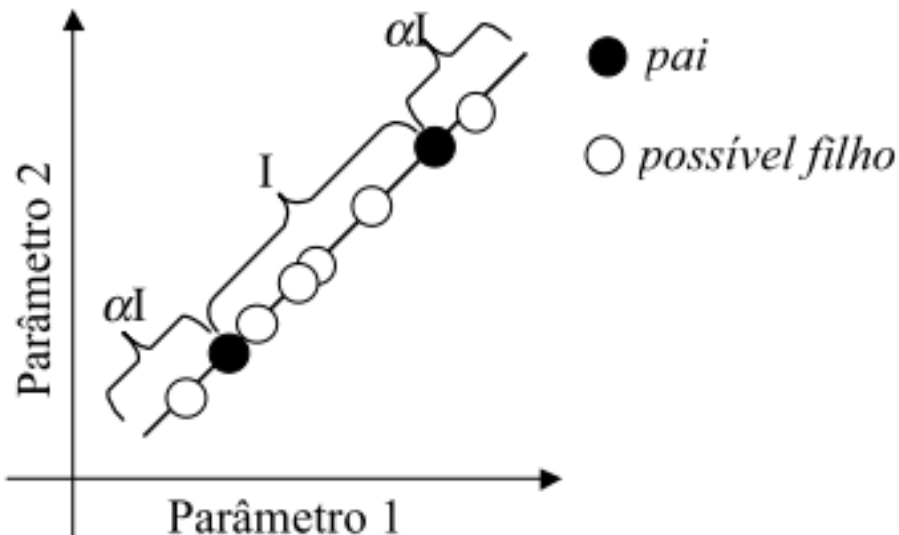
- ▶ Operadores de média tendem a diminuir muito a diversidade dos filhos
 - ▶ Filhos sempre vão estar no meio do intervalo dos pais
- ▶ Operador BLX- α
 - ▶ Filho = pai1 + β *(pai2 – pai1)
onde β é um número aleatório entre $[-\alpha, 1 + \alpha]$
 - ▶ Parâmetro α controla a diversidade dos filhos



Operador Crossover – Representação Real

► Operador BLX- α

- $\alpha = 0$ equivale a gerar filhos aleatoriamente no intervalo numérico entre os pais ($I = \text{pai2} - \text{pai1}$)
- Se $\alpha > 0$, o intervalo dos possíveis filhos é estendido em $\alpha \cdot I$ em ambos os lados



Operador Crossover

- ▶ Geralmente, crossover é aplicado somente com uma dada probabilidade (**taxa de crossover**)
 - ▶ Taxa de crossover é normalmente alta (entre 60% e 90%)
- ▶ Durante a aplicação do operador, é gerado um número aleatório r entre 0 e 1 e aplica-se teste:
 - ▶ Se $r < \text{taxa de crossover}$, então operador é aplicado
 - ▶ Senão, os filhos se tornam iguais aos pais para permitir que algumas boas soluções sejam preservadas



Operador Mutação – Representação Binária

- ▶ Aplicado sobre os cromossomos filhos para aumentar a variabilidade da população
- ▶ Operador para representação binária:
 - ▶ Para cada bit realize **teste de mutação** e troque o valor do bit caso o teste seja satisfeito

Antes	<i>filho₁</i>	(0010101010010010101100)
	<i>filho₂</i>	(0011111011100000111111)
Depois	<i>filho₁</i>	(0010 <u>0</u> 010100100101 <u>1</u> 1100)
	<i>filho₂</i>	(0011 <u>1</u> 11011 <u>0</u> 00000111111)

Operador Mutação – Representação Real

▶ Mutação Uniforme:

- ▶ Realiza teste de mutação para cada parâmetro codificado
- ▶ Substituir parâmetro por um número aleatório escolhido uniformemente em um intervalo $[a; b]$
 - ▶ Filho = $U(a; b)$

▶ Mutação Gaussiana:

- ▶ Realiza teste de mutação para cada parâmetro codificado
- ▶ Substituir parâmetro por um número aleatório escolhido usando uma distribuição Normal
 - ▶ Filho = $N(\mu; \sigma)$



Operador Mutação – Representação Real

► Mutação Creep:

- Adiciona um pequeno número aleatório ao valor atual do parâmetro armazenado no cromossomo; ou
- Multiplica valor atual por número próximo de um
- Observações:
 - Operador menos destrutivo que os anteriores
 - Usado para explorar localmente o espaço de busca
 - Pode ser aplicado em uma taxa um pouco mais alta



Algoritmos Genéticos – Observações Importantes

- ▶ Operador Crossover considera características importantes presentes nos pais
 - ▶ Aplicado a uma taxa relativamente alta, mas cuidado com efeitos destrutivos
- ▶ Operador Mutação explora novas características nos indivíduos que seriam possivelmente úteis
 - ▶ Aplicado a uma taxa relativamente baixa, mas dependendo do problema e operador use taxas mais altas



Algoritmos Genéticos – Observações Importantes

▶ Convergência Prematura

- ▶ Em algumas execuções, AG pode convergir para soluções iguais
 - ▶ Cromossomos com boa aptidão (mas ainda não ótimos) que geram filhos com pouca diversidade
- ▶ Nesses casos, aconselha-se:
 - ▶ Aumento da taxa de mutação e crossover
 - ▶ Evitar a inserção de filhos duplicados



Algoritmos Genéticos – Observações Importantes

▶ Critérios de Parada

- ▶ Número máximo de gerações
- ▶ Função objetivo com valor ótimo alcançado (quando esse valor é conhecido)
- ▶ Convergência na função objetivo (i.e., quando não ocorre melhoria significativa da função)



Algoritmos Genéticos – Observações Importantes

- ▶ População inicial
 - ▶ Não pode ser excessivamente pequena
 - ▶ Pouca representatividade do espaço de busca
 - ▶ Não pode ser excessivamente grande
 - ▶ Demora na convergência
 - ▶ Para melhorar a representatividade população inicial pode possuir indivíduos igualmente espaçados no espaço de busca



Um Algoritmo Genético simples

- ▶ Simplicidade de operação e poder de efeito são das duas principais atrações da abordagem de algoritmos genéticos.
- ▶ Depois que a codificação do problema está pronta, ou seja, a representação da solução está construída, a população inicial pode ser gerada.
- ▶ Retornemos ao nosso problema da caixa preta
 - ▶ A população é composta por 4 indivíduos, ou seja, por 4 possíveis configurações de interruptores (lembrem-se que queremos encontrar a configuração que otimiza uma função f)

```
0 1 1 0 1
1 1 0 0 0
0 1 0 0 0
1 0 0 1 1
```

População inicial instanciada aleatoriamente.



Um algoritmo genético simples

- ▶ Cada indivíduo da população representa uma solução para o problema, e cada um deles tem uma avaliação sobre a sua adequabilidade ao problema.
- ▶ A avaliação de cada indivíduo é o valor da função f quando aplicada sobre ele.
- ▶ A função f pode ser chamada de função objetivo ou função *fitness* – representa o quão adaptado o indivíduo está ao seu ambiente (ao problema). Quanto mais a solução que o indivíduo representa estiver próxima da ótima, mais adaptada está esta solução ao problema.
- ▶ Suponha que a função *fitness* retorne a adequabilidade de cada indivíduo de acordo com as informações da tabela.

No.	String	Fitness	% do Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0



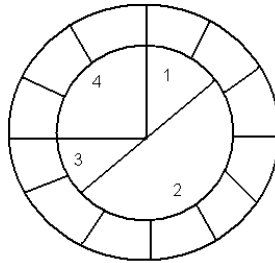
Um algoritmo genético simples

- ▶ Um algoritmo genético simples, que rende bons resultados na prática, é composto por três operadores: **Seleção (ou reprodução), Crossover e Mutação**
- ▶ A reprodução é um processo no qual indivíduos são copiados de acordo com seus valores para a função *fitness*. Também chamada de elitismo ou reprodução assexuada.
 - ▶ Copiar indivíduos de acordo com sua função *fitness* significa que indivíduos com um valor mais alto tem maior probabilidade de contribuir com descendentes nesta ou na próxima geração.
 - ▶ Este operador, naturalmente, é uma versão artificial da seleção natural, a sobrevivência Darwiniana dos indivíduos mais adaptados ao meio.
 - ▶ Na natureza, o *fitness* é determinado pela habilidade de um indivíduo a sobreviver contra predadores, pestes e outros obstáculos.
 - ▶ A função *fitness* é, na realidade, o árbitro que decide quem vive e quem morre.



Um algoritmo genético simples

- ▶ Uma implementação simples para o operador de (seleção) reprodução.
 - ▶ A roleta:
 - ▶ Cada indivíduo na população tem um “slot” na roleta de tamanho proporcional ao seu *fitness*.



- ▶ Para processar o operador de reprodução basta “rodar a roleta” e verificar qual indivíduo é selecionado.
 - ▶ Cada vez que uma descendência é requerida, a roleta é acessada para indicar o candidato.
 - ▶ Quando um indivíduo é escolhido para reproduzir, faz-se uma réplica exata dele.
 - ▶ Esse novo indivíduo comporá o conjunto de indivíduos que são candidatos a compor uma nova população (geração) para o algoritmo.

Um algoritmo genético simples

- ▶ Depois da seleção, o operador de crossover pode ser executado.
- ▶ O crossover pode ser de vários tipos, pode ocorrer sob uma determinada probabilidade de ocorrência e é também conhecido como reprodução sexuada.
- ▶ Procede-se em dois passos:
 - ▶ Membros da nova população (indivíduos reproduzidos) são escolhidos randomicamente;
 - ▶ Cada par de indivíduos sofre recombinação:
- ▶ Seleciona-se, randomicamente, uma posição K da string (k pode variar de 1 ao tamanho da string - 1);
- ▶ Dois novos indivíduos *filhos* são criados por meio da combinação dos genes dos indivíduos *pais*.



Um algoritmo genético simples

- ▶ Ilustrando:

- ▶ Pai1 = 0 1 1 0 1

- ▶ Pai2 = 1 1 0 0 0

- ▶ Posição sorteada: $k = 4$

- ▶ Pai1 = 0 1 1 0 | 1

- ▶ Pai2 = 1 1 0 0 | 0

- ▶ Filho1 = 0 1 1 0 0

- ▶ Filho2 = 1 1 0 0 1



Um algoritmo genético simples

▶ Mutação:

- ▶ É um operador ocasional que provoca alterações randômicas dos valores de algumas posições na string (em alguns genes dos indivíduos).
- ▶ Na codificação binária do problema da caixa preta, a mutação simplesmente muda de 1 para 0 o valor de um bit da string.
- ▶ A frequência da mutação deve ser baixa pois ela representa uma busca aleatória no espaço de busca.
- ▶ A implementação dela se dá como segue:
 - ▶ Se a mutação deve acontecer (de acordo com uma probabilidade de mutação)
 - ▶ Então um indivíduo é randomicamente escolhido
 - ▶ E um alelo deste indivíduo é randomicamente escolhido e
 - ▶ O valor para este alelo é randomicamente escolhido dentro do alfabeto correlato.



Um algoritmo genético simples

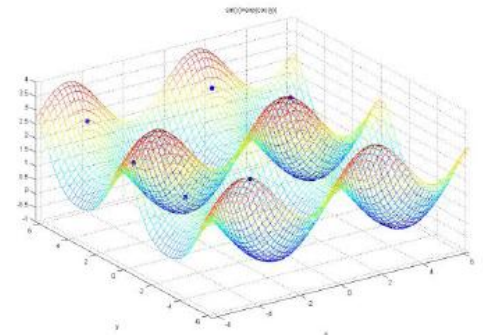
String no.	População inicial Gerada randomicamente	Valor de x Inteiro associado	f(x) x^2	%	Seleções
1	0 1 1 0 1	13	169	0,14	1
2	1 1 0 0 0	24	576	0,49	2
3	0 1 0 0 0	8	64	0,06	0
4	1 0 0 1 1	19	361	0,31	1
			1170	1,00	

Pool de reprodução	Seleção de pares	Ponto de crossover	Nova população	Valor de x	f(x)
0 1 1 0 1	2	4	0 1 1 0 0	12	144
1 1 0 0 0	1	4	1 1 0 0 1	25	625
1 1 0 0 0	4	2	1 1 0 1 1	27	729
1 0 0 1 1	3	2	1 0 0 0 0	16	256



Algoritmos Genéticos (Goldberg)

- ▶ Diferenciando dos métodos tradicionais
 - ▶ AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros;
 - ▶ AGs realizam busca a partir de uma população de pontos (soluções) e não a partir de um único ponto;
 - ▶ AGs usam informação de retorno – *feedback* da função objetivo - e não informação derivativa ou outro tipo de conhecimento auxiliar;
 - ▶ AGs usa regras de transição probabilísticas e não regras determinísticas.



Explorando ... (Goldberg)

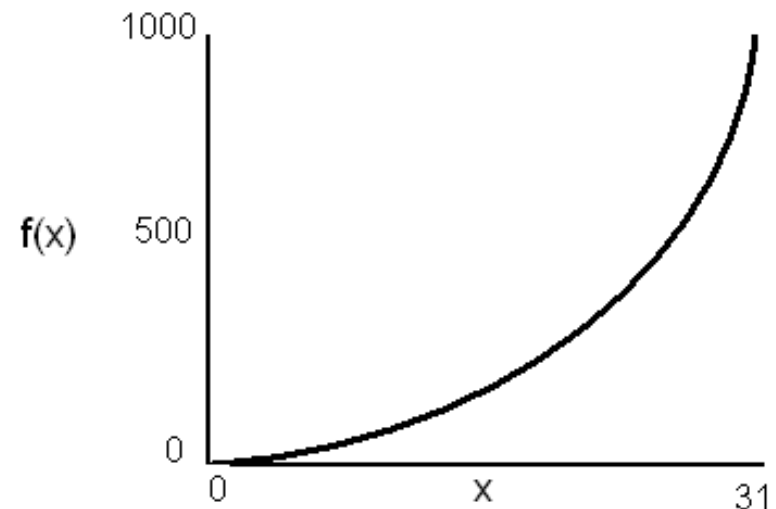
- ▶ AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.
 - ▶ O conjunto de parâmetros natural do problema deve ser codificado como uma string de comprimento finito, instanciada por meio de um alfabeto finito.

▶ Exemplo:

- ▶ Maximização de $f(x) = x^2$, no intervalo $[0, 31]$.
- ▶ Representação binária para o parâmetro
- ▶ x .

0	1	1	1	1
---	---	---	---	---

$x = 15$



Explorando (Goldberg)

- ▶ Considere a caixa preta de interruptores.



- ▶ Este problema consiste em um recurso do tipo “caixa preta” com um seqüência de 5 interruptores de entrada.
- ▶ Para cada configuração dos 5 interruptores existe um sinal de saída **f**, **matematicamente $f = f(s)$, onde s é uma configuração específica.**
- ▶ O objetivo do problema é posicionar os interruptores de maneira a obter o maior valor possível para **f**.
- ▶ Codificação dos interruptores: Uma string de comprimento 5, valorado com 0s ou 1s onde cada um dos 5 interruptores é representado por 1 se o interruptor está ligado e 0 se o interruptor está desligado.



Explorando (Goldberg)

- ▶ AGs realizam busca a partir de uma população de pontos (soluções) e não a partir de um único ponto.
 - ▶ Em muitos métodos de otimização, move-se de um estado para o próximo, ou de uma solução para a próxima, usando alguma regra de transição para determinar o próximo estado ou situação (ponto).
 - ▶ Este método ponto-a-ponto é perigoso porque pode cair em mínimos ou máximos locais.
 - ▶ Os AGs trabalham com uma população de soluções, buscando o máximo ou mínimo de forma paralela.
- ▶ No nosso problema da caixa preta de interruptores, um método clássico começaria com a determinada configuração de interruptores, aplicaria a regra de transição e geraria a nova configuração.
 - ▶ Um algoritmo genético começaria com uma população de strings e geraria sucessivas populações de strings.
- ▶ Por exemplo:
 - ▶ Uma população inicial de 4 strings seria gerada randomicamente;
 - ▶ Em seguida, a partir desta, sucessivas populações poderiam ser geradas



Explorando (Goldberg)

- ▶ AGs usam informação de retorno – *feedback* da função objetivo - e não informação derivativa ou outro tipo de conhecimento auxiliar;
 - ▶ Muitas técnicas exigem muita informação auxiliar sobre o problema para trabalhar adequadamente.
 - ▶ Técnicas baseadas no gradiente, por exemplo, precisam de derivadas para serem capazes de “subir um morro”.
 - ▶ Técnicas gulosas requerem acesso à maioria, senão a todos, os parâmetros informacionais –por exemplo, as possibilidades a serem avaliadas.
- ▶ Algoritmos genéticos não exigem informações diretas sobre o problema (eles são ditos cegos) **eles requerem apenas o *feedback*.**

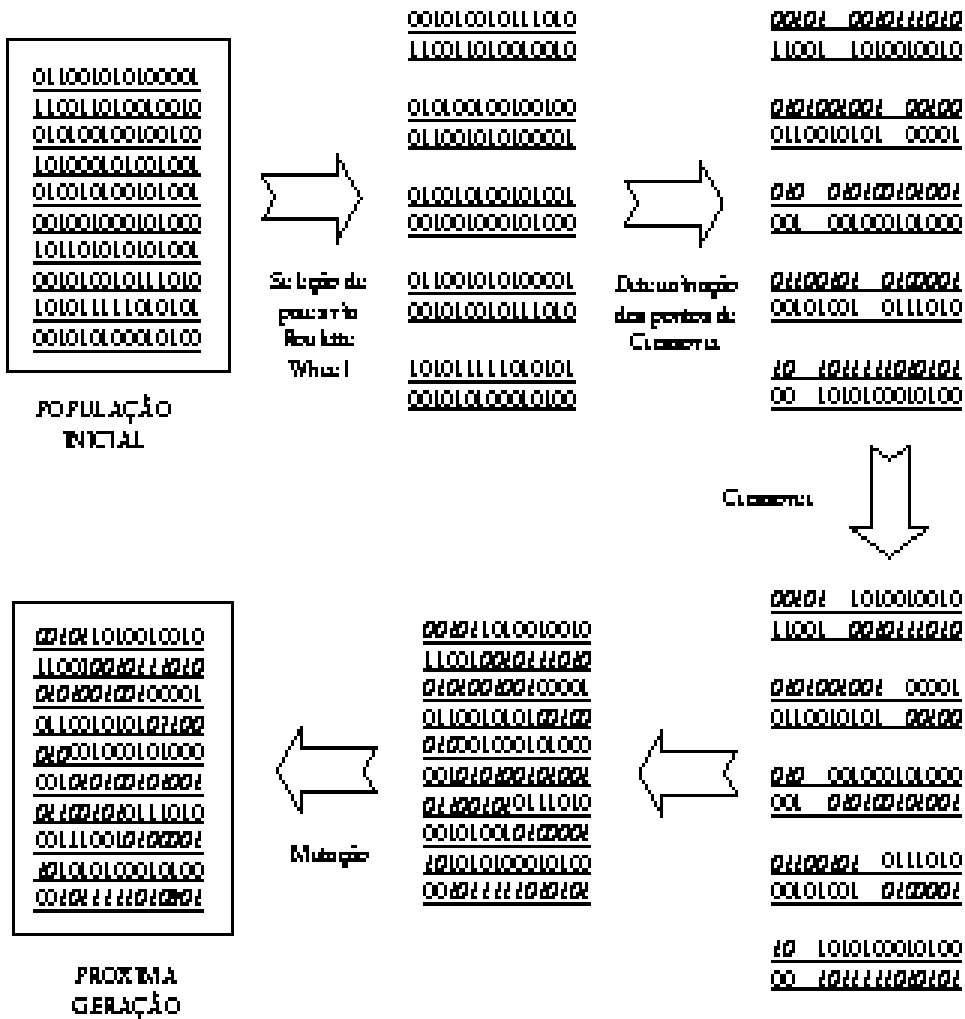


Explorando (Goldberg)

- ▶ AGs usa regras de transição probabilísticas e não regras determinísticas.
- ▶ A escolha por gerar um novo estado é feita baseada em probabilidades que ajudar o algoritmo a caminha por regiões promissoras no espaço de busca.



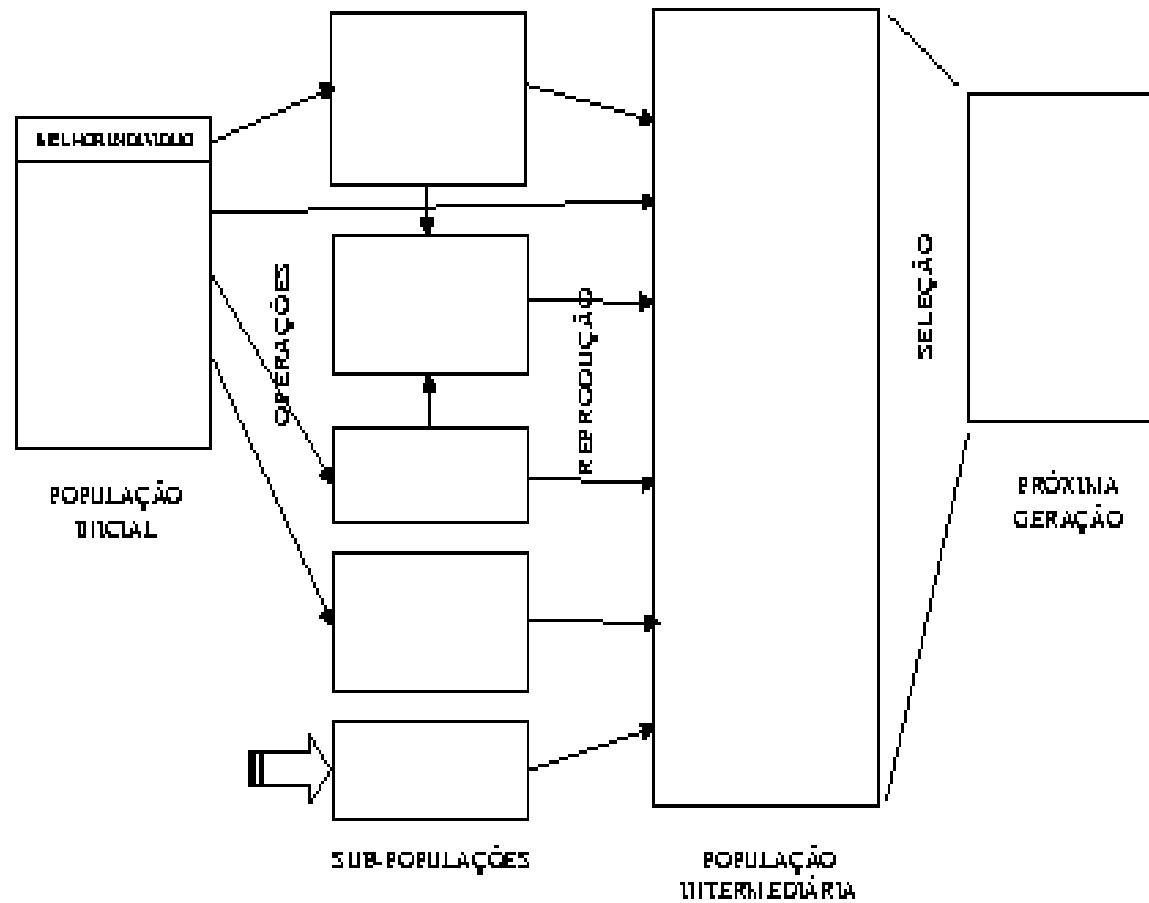
Relembrando: Algoritmo Genético Clássico



Algoritmo Genético Modificado

- ▶ Geração de sub-populações por meio da aplicação de operadores genéticos e outros operadores sobre membros da geração
- ▶ Avaliação e ordenação da população intermediária
- ▶ Seleção para nova geração

Algoritmo Genético Modificado



Algoritmo Genético Clássico

- ▶ Problemas com algoritmo clássico
 - ▶ Política de reprodução/seleção permite a perda do melhor indivíduo
 - ▶ Posição dos genes no cromossomo influi na probabilidade de permanecerem no mesmo cromossomo após crossover
 - ▶ Dificuldades na codificação binária de números reais

Exemplos de operações e seleção

- ▶ Crossover simples entre indivíduos aleatórios (Roulette Wheel)
- ▶ Crossover simples entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo
- ▶ Crossover uniforme entre indivíduos aleatórios (Roulette Wheel)
- ▶ Crossover uniforme entre indivíduos aleatório (Roulette Wheel) e melhor indivíduo
- ▶ Mutação de indivíduos aleatórios (Roulette Wheel) da população Inicial e/ou Sub-populações
 - ▶ Mutação aleatória
 - ▶ Mutação indutiva (somente para codificação real)
- ▶ Mutação do melhor indivíduo
 - ▶ Mutação aleatória
 - ▶ Mutação indutiva (somente para codificação real)

Exemplos de operações e seleção

- ▶ Crossover uniforme entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo
- ▶ É escolhido um indivíduo da população inicial e / ou sub populações, aleatoriamente ou por meio de Roulette Wheel e o melhor da população inicial e/ou sub populações, determina-se a porcentagens de genes que serão trocados, procede-se a um sorteio de alelos na quantidade determinada efetuando o crossover somente dos alelos envolvidos.

Exemplos de operações de seleção (ou gera uma máscara – slide anteriores)



Pai 1



Pai 2



Filho 1



Filho 2



Exemplos de mecanismo de seleção

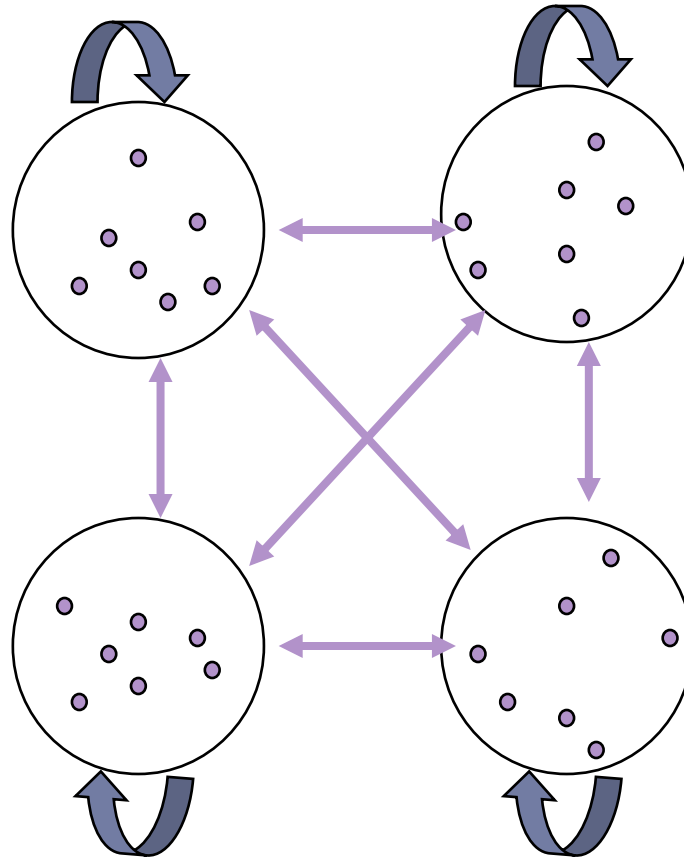
- ▶ **Seleção Elitista** – são selecionados os N melhores indivíduos da população intermediária
- ▶ **Seleção aleatória** – são selecionados aleatoriamente N indivíduos da população intermediária
 - ▶ Salvacionista – seleciona-se o melhor indivíduo e os outros aleatoriamente
 - ▶ Não salvacionista – seleciona aleatoriamente todos os indivíduos
 - ▶ Distribuição uniforme – todos tem a mesma chance
 - ▶ Distribuição proporcional
 - Roulette wheel por fitness
 - Roulette wheel por fitness com sigma scaling
 - Seleção de boltzman
 - Seleção por rank

Outros mecanismos de seleção

- ▶ **Seleção por diversidade** – são selecionados os indivíduos mais diversos na população intermediária, a partir do melhor indivíduo
- ▶ **Seleção bi-classista** – são selecionados os $P\%$ melhores indivíduos e os $(100-P)\%$ piores indivíduos
- ▶ **Seleção por torneio** – dois indivíduos são escolhidos aleatoriamente. Um número aleatório r entre 0 e 1 é gerado. Se $r < k$ (parâmetro) o melhor dos indivíduos é escolhido. Senão o outro é escolhido
- ▶ **Seleção Steady-State** – população original é mantida, com exceção de alguns poucos indivíduos menos adaptados. A taxa de sobreposição desta população é definida por uma constante, que representa o percentual desta substituído a cada geração.

Algoritmos genéticos paralelos

- K Populações são iniciadas evoluem paralelamente



- A cada n-ésima geração, as populações trocam indivíduos

Porque converge?

- ▶ **Esquemas**
 - ▶ “sub-partes” comuns recorrentes
- ▶ **Teorema dos esquemas**
 - ▶ o número de esquemas bem adaptados cresce exponencialmente
- ▶
- ▶ **Building-blocks hypothesis:**
 - ▶ a otimalidade é obtida por justaposição de pequenos esquemas altamente adaptados



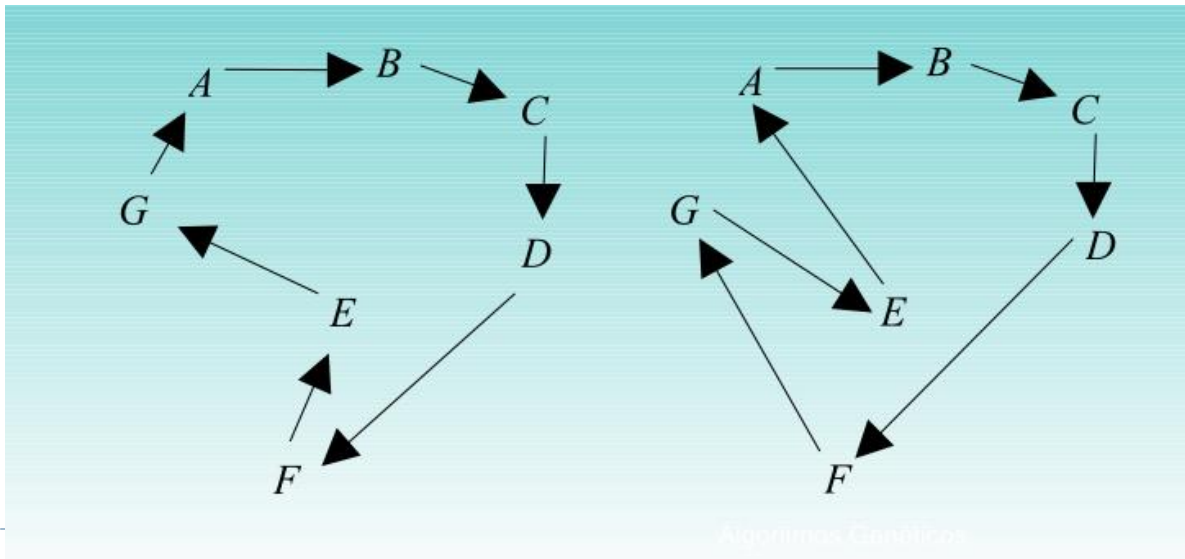
Caixeiro Viajante



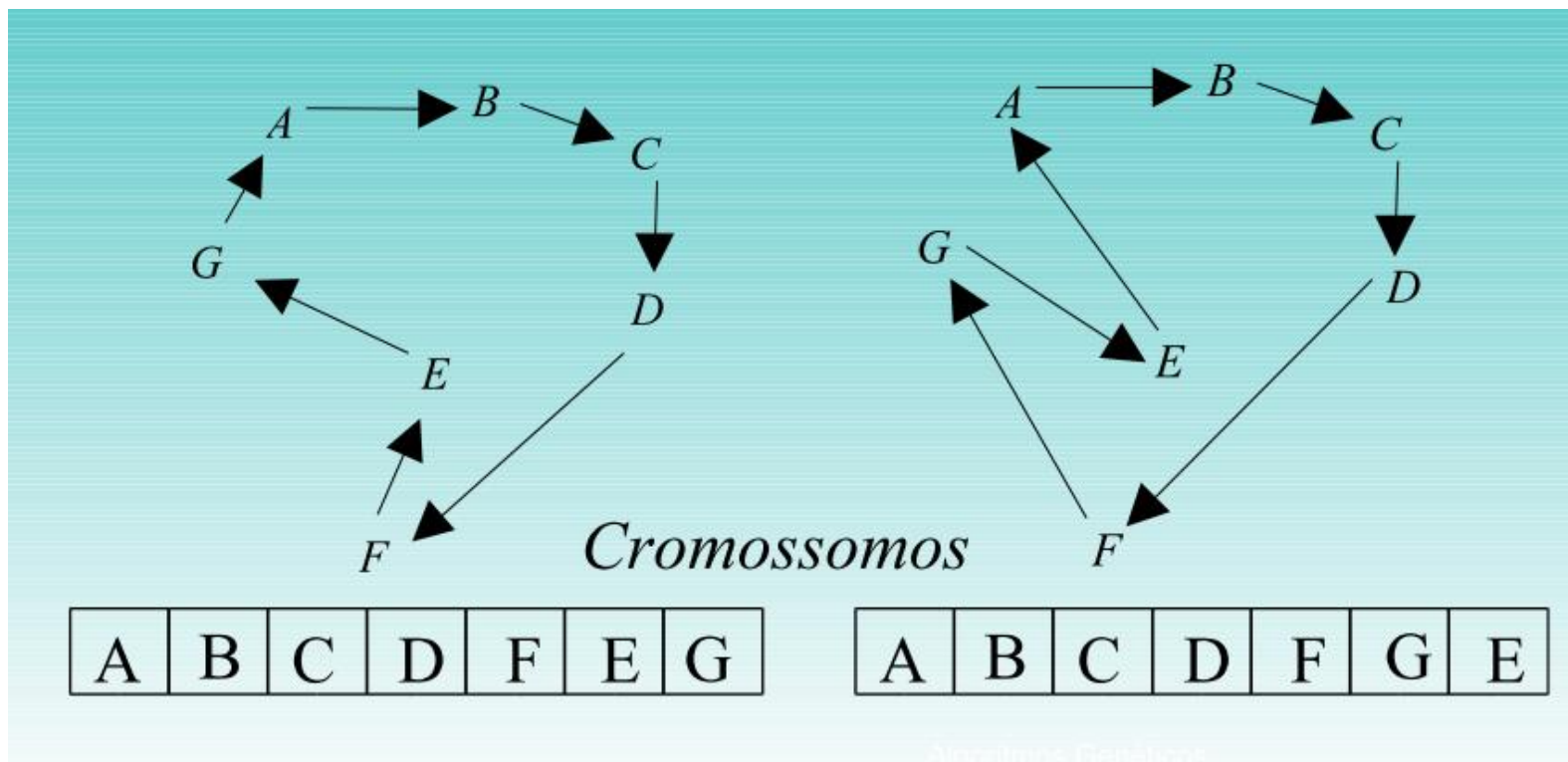
Caixeiro Viajante

O Problema

- ▶ Dado um número de cidades, encontrar o caminho mais curto passando por todas as cidades uma única vez
- ▶ Função Objetivo = Distância Total Percorrida



Representação



Crossover

- Crossover baseado em posição
 - São selecionadas n cidades. Cada filho mantém a posição das cidades selecionadas de um pai

<i>pai1</i>	A	B	C	D	F	E	G
<i>pai2</i>	C	E	G	A	D	F	B
...							
<i>filho1</i>	B	E	C	A	D	F	G
<i>filho2</i>	C	B	E	D	F	G	A

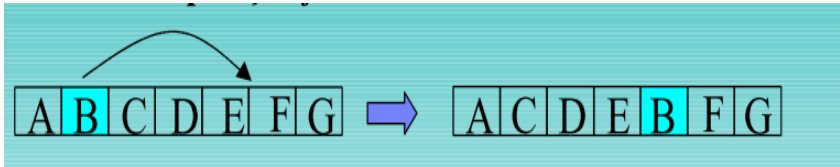
Crossover

- Crossover baseado em ordem
 - São selecionadas n cidades. Cada filho herda a ordem das cidades selecionadas de um pai

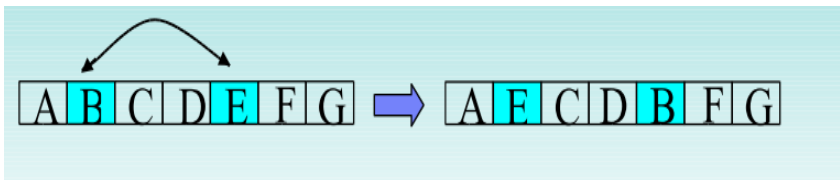
<i>pai1</i>	A	B	C	D	F	E	G
<i>pai2</i>	C	E	G	A	D	F	B
<i>filho1</i>	A	D	C	F	B	E	G
<i>filho2</i>	C	A	G	D	E	F	B

Mutação

- Mutação baseada na troca de posição de uma cidade



- Mutação baseada na troca da ordem de duas cidades



	AG	EE	PE	PG
Representação	Cadeias binárias/reais	Vetores reais	Vetores reais	arvores
Auto-adaptação	nenhuma	Desvio padrão e co-variâncias	Desvio padrão e coeficientes de co-relação	nenhum
O fitness é	Valor escalonado da função objetiva	Valor da função objetiva	Valor escalonado da função objetiva	Valor escalonado da função objetiva
Mutação	Operador secundário	Operador principal	Único operador	Um dos operador
Recombinação	Principal operador	Diferentes variações	nenhuma	Um dos operadores
Seleção	probabilística	determinística	probabilística	probabilística



Exemplo

- ▶ <http://www.obitko.com/tutorials/genetic-algorithms/portuguese/example-function-minimum.php>
- ▶ <http://www.obitko.com/tutorials/genetic-algorithms/portuguese/example-3d-function.php>
- ▶ <http://www.obitko.com/tutorials/genetic-algorithms/portuguese/tsp-example.php>



Bibliografia

- ▶ RUSSEL, S. e NORVIG, P. Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.
 - ▶ Pgs: 63-68; 70-71; 95-99; 104-106; 113-117.
- ▶ GOLDBERG, D. Genetic Algorithms. Addison Wesley, 1988
- ▶ LINDEN, R. Algoritmos Genéticos: uma importante ferramenta da Inteligência Computacional. Ed. Brasport, 2006.

