Buscas - Inteligência Artificial

Profa. Dra. Sarajane Marques Peres

10 de maio de 2018

Disciplina: Inteligência Artificial
Bacharelado em Sistemas de Informação
http://www.each.usp.br/si

Material desenvolvido baseado em:

- Russel, S.; Norvig, P. Inteligência Artificial. 2a. 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, 5, 6, 7, e 9.

Resolução de problemas por meio de busca

Busca

Um algoritmo de busca recebe um problema como entrada e retorna uma solução sob a forma de uma sequência de ações. Ainda, há problemas em que a solução é um estado.

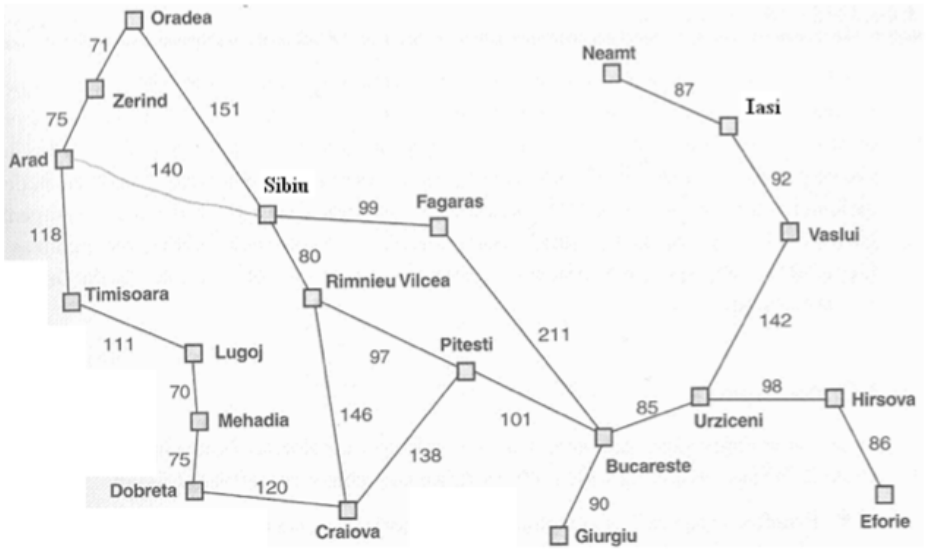
Definição de um problema (representação):

- 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 de um 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

Preciso chegar em Bucareste

A solução é o caminho entre uma cidade e outra. A busca retorna O CAMINHO à medida que o encontra, ou o caminho completo em algum momento da execução do procedimento.



- Estado inicial: Estou em Arad.
- Ações possíveis: De Arad, sigo por uma estrada até a próxima cidade = Estou na próxima cidade.
- Teste de objetivo: Estou em Bucareste?
- Função custo: Quanto tempo gastei para chegar onde estou?

Alguns conceitos

Função sucessor

Dado um estado particular x, SUCESSOR(x) retorna um conjunto de pares ordenados $<$*ação*,*sucessor*$>$, em que cada ação é uma das ações válidas no estado x e cada sucessor é um estado que pode ser alcançado a partir de x aplicando-se a ação.

- EM(Arad)
- {<IR(Sibiu), EM(Sibiu)>, <IR(Timissoara), EM(Timissoara)>, <IR(Zerind), EM(Zerind)>}

Espaço de Estados

O estado inicial e a função SUCESSOR definem o espaço de estados do problema - o conjunto de todos os estados acessíveis a partir do estado inicial.

Função de custo

Atribui um custo numérico à solução (obtida até o momento).

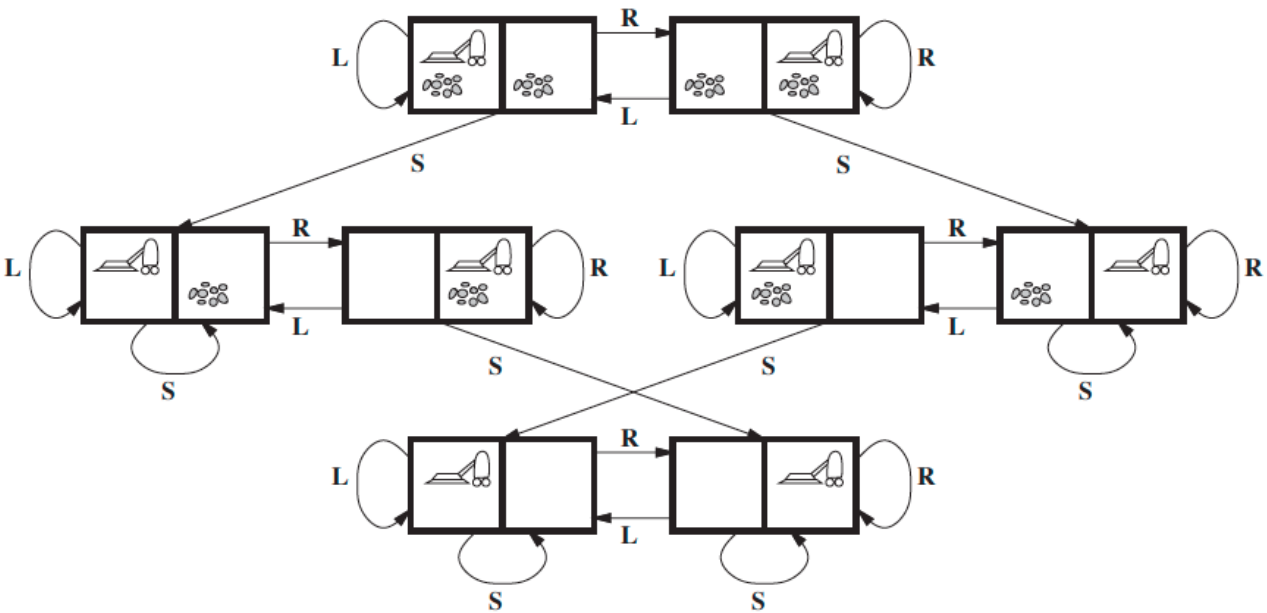
Custo do passo

É o custo de adotar a ação a para ir do estado x para o estado y.

Exemplos de problemas

Aspirador de Pó

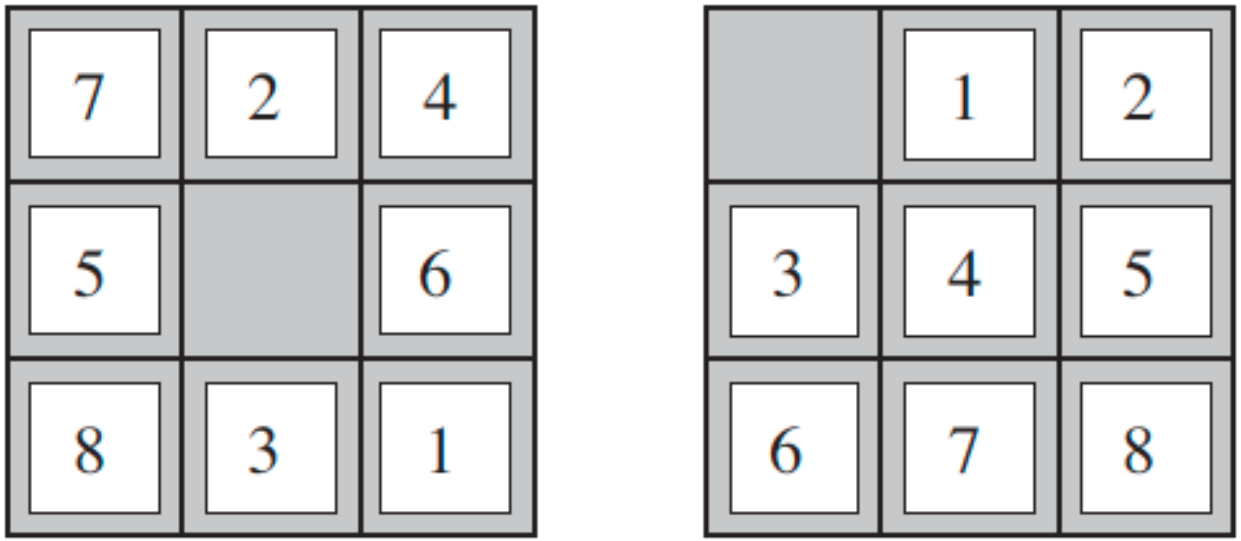
O mundo do aspirador de pó tem apenas dois locais. O agente aspirador de pó percebe em que quadrado está e se existe sujeira no quadrado. Ele pode optar por mover-se para a esquerda, mover-se para a direita, aspirar a sujeira ou não fazer nada. A função dele é: se o quadrado estiver sujo, então aspirar, caso contrário mover-se para o outro quadrado.



Exemplos de problemas

Quebra cabeça de 8

Consiste em um tabuleiro de 3 x 3, com oito peças numeradas e um espaço vazio. Uma peça adjacente ao espaço vazio pode deslizar para o espaço. O objetivo é alcancar um estado objetivo especificado.

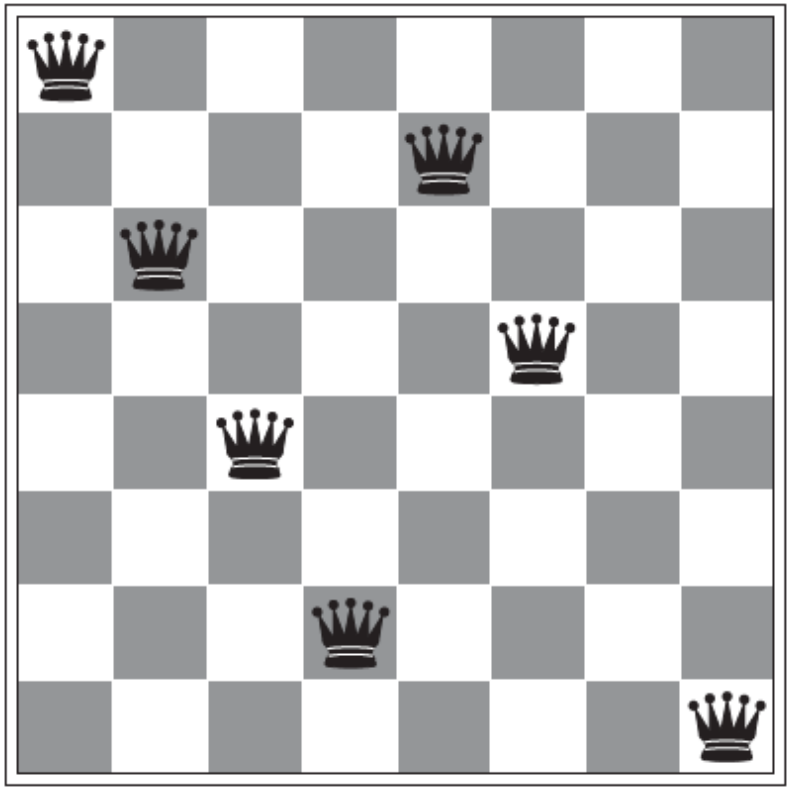


À esquerda está um estado inicial possível. À direita está o estado objetivo.

Exemplos de problemas

8 rainhas

Posicionar as oito rainhas em um tabuleiro de xadrez de tal forma que nenhuma rainha ataque qualquer outra. Uma rainha ataca qualquer peça situada na mesma linha, coluna ou diagonal. Neste caso a solução não é um caminho, mas sim um estado.



Problemas do mundo real

- Problema de roteamento (exemplo – viagens aéreas)
  - Estados: Cada um é representado por uma posição (por exemplo, um aeroporto) e pela hora atual.
  - Estado inicial: É especificado em uma instância do problema.
  - Função sucessor: Retorna os estados resultantes de tomar qualquer vôo programado que parte depois da hora atual somada ao tempo de trânsito no aeroporto.
  - Teste de objetivo: Estamos no destino após algum tempo previamente especificado?
  - Custo de caminho: Depende do custo monetário, do tempo de espera, do tempo de vôo, dos procedimentos alfandegários e de imigração, da qualidade da poltrona, da hora do dia, do tipo de aeronave, dos prêmios por milhagem .....
- Problema de Tour / Problema do Caixeiro Viajante
- Layout de VLSI

Em busca de soluções - Busca em Árvore

Uma solução tradicional ....

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

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

repita

se não existe nenhum candidato para expansão

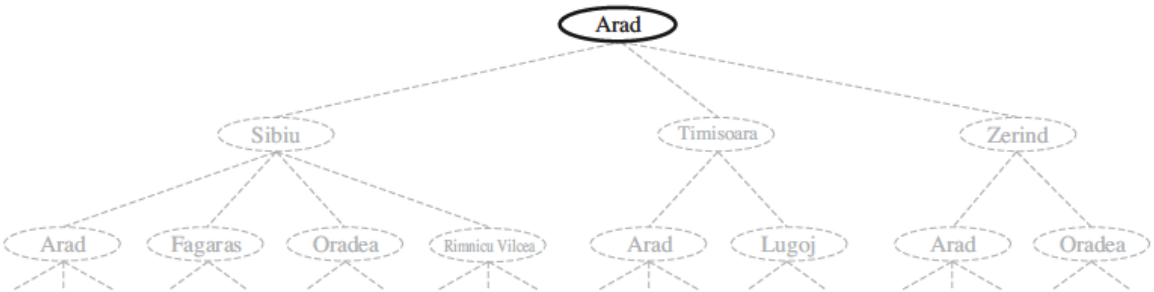
então retornar falha; // (exit)

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

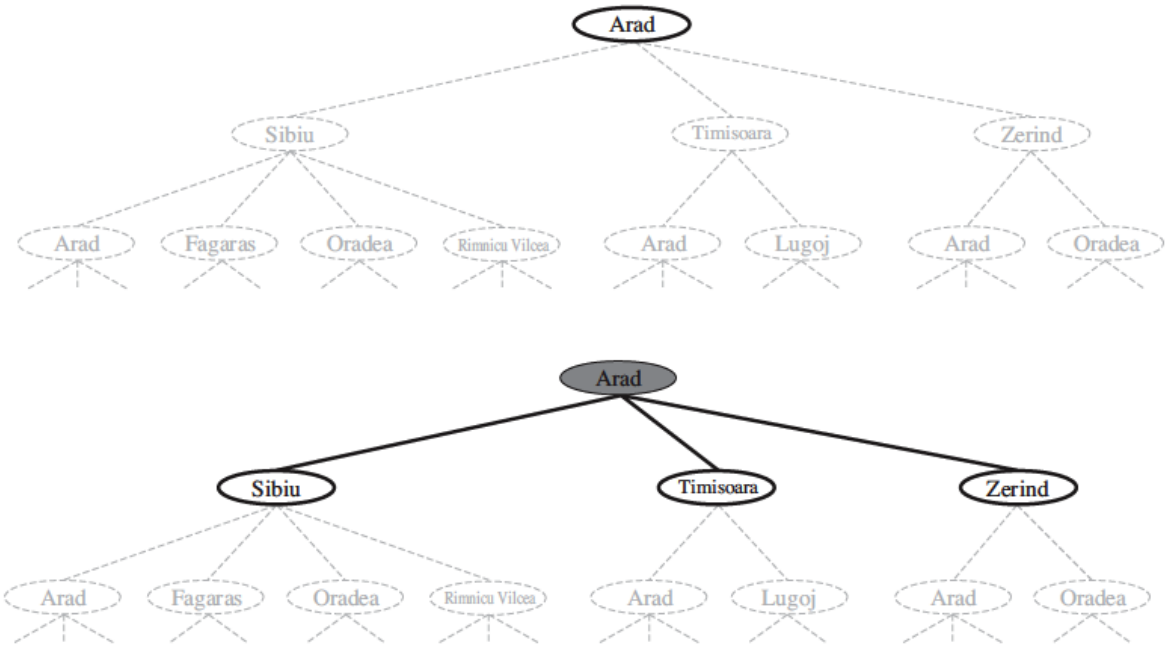
se o nó contém um ***estado objetivo*** então retornar a solução correspondente

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

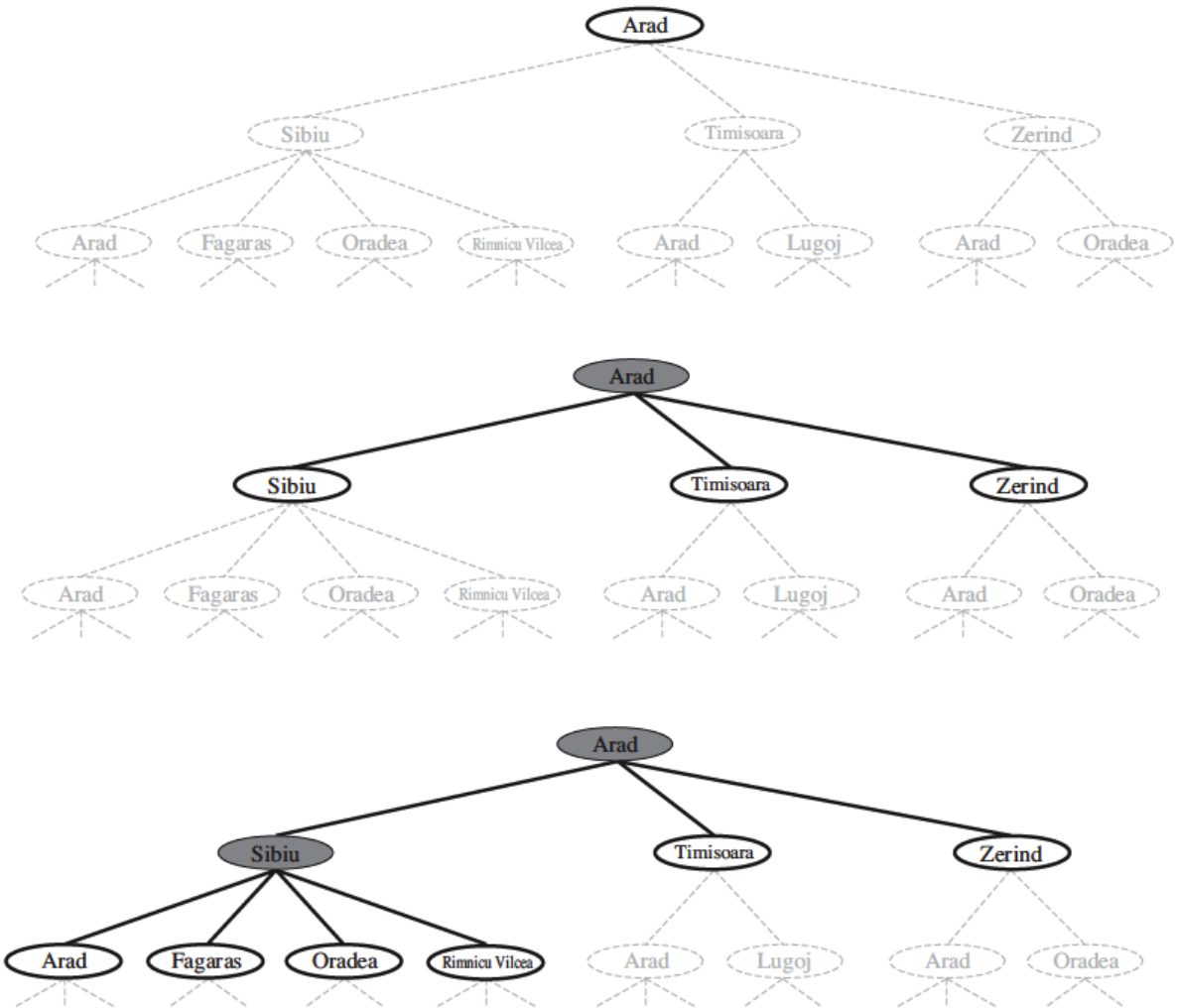
Preciso chegar em Bucareste



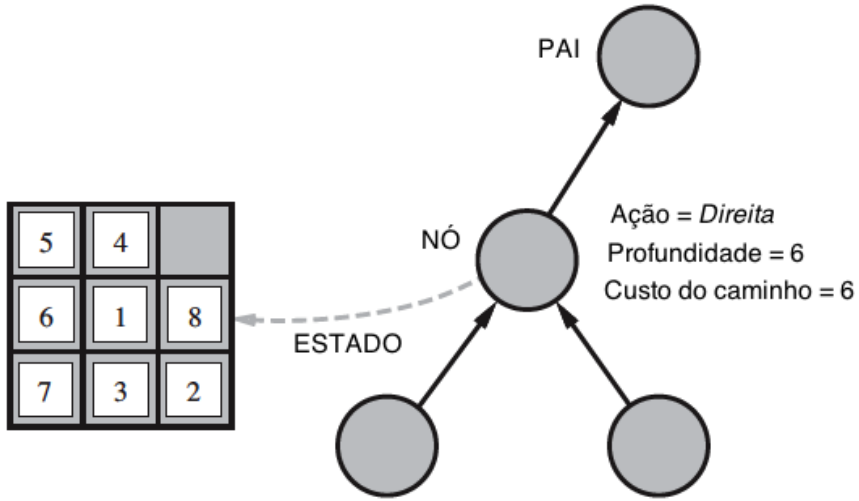
Preciso chegar em Bucareste



Preciso chegar em Bucareste



Tipicamente ... um nó



- Estado: o estado no espaço de estados a que o nó corresponde
- Nó-pai: o nó na árvore de busca que gerou esse nó
- Ação: a ação que foi aplicada ao pai para gerar o nó
- Custo do caminho: o custo $g(n)$ do caminho desde o estado inicial até o nó
- Profundidade: O número de passos ao longo do caminho, desde o estado inicial

Medição de desempenho da resolução de problemas

Trata-se do desempenho do algoritmo usado na resolução do problema:

- Completeza: o algoritmo oferece garantia de encontrar uma solução quando ela existir?
- Otimização: a estratégia encontra a solução ótima?
- Complexidade de tempo: quanto tempo ele leva para encontrar uma solução?
- Complexidade de espaço: Quanta memória é necessária para executar a busca?

Estratégias de busca sem informação

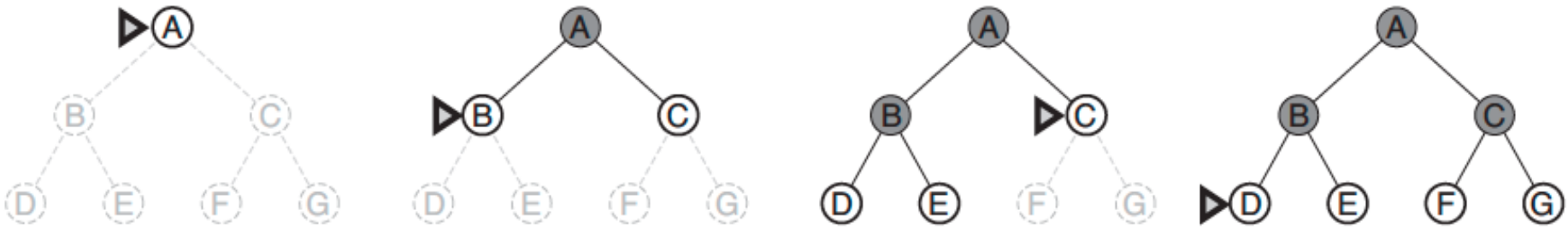
A **busca sem informação** também é conhecida como **busca cega**. Essas estratégias não possuem nenhuma informação adicional sobre os estados além daquelas fornecidas na definição do problema. Tudo o que elas podem fazer é gerar sucessores e distinguir um estado objetivo de um estado não objetivo.

- Busca em extensão
- Busca de custo uniforme
- Busca em profundidade
- Busca em profundidade limitada
- Busca de aprofundamento iterativo em profundidade

As estratégias que sabem se um estado não-objetivo é mais promissor do que outro são chamadas de **buscas com informação**, ou **buscas heurísticas**.

Busca em extensão e Busca de custo uniforme

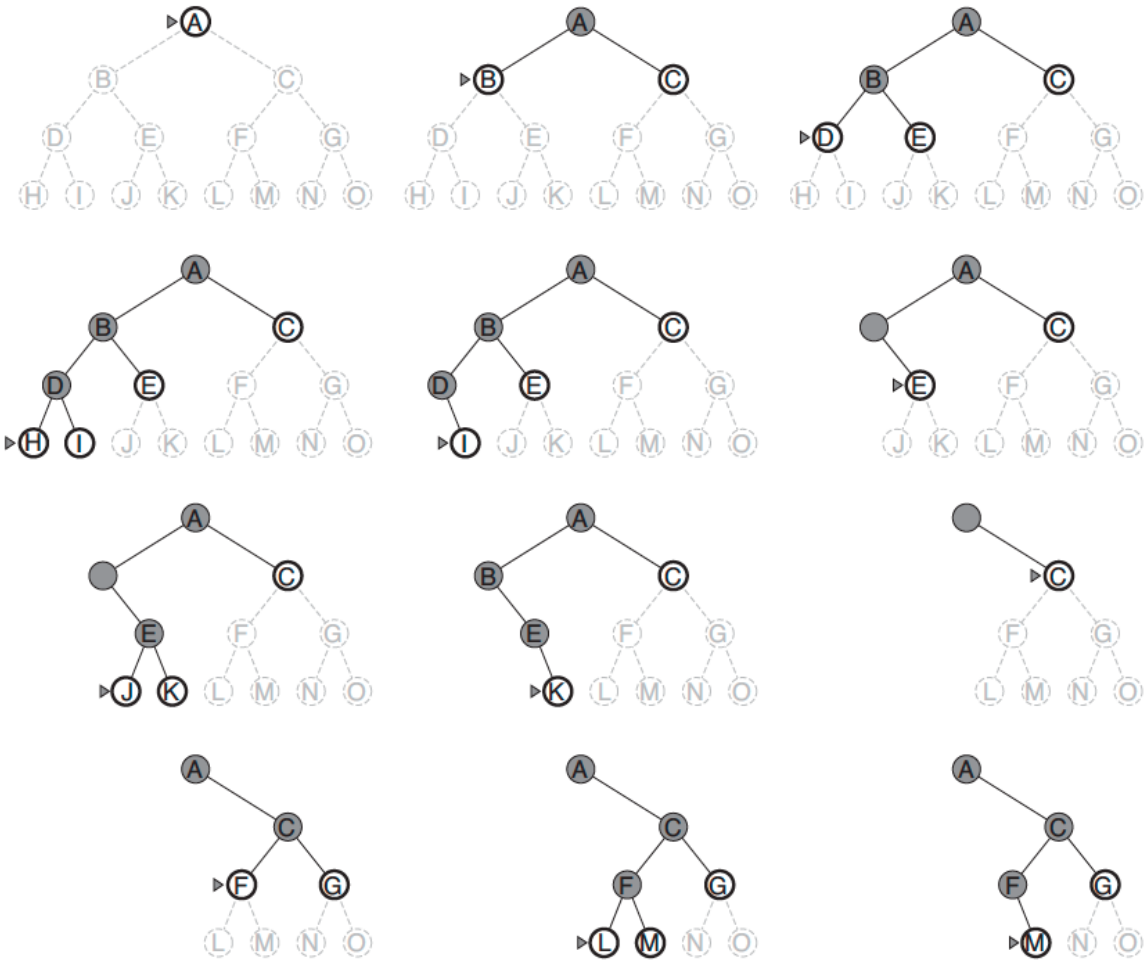
Busca em extensão é uma estratégia simples em que o nó raíz é expandido primeiro, em seguida todos os sucessores do nó raíz são expandidos, depois os sucessores desses nós e assim por diante. É uma estratégia FIFO.



Na **busca de custo uniforme** expande-se o nó *n* com o caminho de custo mais baixo. Estabelece-se uma fila de prioridade de acordo com o custo de cada caminho. Se todos os "custos de passos" forem iguais, essa busca será idêntica à busca em extensão.

Busca em profundidade

A **busca em profundidade** sempre expande o nó mais profundo na borda[1] da árvore de busca.



[1] Nós gerados mais ainda não espandidos.

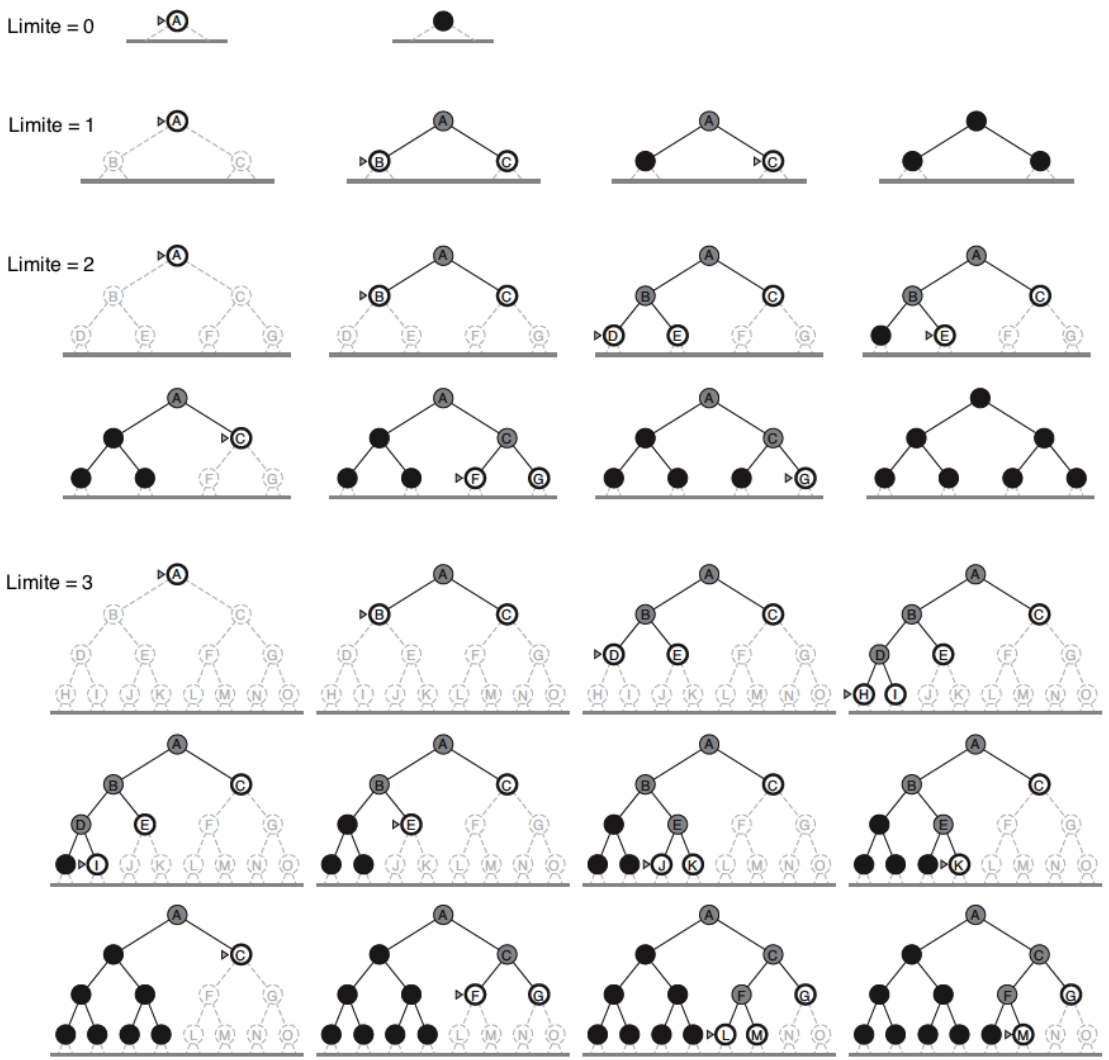
Busca em profunidade e Busca em profundidade limitada

Uma variação da busca em profunidade é a *busca com retrocesso*. Na busca com retrocesso, apenas um sucessor é gerado de cada vez, em lugar de todos os sucessores; cada nó parcialmente expandido memoriza o sucessor que deve gerar em seguida. Utiliza menos memória.

Na busca em **profundidade limitada**, nós na profundidade l são tratados como se não tivessem sucessores. O limite da profundidade resolve o problema de caminhos infinitos. Pode não chegar na solução.

A **busca de aprofundamento iterativo** é uma estratégia geral, usada com frequência em combinação com a busca **em profundidade**, que encontra o melhor limite de profundidade. Ela faz isso aumentando gradualmente o limite, até encontrar um objetivo. O aprofundamento iterativo combina os benefícios da busca em profundidade e da busca em extensão.

Busca de aprofundamento iterativo em profundidade



Busca com informação

MAFALDA - Joaquín Salvador Lavado

- 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 selecionado para expansão porque a avaliação mede a distância dele até o objetivo (de alguma forma - usando por exemplo, algum tipo de custo).

Busca com informação

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 da busca com informação - HEURÍSTICA

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.
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.

Função Heurísica

Geralmente denotada por $h(n)$

- custo **estimado** do caminho mais econômico do nó n até um nó objetivo
- avaliação do próprio nó como uma solução para o problema

Funções arbitrárias específicas para um problema, com uma restrição: se n é o nó objetivo então $h(n) = 0$

Seu valor depende do estado do nó que lhe é passado como entrada, quando é o custo do caminho (do processo).

Busca Gulosa – busca guiada pela melhor escolha

- Estratégia que expande o nó mais próxima à meta, na suposição de que isso provavelmente levará a uma solução rápida e eficiente.
- Avalia os nós usando apenas a função heurística: $f(n) = h(n)$, onde $f(n)$ é chamada de função de avaliação.

Considerando 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.

Mapa rodoviário da Romênia - com custos dos caminhos

Valores de h_{DLR}

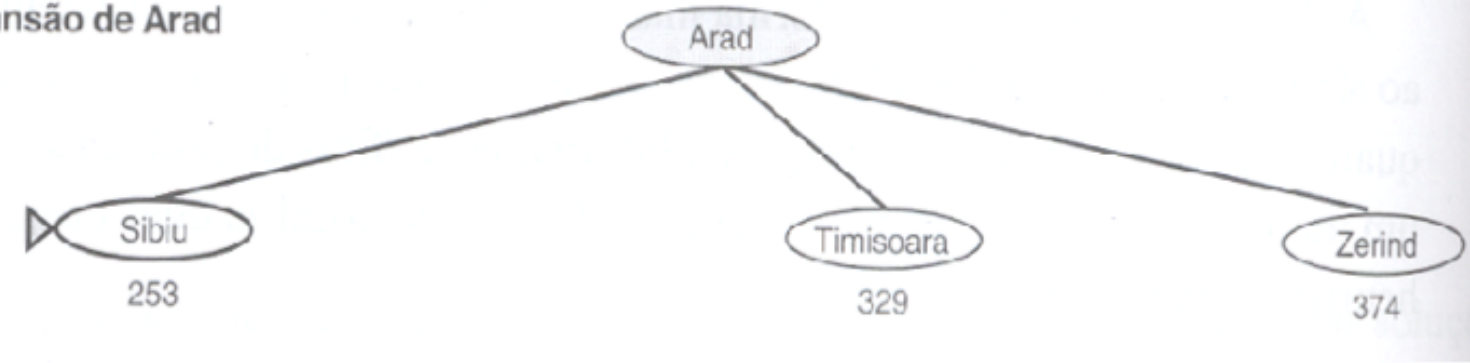
Arad	366	Mehadia	241
Bucareste	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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 reais e que, portanto, é uma heurística útil.

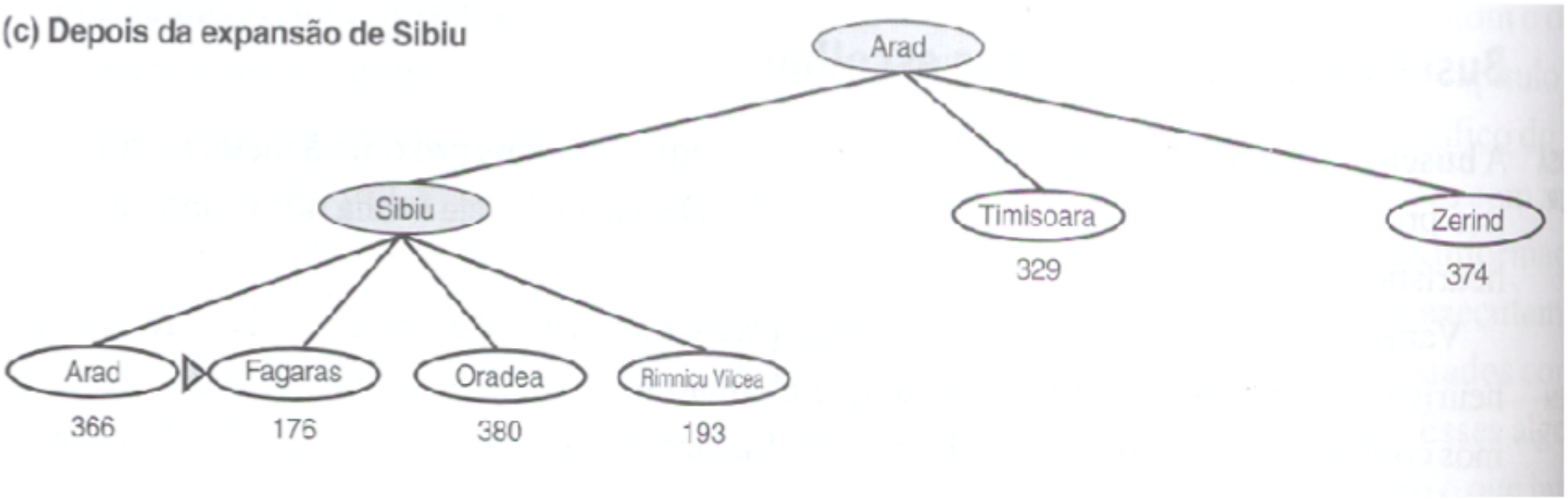
O processo executado pela busca gulosa

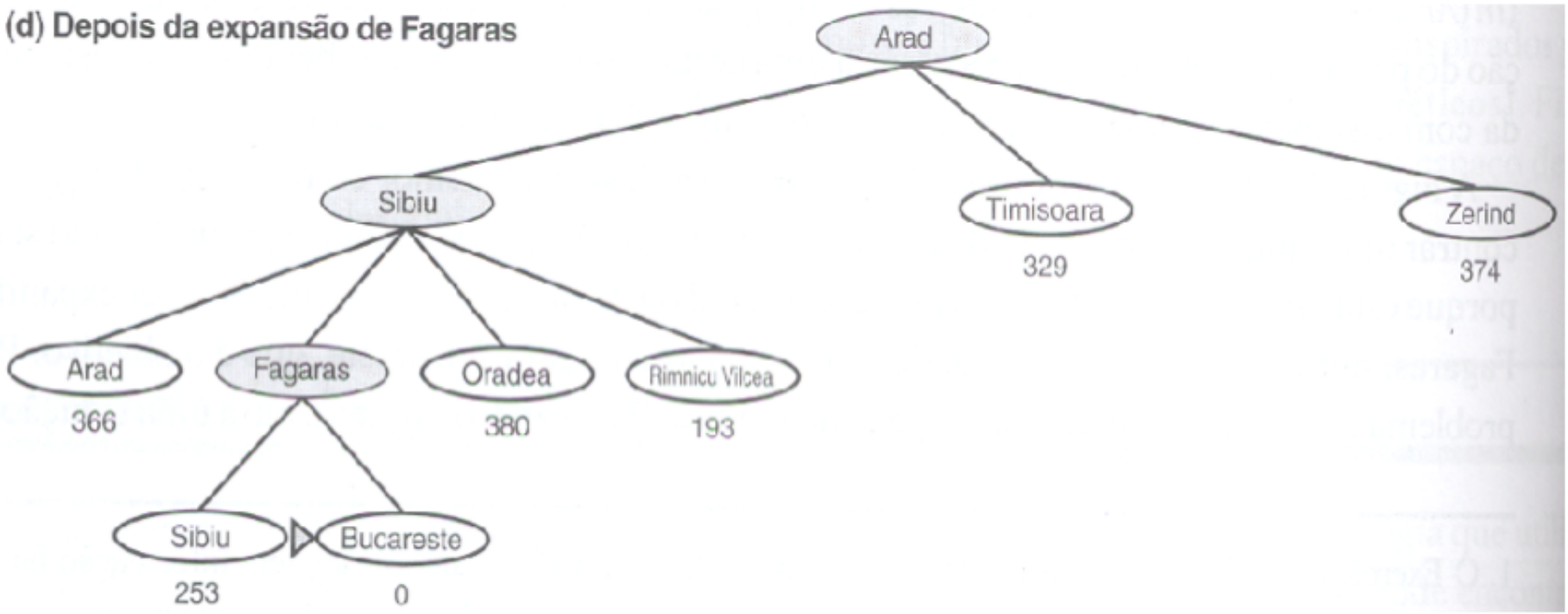
(a) O estado inicial

(b) Depois da expansão de Arad



(c) Depois da expansão de Sibiu





A heurística não é ótima

O caminho até Bucareste 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 combinado $g(n)$ e $h(n)$ gerando uma função de avaliação $f(n)$.
- A heurística usada no algoritmo deve ser admissível.

Função de avaliação $f(n)$

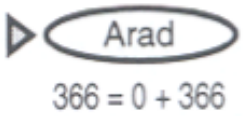
$f(n) = g(n) + h(n)$ onde $g(n)$ é o custo para alcançar cada nó, e $h(n)$ é o custo para ir do nó atual até o objetivo.

Heurística admissível

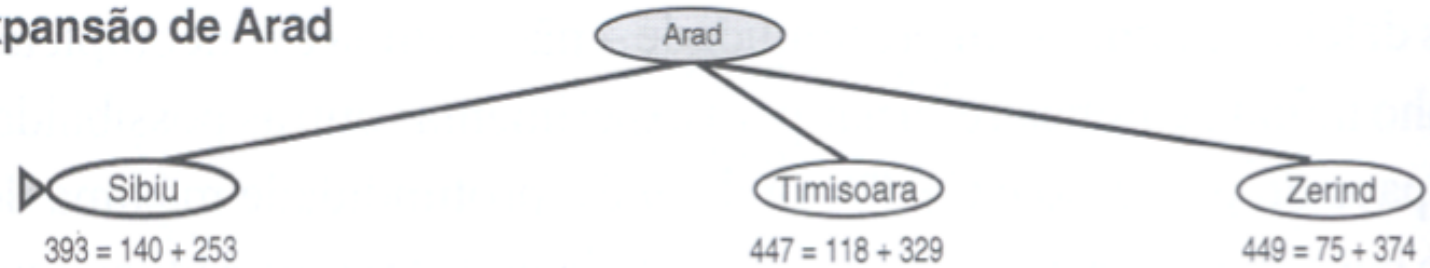
$h(n)$ é admissível se ela nunca superestimar o custo para alcançar o objetivo!!!

O processo executado por uma busca A^*

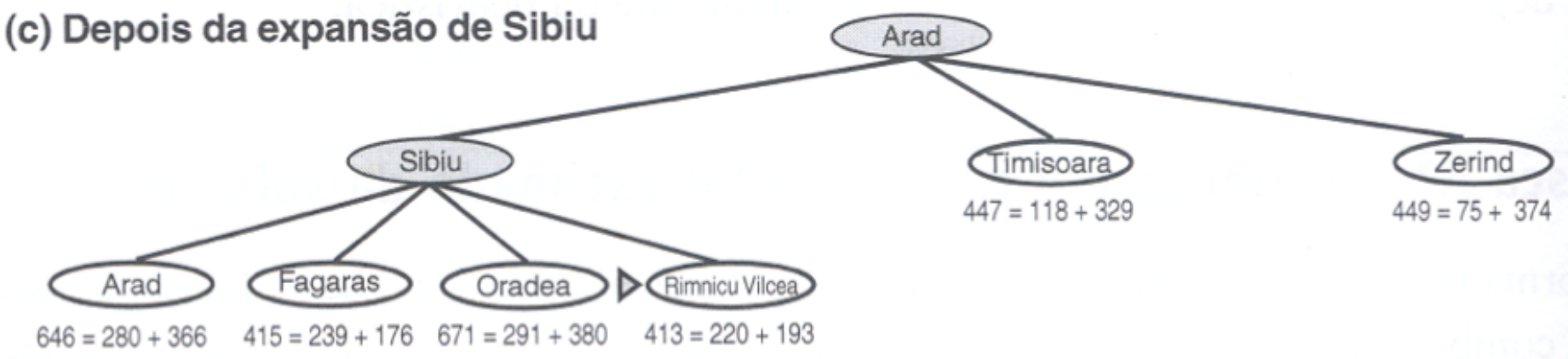
(a) O estado inicial



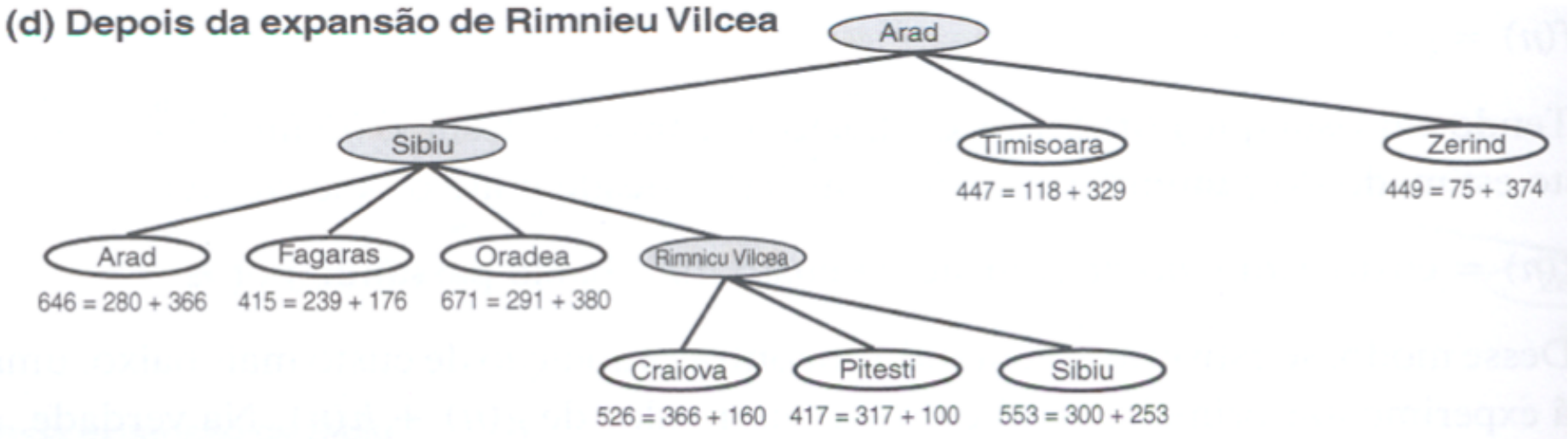
(b) Depois da expansão de Arad



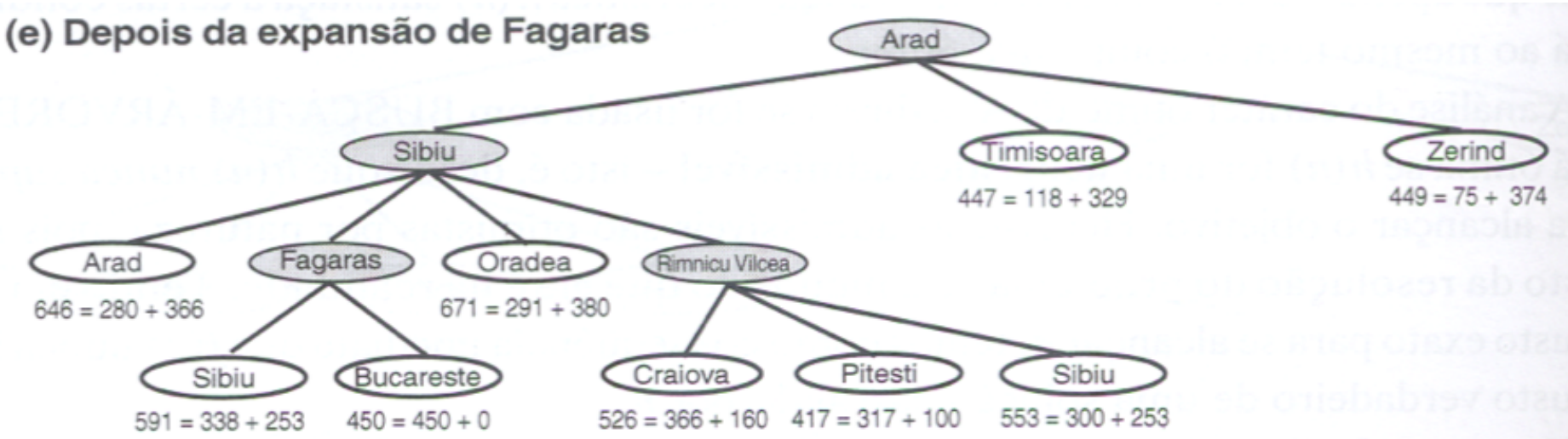
O processo executado por uma busca A^*



O processo executado por uma busca A^*

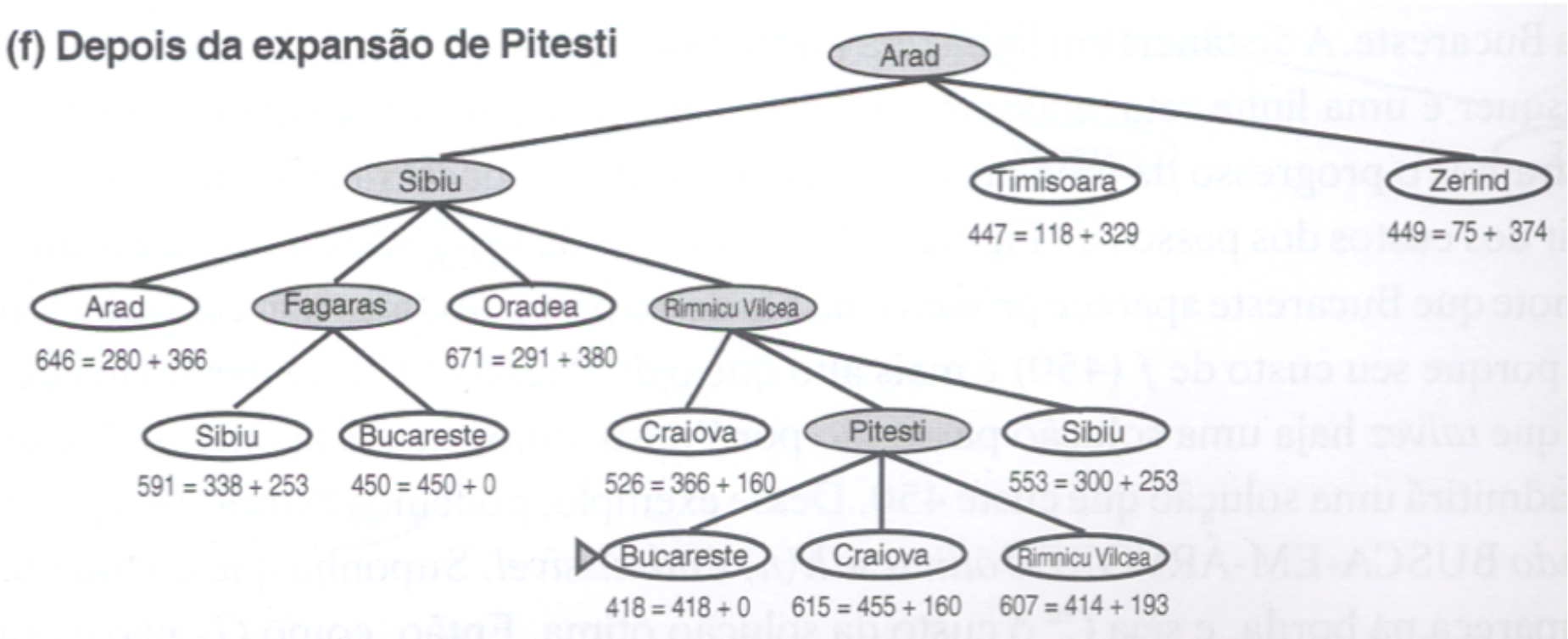


O processo executado por uma busca A^*



A^* permite voltar!!!!!!

O processo executado por uma busca A^*



A^* usando BUSCA EM ÁRVORE é ótima se $h(n)$ for admissível!!!!!!

Funções Heurísticas

Considere o problema do quebra-cabeças de 8 peças?

Estado inicial

Estado objetivo

Objetivo

Deslizar um bloco no sentido horizontal ou vertical para o espaço vazio, até que a configuração coincida com a configuração objetivo.

Fator de ramificação

O fato 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 possíveis

Funções Heurísticas

Se quisermos descobrir soluções usando o A^*, precisaremos de uma função heurística que nunca superestime "o número de passos até o objetivo".

Relaxando o problema

1. 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, ...
2. se as regras do quebra-cabeça fossem alteradas de forma que um bloco pudesse se mover para um quadrado em qualquer direção ....

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

Funções heurísticas

Candidatas para o quebra-cabeça de 8:

1. 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.
2. 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 a 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: $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$.

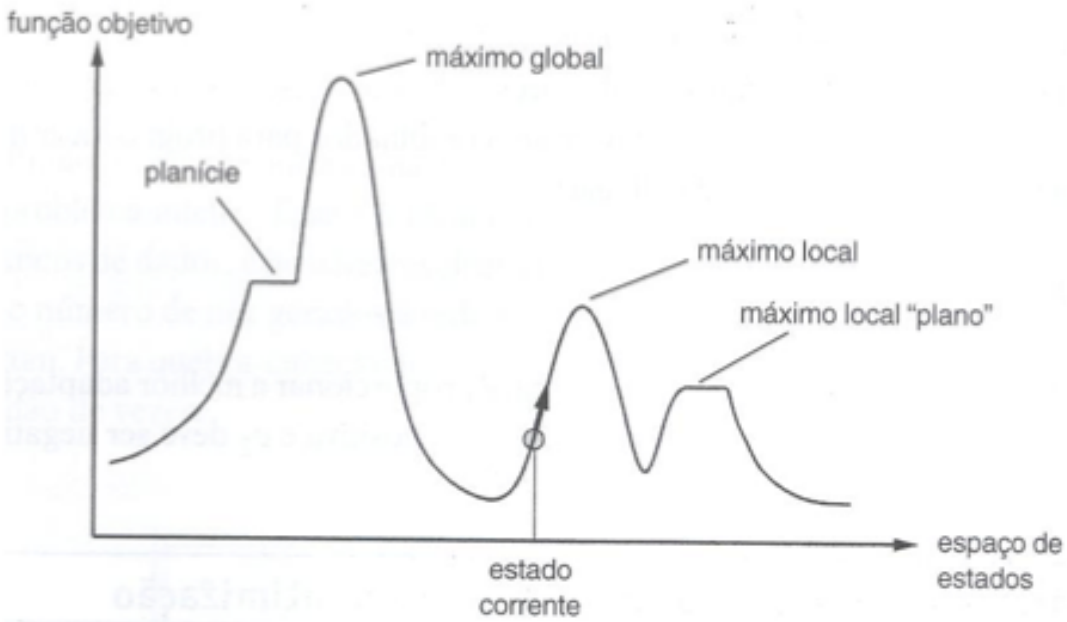
O custo (real) da solução é 26.

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.

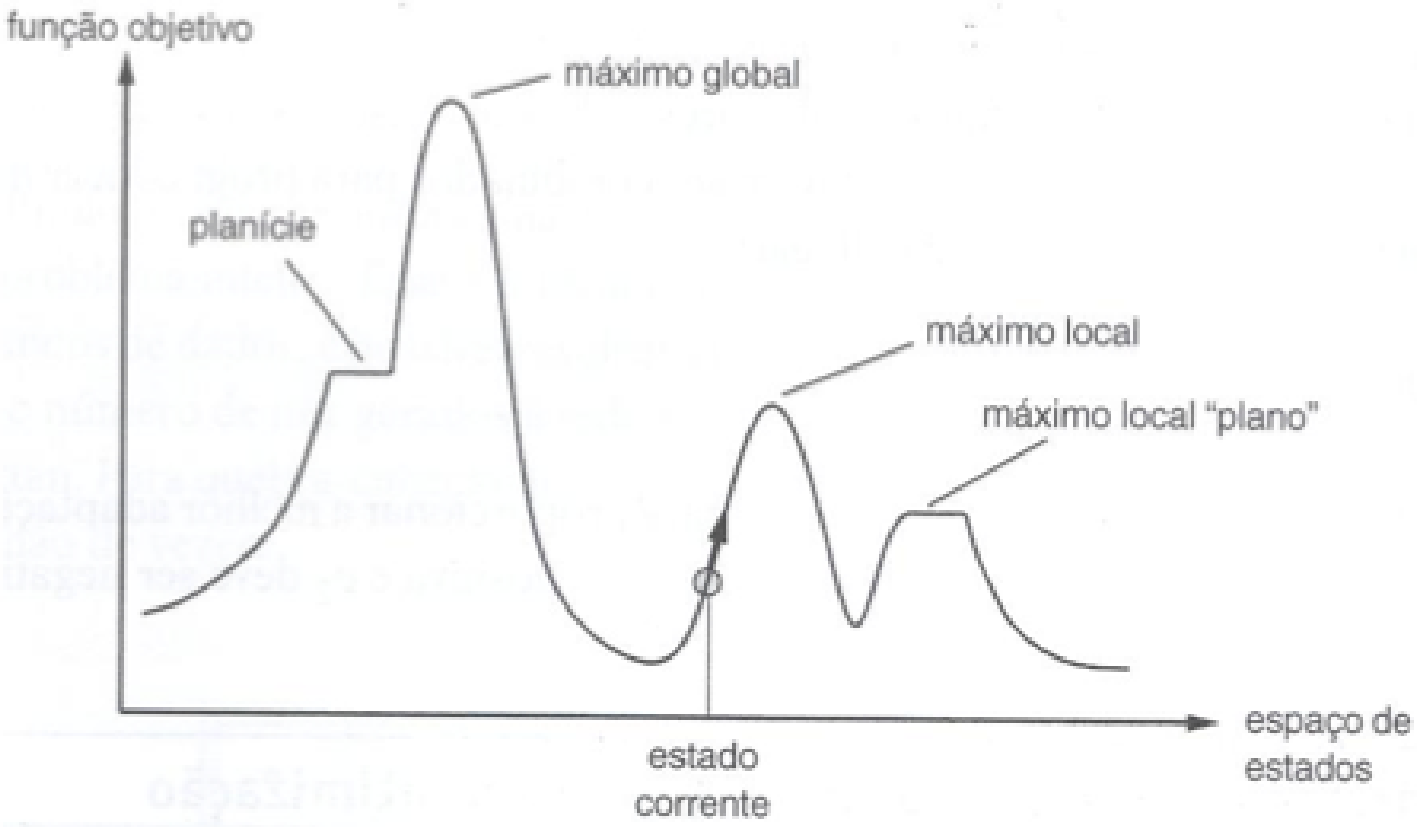
Busca Local - problema de otimização

Os algoritmos que vimos até agora são projetados para explorar sistematicamente um espaço de busca. Geralmente eles guardam o caminho realizado até chegar ao nó objetivo e então retornam o caminho como resposta. Se o caminho até o objetivo não importa, podemos considerar uma classe diferente de algoritmos, aqueles que não se preocupam de forma alguma com os caminhos.

A **busca local** opera usando um único estado corrente e em geral se move apenas para os vizinhos desse estado. Normalmente, os caminhos seguidos pela busca não são guardados.



Busca Local - problema de otimização



- 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.

Subida de Encosta

Um laço repetitivo que se move de forma contínua no sentido do valor crescente, isto é, encosta acima (para problemas de maximização - inverta o racicínio para problemas de minimização).

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

entradas: problema //um problema

variáveis locais: *corrente*, vizinho // nós

corrente ← CRIAR-NÓ (ESTADO-INICIAL[problema]);
repita
vizinho ← um sucessor de *corrente*;
se VALOR[vizinho] <= VALOR[corrente]
então retornar ESTADO[corrente]; // possibilidade de término
senão corrente ← vizinho;

Subida de Encosta

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

entradas: problema //um problema

variáveis locais: *corrente*, vizinho // nós

corrente ← CRIAR-NÓ (ESTADO-INICIAL[problema]);

repita

vizinho ← um sucessor de *corrente*;

se VALOR[vizinho] <= VALOR[corrente]

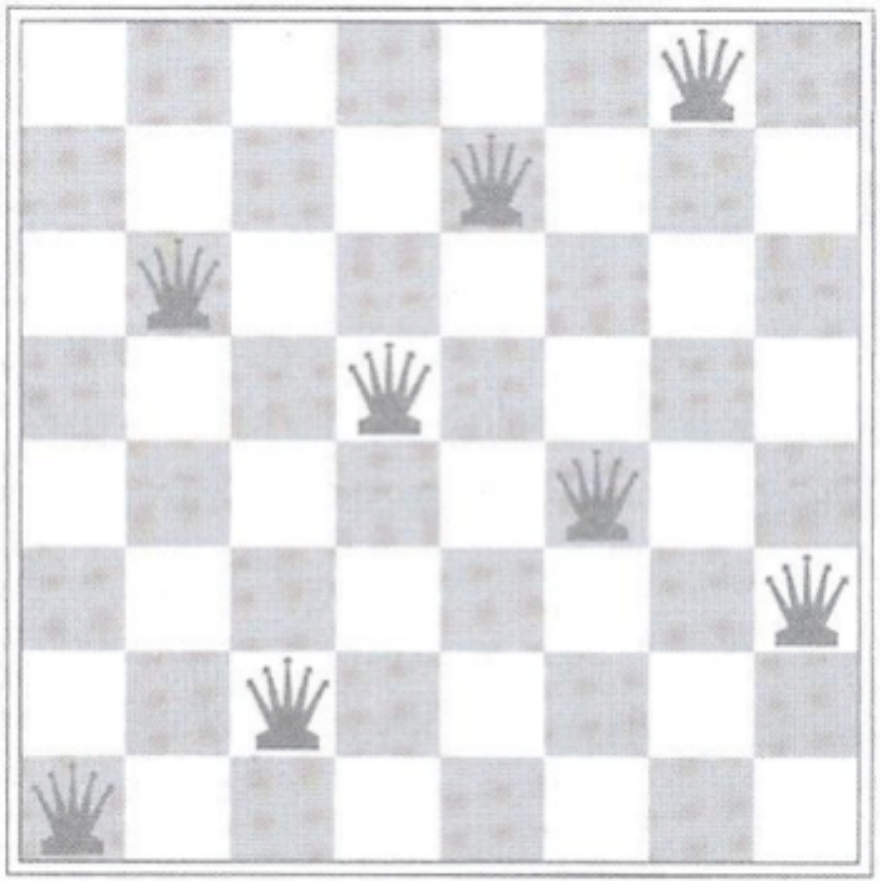
então retornar ESTADO[corrente]; // possibilidade de término

senão corrente ← vizinho;

Conceitos

- Vizinhança
- Primeira melhora
- Melhor melhora

Subida de Encosta



Subida de Encosta

Razões para 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 (slide anterior - segunda figura) é um mínimo local, pois todo movimento de uma única rainha piora a situação.
- Picos: resultam em uma sequência de máximos locais que torna muito difícil a navegação.
- Platôs: área da topologia do espaço de estados em que a função de avaliação é plana. Pode ser um máximo local plano ou uma planície.

Simulated Annealing - Recozimento (Têmpera) Simulado

Um algoritmo de subida de encosta nunca faz movimentos "encosta abaixo" em direção a estados com valor mais baixo (ou de custo mais alto). O algoritmo de Têmpera Simulada combina a subida de encosta com um percurso aleatório, e permite pioras no resultado, resultando em eficiência.

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 (se o movimento não melhora a situação), ele será aceito com algum **probabilidade** menor que I;
- a **probabilidade** I diminui exponencialmente de acordo com a má qualidade do movimento;
- a **probabilidade** I também diminui à medida que a **temperatura** T se reduz (movimentos ruins têm maior probabilidade de serem aceitos no início);

Simulated Annealing - Recozimento (Têmpera) Simulado

Recozimento é conhecido como um processo térmico para obter estados de baixa energia de um sólido. O aumento da temperatura faz com que o sólido derreta – na fase líquida, o arranjo das partículas é aleatório. Ao diminuir a temperatura cuidadosamente, as partículas se arrajam no estado de mínima energia do sólido.

Analogia com o sistema físico

- soluções em um problema de otimização são equivalentes a estados de um sistema físico;
- o custo de uma solução (valor da função objetivo) é equivalente à energia de um estado;
- a temperatura é equivalente a um parâmetro de controle;

Simulated Annealing - Recozimento (Têmpera) Simulado

Algoritmo apresentado no livro texto (simplificado)

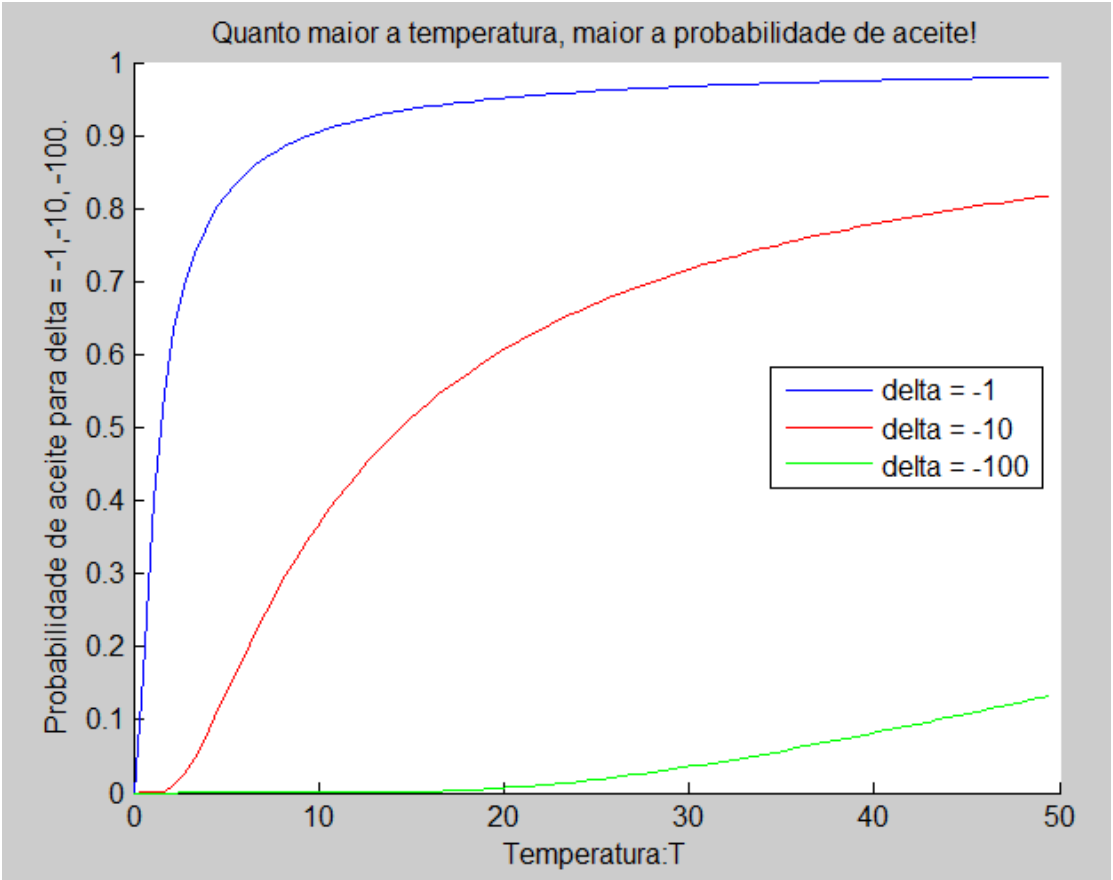
função TEMPERA-SIMULADA (problema, escalonamento) **retorna** um estado solução
  entradas: problema // um problema
  escalonamento // um mapeamento de tempo para "temperatura"
  variáveis locais: corrente, próximo // dois nós
  T // uma "temperatura" que controla a probabilidade de passos descendentes

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 $e^{\Delta E/T}$;

Pode ser considerada uma meta-heurística!!!!
Meta-heurística ajuda a sair de mínimos e máximos locais.

Simulated Annealing - Recozimento (Têmpera) Simulado

Estudo da probabilidade de aceitação de um movimento de piora $e^{\Delta E}/T$



Simulated Annealing - Recozimento (Têmpera) Simulado

Como determinar o parâmetro de temperatura????

- calculando o custo de *várias soluções* na vizinhança de uma *solução aleatória* e tomar como temperatura inicial o maior custo;
- gerar um *conjunto de soluções* vizinhas de uma *solução aleatória*; iniciar com uma temperatura baixa e contar quantas soluções vizinhas são aceitas; aumentar a temperatura de forma a aceitar uma quantidade *grande* de vizinhos.

Simulated Annealing - Recozimento (Têmpera) Simulado

Vídeo: Traveling Salesman Problem, four algoritms
`https://www.youtube.com/watch?v=q6fPk0--eHY`

- 200 cidades, 4 algoritmos de busca
  1. Random Path - comprimento da rota $= 327.452,4$
  2. Greed Search (Busca Gulosa) - comprimento da rota $= 36.226,2$
  3. 2-Opt - comprimento da rota $= 31.887,0$
  4. 2-Opt + Simulated Annealing - comprimento da rota $= 30.944,3$

Heurísticas de construção e Heurísticas de melhoria

A busca gulosa pode ser vista, neste contexto, como uma **heurística de construção**. Ela consegue, passo por passo, usando a heurística da cidade mais próxima, conseguir uma solução (não tão boa) para o problema. 2-Opt é uma **heurística de melhoria**: escolhe duas arestas para eliminar e substitui por arestas cruzadas, considerando os pontos das arestas originais. Se melhorar, mantenha. Também pode ser vista como geração do sucessor (vizinhança) na sistemática de busca local.

Simulações

- quadrado de 8 peças com Busca Gulosa: `https://www.youtube.com/watch?v=e6njY7rVe3E`
- shorts path com A^*: `https://www.youtube.com/watch?v=-L-WgKMFuhE`
- N-rainhas com SA: `https://www.youtube.com/watch?v=FsJ8npn8z2w`

Algoritmo Genético

Processo evolutivo natural

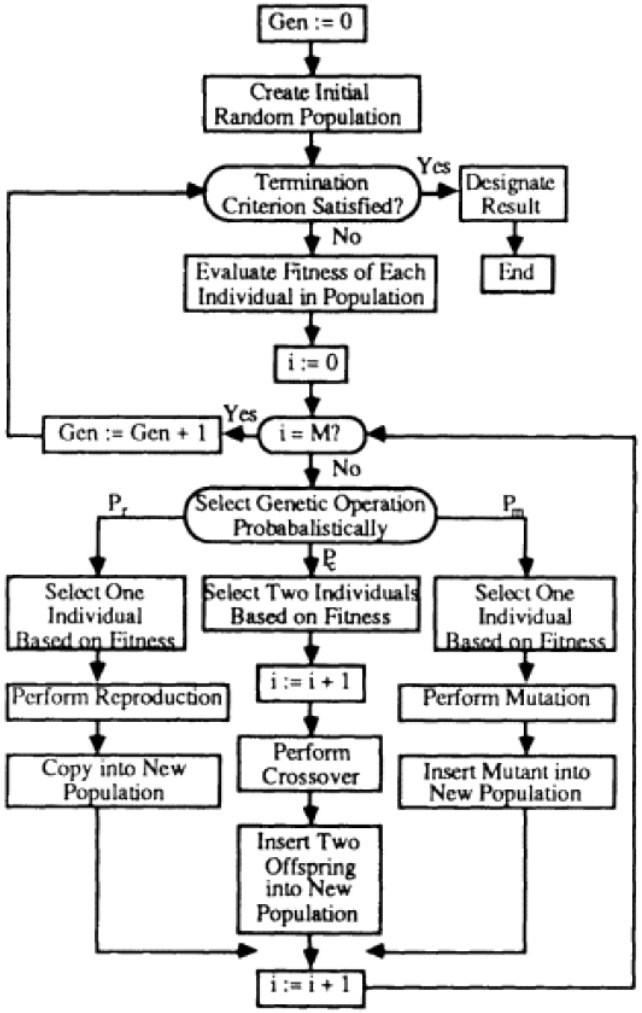
- Uma entidade tem a habilidade de se reproduzir
- Existe uma população formada por essas entidades
- Existe alguma diversidade entre essas entidades
- Algumas diferenças na habilidade de sobreviver nesse ambiente está associada com essa variedade

John Holland, em seu livro (pioneiro) *Adaptation in Natural and Artificial Systems* (1975), apresentou uma estrutura geral para representar todos os sistemas adaptativos (naturais ou artificiais) e então mostrou como um processo evolutivo pode ser aplicado a sistemas artificiais. Qualquer problema com características adaptativas pode, geralmente, ser formulado em termos genéticos. Uma ver formulado nesses termos, o problema pode ser, frequentemente, resolvido por meio de um **algoritmo genético**.

Algoritmo Genético

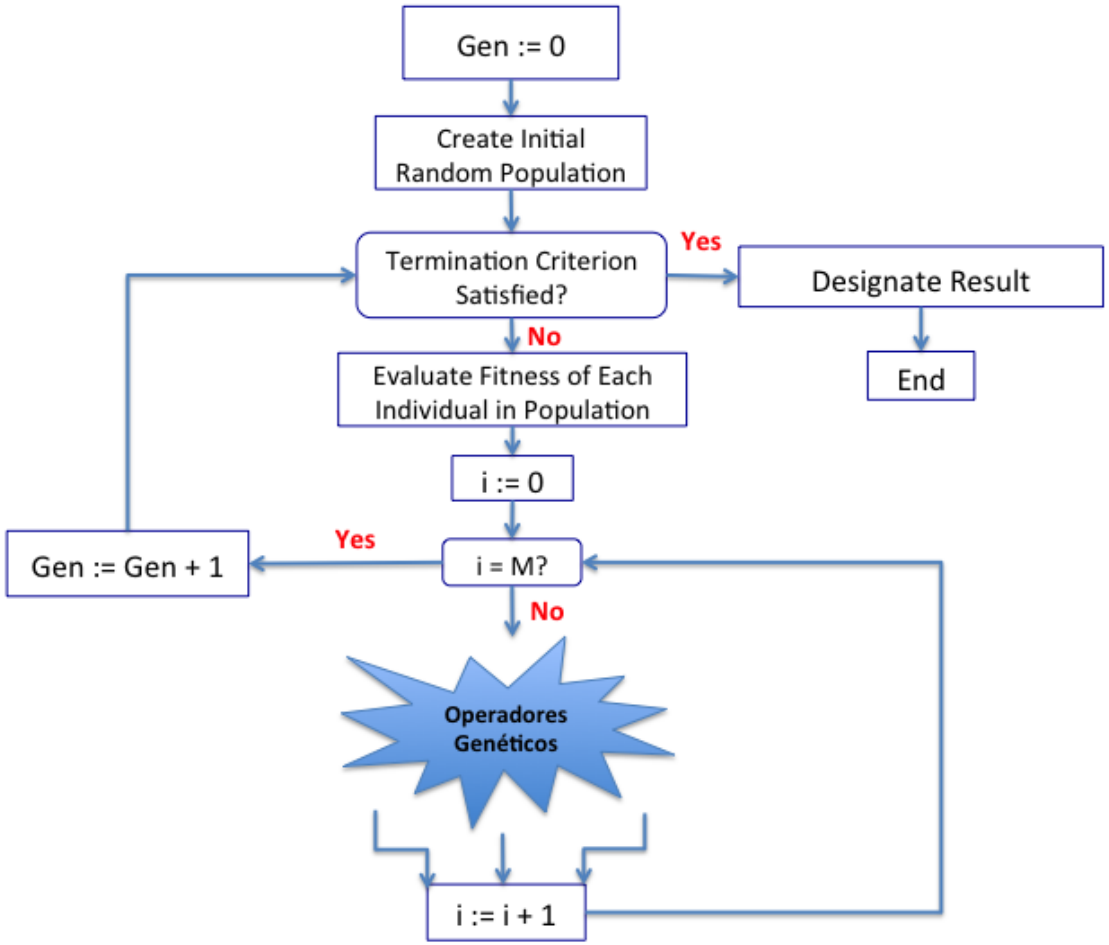
Um **algoritmo genético** é um algoritmo matemático (altamente) paralelo que transforma um conjunto (população) de objetos matemáticos individuais (tipicamente uma string de caracteres de tamanho fixo - o **cromossomo**), cada um associado a um valor de **fitness**, em uma nova população (a próxima geração) usando operações baseadas no princípio Darwiniano de reprodução (operações genéticas) e sobrevivência do mais bem adaptado.

Algoritmo Genético - básico



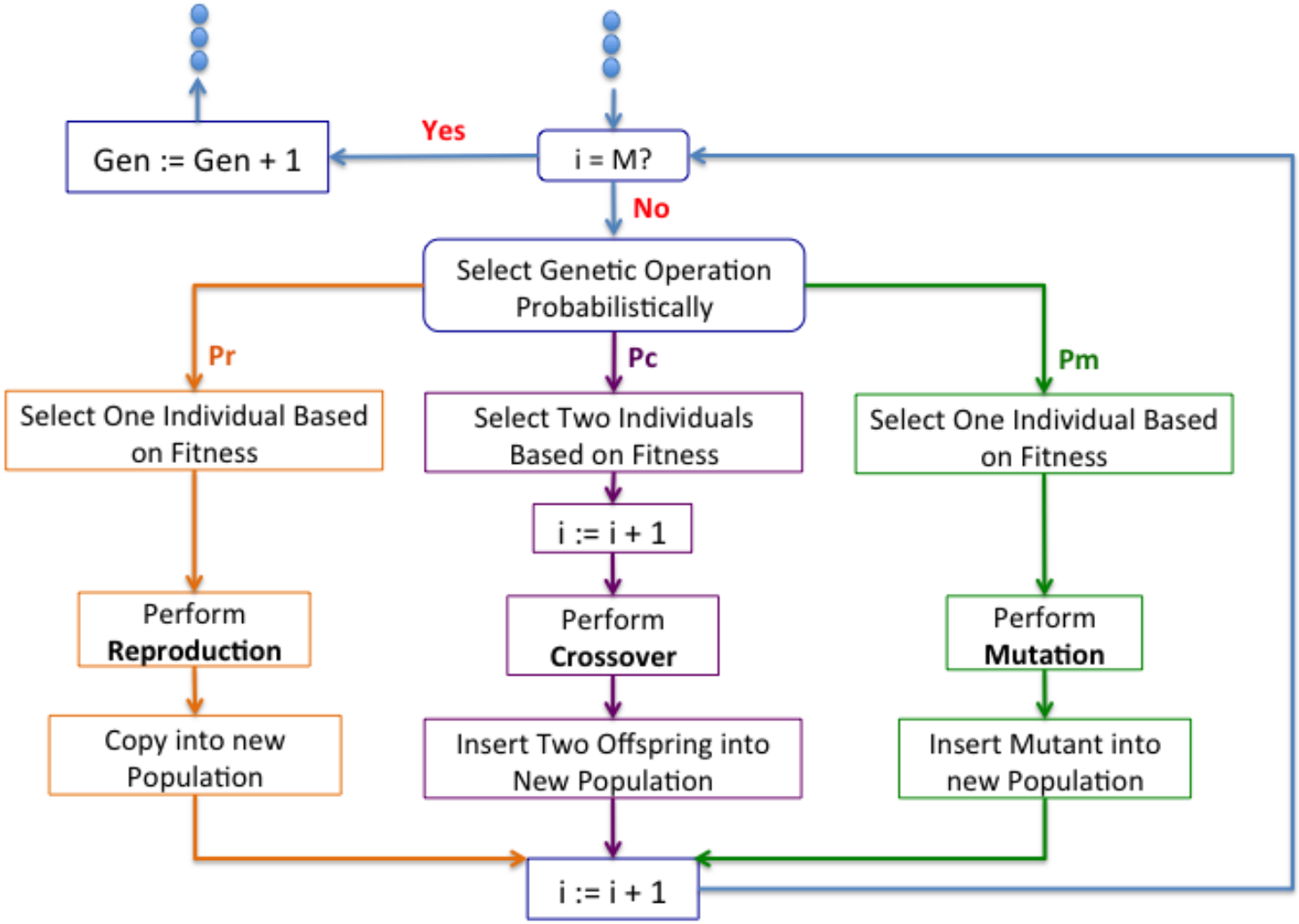
Koza J. R.. Genetic Programming: On the Programming of Computers by Means of Natural Selection. 6th print. MIT Press, England, 1998.

Algoritmo Genético - gerações



i: contador para quantidade de indivíduos gerados para a próxima geração.

Algoritmo Genético - operadores genéticos básicos



O problema do Restaurante Hambuguer

Definição do problema

Trata-se do problema de encontrar a melhor estratégia de negócios para uma cadeia de quatro restaurantes, considerando três decisões binárias:

- Preço: o preço do hambúrguer dever ser 50 centavos ou 10 dólares?
- Bebida: deve ser servido vinho ou refrigerante com o hambúrguer?
- Serviço: o restaurante deveria fornecer um serviço lento e calmo, com garçons vestidos com ternos, ou deveria fornecer um serviço rápido e divertido, com garçons vestidos em uniformes de poliester brancos?

Meta

A meta é encontrar uma combinação dessas três decisões (i.e. uma estratégia de negócios) que maximize o lucro.

O problema do Restaurante Hambuguer

Representação - cromossomo

Uma string de comprimento $L = 3$ com um alfabeto de tamanho $K = 2$. Para cada decisão, o valor 0 ou 1 é associado a uma posição no cromossomo. O espaço de busca para este problema consiste em $2^3 = 8$ possíveis **estratégias de negócios**.

Considere:

- Preço alto = 0; Preço baixo = 1;
- Bebida vinho = 0; Bebida refrigerante = 1;
- Servico lento = 0; Serviço rápido = 1;

Tabela : Representação de quatro restaurantes

Id. Restaurante	Preço	Bebida	Serviço	Cromossomo
1	alto	cola	rápido	011
2	alto	vinho	rápido	001
3	baixo	cola	lento	110
4	alto	cola	lento	010

O problema do Restaurante Hambuguer

Fitness

A avaliação do indivíduos (cromossomos - restaurante) será o valor decimal equivalente à string binária.

Tabela : Fitness da população na geração 0

id.	String (X_i)	Fitness $f(X_i)$
1	011	3
2	001	1
3	**110**	**6**
4	010	2
	Total	12
	Pior	1
	Médio	3,00
	Melhor	6

Operadores

Quatro operadores básicos:

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

O problema do Restaurante Hambuguer

Considere a população da geração 0. O fitness do indivíduo 3 representa $\frac{1}{2}$ do fitness total da população. Ele possui 50% de chance de ser selecionado para sofrer a atuação dos demais operadores. Considere, como exemplo, uma possível situação de aplicação dos operadores de seleção e reprodução.

Tabela : Fitness da população na geração 0 e 1

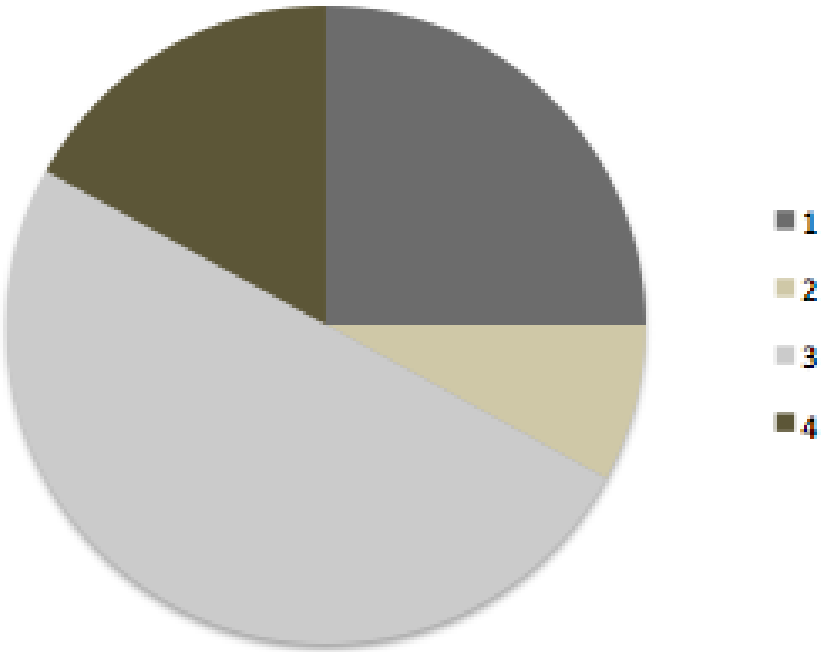
id.	String (X_i)	Fitness $f(X_i)$	$\frac{f(X_i)}{\sum f(X_i)}$	Sel. e Reprod.	$f(X_i)$
1	011	3	0,25	011	3
2	001	1	0,08	**110**	**6**
3	**110**	**6**	0,50	**110**	**6**
4	010	2	0,17	010	2
	Total	12			17
	Pior	1			2
	Médio	3,00			4,25
	Melhor	6			6

Note que o fitness médio da população melhorou, o fitness total e o fitness do pior indivíduo também, mas o fitness do melhor indivíduo continua o mesmo.

O operador de seleção - roleta

Passos:

- calcule o fitness total da população;
- calcule a distribuição dos fitness dos indivíduos considerando o fitness total da população;
- construa a roleta atribuindo uma parte dela para cada indivíduo considerando a sua probabilidade (fitness) de sobrevivência na população;
- sorteie um número entre 1 e 100;
- posicione o número sorteado na roleta e selecione o indivíduo que ocupa essa parte da roleta.



O operador crossover de um ponto

Permite que novos indivíduos sejam criados. Com a aplicação do crossover, novos pontos no espaço de busca são testados. Na sua forma básica, produz dois novos indivíduos (seus descendentes). Os descendentes são (preferencialmente) diferentes dos pais e diferentes entre si e possuem material genético de ambos os pais.

Passos:

- dado dois indivíduos selecionados a partir do operador de seleção;
- selecione aleatoriamente um número entre 1 e $L - 1$ usando uma distribuição de probabilidade uniforme (ponto de crossover);
- separe cada um dos cromossomos pais em duas partes, de acordo com o ponto de crossover;
- combine as partes criando os dois novos cromossomos.

Executando ...

- PAI 1: 011 PAI 2: 110
- ponto de crossover: $L = 2$
- fragmentos do PAI 1: 01- e - -1
- fragmentos do PAI 2: 11- e - -0
- FILHO 1: 010 FILHO 2: 111

O problema do Restaurante Hambuguer

Uma possibilidade e população considerando a aplicação dos operadores de reprodução e de crossover é:

Tabela : Fitness da população na geração 0, 1 e 2

id.	String (X_i)	Fitness $f(X_i)$	$\frac{f(X_i)}{\sum f(X_i)}$	Sel. e Reprod.	$f(X_i)$	Pto. Cross.	X_i	$f(X_i)$
1	011	3	0.25	011	3	2	111	7
2	001	1	0.08	**110**	**6**	2	010	2
3	**110**	**6**	.050	**110**	**6**	–	110	6
4	010	2	0.17	010	2	–	010	2
	Total	12			17			17
	Pior	1			2			2
	Médio	3.00			4.25			4.25
	Melhor	6			6			7

Obs.1: Considere que há uma probabilidade de crossover. Então apenas uma crossover ocorreu. Arbitrariamente os indíviduos 3 e 4, que não reproduziram, foram repetidos no conjunto de indivíduos.
Obs.2: O indivíduo ótimo apareceu na população. Porém não necessariamente sabemos disso. Esse indivíduo deve dominar o operador de seleção e a população vai convergir para ele (os demais cromossomos vão, com o tempo, assumir a mesma codificação que este indivíduo).

O problema do Restaurante Hambuguer

Assumindo que atingimos um critério de parada, a estratégia de negócios de resposta do algoritmo deve ser aquela que advém do indivíduo de melhor fitness:

- vender o hamburguer a 50 centavos;
- servir bebida a base de cola;
- oferecer um serviço rápido.

Critérios de parada

Número máximo de gerações, convergência da população, alcance da solução ótima se esse valor de fitness for conhecido.

O operador de mutação básico

Esse operador deve ser usado com moderação. Opera sobre um único indivíduo. A função do operador de mutação é inserir diversidade genética na população.

- Aleatoriamente selecione um indivíduo na população;
- Aleatoriamente escolha um número entre 1 e L (ponto de mutação);
- Aleatoriamente escolha um caracter do alfabeto e o insira no ponto de mutação;

Algoritmo Genético

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

Esquemas

Esquema

Um esquema consiste em um *template* que descreve um suboconjunto dentre o conjunto de todos os indivíduos possíveis – descreve quais posições de sua codificação genética são idênticas. Um símbolo "coring"(*, por exemplo) é usado para representar as posições nas quais os cromossomos diferem.

Tabela : Exemplo de esquemas

Esquema	Indivíduos
1*	10 11
1*0*1	10001 10011 11001 11011
**0	000 010 100 110

Teorema dos esquemas: by Holland

Um Algoritmo Genético é um manipulador de esquemas. Os esquemas contêm as características positivas e negativas que podem levar a uma boa ou má avaliação. O Algoritmo Genético apresenta uma tendência a propagar bons esquemas por todas a população durante sua execução.

Variações nos operadores

- **Crossover de dois pontos**: Melhora a capacidade do Algoritmo Genético de preservar *esquemas*. Por exemplo, o esquema 1 ****** 1 não pode ser mantido quando um crossover de um ponto é usado. No crossover de dois pontos seleciona-se dois pontos de corte. O primeiro filho será formado pela parte do primeiro pai fora dos pontos de corte e pela parte do segundo pai entre os pontos de corte. O segundo filho será formado pelas partes restantes. O crossover de dois pontos, por outro lado, tem mais probabilidade de quebrar esquemas longos.
- **Crossover uniforme**: É mais adequado para o objetivo de combinar esquemas, porém corre o risco de destruir totalmente o esquema. Para cada gene é sorteado um número no conjunto $\{0, 1\}$. Se o valor sorteado for igual a um, o filho número um recebe o gene da posição corrente do primeiro pai e o segundo filho recebe o gene corrente do segundo pai. Se o valor sorteado for zero, as atribuições serão invertidas.

Variações nos operadores

- **Crossover baseado em maioria**: Acelera a convergência genética. Nesse operador, sorteia-se *n* pais e faz-se com que cada *bit* do filho seja igual ao valor da maioria dos pais selecionados. Alternativamente pode-se associar probabilidade para cada valor, em vez de decidir diretamente pelo voto.
- **Probabilidades variáveis**: Pode ser interessante estabelecer uma função de decaimento da probabilidade do crossover, enquanto a mesma função rege o aumento da probabilidade de mutação. Essa mudança das probabilidades deve ser feita em função do tempo (gerações) e deve ser usada com cuidado.

Variações nos operadores

- **Mutação dirigida**: Imagine que o problema de convergência precoce (ou insistente) de uma população seja responsabilidade de um *esquema dominante* presente nos melhores indivíduos. Nesse caso, é necessário dirigir a mutação para esse esquema. Esse operador deve ser ativado depois que um grande número de gerações foi gerado. Quando atividado, ele busca as n melhores soluções e verifica qual é a bagagem cromossômica que elas têm em comum. Três problemas a tratar com esse operador: (a) quando começar a usar; (b) como dar sobrevida aos cromossomos gerados; (c) quando parar de usar.

Variações nos operadores

- **Seleção por torneio**: escolhe-se aleatoriamente k indivíduos (que participarão do torneio). O torneio consiste em deixar que o mais bem adaptado no grupo seja selecionado. Se k igual ao tamanho da população, então o mehor indivíduo sempre será selecionado. $k = 1$ é a única chance do pior indivíduo ser selecionado.
- **Seleção por ranking**: ordena os indivíduos de acordo com o seu valor de fitness e então aplica um procedimento que minimiza as diferenças entre suas avaliações. Então, o método roleta pode ser aplicado. Note que com a minimização das diferenças, os indivíduos terão chances mais parecidas de serem selecionados.
- **Seleção truncada**: apenas os $x\%$ melhores indivíduos participarão da seleção.

Adaptação de parâmetros

- **determinística**: aplica-se mudanças nos parâmetros, no decorrer da execução do Algoritmo Genético, seguindo alguma regra determinística. A regra modifica os parâmetros sem nenhum tipo de *feedback* do algoritmo. Ex: aumentar a probabilidade de mutação em $0,01\%$ a cada geração criada.
- **adaptativa**: algum *feedback* do algoritmo é usado para determinar o valor do parâmetro para a próxima geração. Ex. Regra de $\frac{1}{5}$: se mais de $\frac{1}{5}$ das mutações forem bem sucedidas aumenta-se a taxa de mutação, caso contrário, diminui-se a taxa de mutação.
- **auto-adaptativa**: codificar os parâmetros do Algoritmo Genético dentro do cromossomo e deixá-los evoluir junto com a solução. *% sugestão do Linden ... sem mais esclarecimentos na bibliografia*

Exercício - Modelagem

Para cada um dos contextos:

- apresente uma ou mais representações para o cromossomo;
- apresente uma ou mais funções fitness;
- para cada representação e função fitness criada, mostre uma solução decodificada (fenótipo) e o seu valor de fitness;
- apresente um estudo sobre a geração de cromossomos infactíveis a partir da aplicação dos operadores evolutivos (crossover e mutação);
- no caso dos operadores evolutivos gerarem soluções infactíveis, discuta maneiras de tratar o problema ou com operadores alternativos ou com penalização na função fitnes.

Exercício - Modelagem

- **Problema das N rainhas**: 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.
- **Escalonamento de tarefas**: considere um conjunto de N tarefas de tempos de processamento diferentes a serem processados em M máquinas. As máquinas são paralelas de igual poder de processamento. Não há ordem para a execução das tarefas. Determine a melhor distribuição de tarefas nas máquinas de forma a minimizar o tempo de execução do conjunto completo de tarefas.

Exercício - Modelagem

- **Caixeiro Viajante**: um caixeiro precisa percorrer N cidades, passando uma vez em cada uma delas. Nenhuma delas pode ficar fora da rota e a rota deve ser mínima (mais barata). Considere que você tem um grafo (cidades são os nós e as arestas são os caminhos entre as cidades) totalmente conectado e bidirecional, e que o custo do caminho entre uma cidade e outra é dada como um peso na aresta do grafo.
- **Problema da mochila**: dado um conjunto de N objetos e uma mochila com: C_j representando o benefício do objeto j; s_j representando o peso do objeto j e b representando a capacidade da mochila. Determine quais objetos devem ser colocados na mochila para maximizar o benefício total de forma que o peso da mochila não ultrapasse sua capacidade.

Exercício - Modelagem

- **O problema do agrupamento**: dado um conjunto de N objetos, encontre uma distribuição desses objetos em k grupos, de tal forma que os objetos associados a um mesmo grupo sejam mais similares entre si do que são em relação ao objetos associados a grupos diferentes.
  - os objetos são descritos por atributos que assumem valores numéricos; assim podem ser visto como um vetor onde cada coordenada é o valor que o objeto assume para um determinado atributo; assim $\overrightarrow{ob} = \{at_1, at_2, ..., at_d\}$
  - para saber a similaridade entre os objetos, assuma uma medida de distância entre vetores; quanto **menor** a distância **maior** a similaridade;

Discutindo o problema da mochila

Dados:

- um conjunto finito de n objetos (x);
- um fator de importância associado a cada objeto (v);
- o peso de cada objeto (w);
- uma mochila de capacidade finita (W).

Problema da mochila - *Knapsack problem*

O problema da mochila consiste em determinar quais objetivos devem ser colocados na mochila, maximizando o benefício total da carga sem que o peso da carga ultrapasse a capacidade da mochila.

Problema de maximimização de

$$solution = \sum_{i=1}^{n} v_i x_i$$

sujeito a

$$\sum_{i=1}^{n} w_i x_i \leq W \text{ e } x_i \in \{0, 1\}.$$

Discutindo o problema da mochila

- cromossomo: um array de n posições, sendo que cada posição representa um objeto do conjutno de objetos disponíveis;
- alfabeto: binário $\{0, 1\}$, sendo que 0 indica que o objeto não está na mochila e 1 indica que o objeto está na mochila;
- função fitness (opção 1): soma da importância dos objetos colocados na mochila, ou seja, $\sum_{i=1}^{n} v_i x_i$;
- função fitness (opção 2): função fitness (opção 1) acrescentada de um fator de penalização, ou seja, $\sum_{i=1}^{n} v_i x_i - \alpha * max(0, \sum_{i=1}^{n} w_i x_i - W)$, sendo que $\alpha = W$, por exemplo, é um exemplo de coeficiente de penalidade.
- operador de correção: retire um (ou mais) objeto da mochila aleatoriamente, até que a capacidade da mochila esteja respeitada.
- operador de correção com estratégia gulosa: retire primeiro o(s) objeto(s) com maior razão importância/peso.

Profa. Dra. Sarajane Marques Peres
Universidade de São Paulo
Escola de Artes, Ciências e Humanidades
Sala 320-A - Bloco I1
sarajane@usp.br www.each.usp.br/sarajane