



Universidade do Minho

Escola de Engenharia

Departamento de Informática

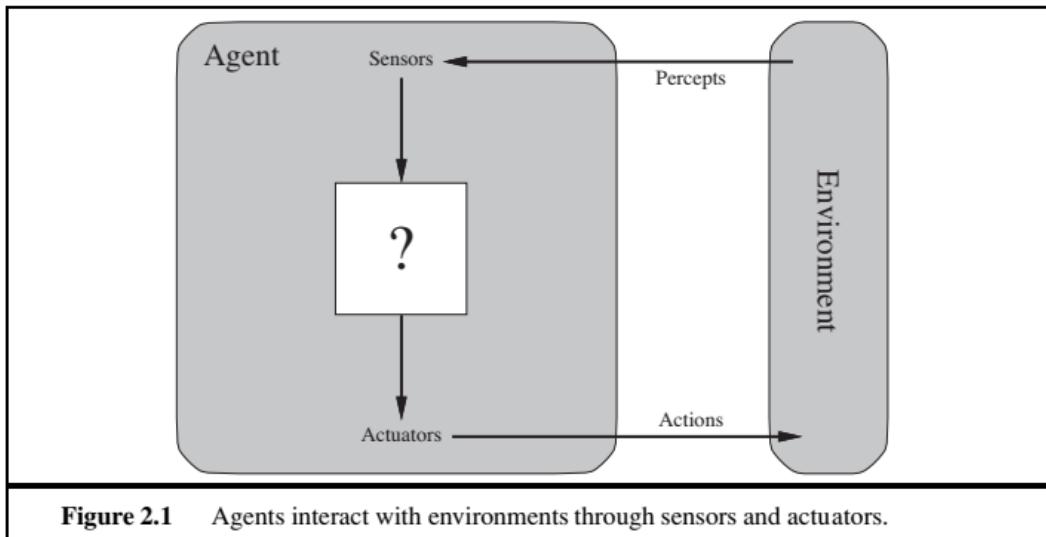
MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial

- Formulação de Problemas
- Resolução de problemas
- Tipos de problemas
- Exemplos de problemas do mundo real
- Pesquisa de soluções
- Estratégias de Pesquisa
 - Pesquisa Não-Informada (cega)
 - Pesquisa Informada (heurística)
- Para além da pesquisa Clássica



- **Agente:** é algo que perceciona o ambiente através de sensores e atua no ambiente através de atuadores



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- O termo **perceção** refere-se às percepções do agente num dado instante. Uma **sequência de percepções** do agente é o histórico total de tudo o que agente percecionou.
- O comportamento do agente é descrito matematicamente pela **função do agente** que mapeia qualquer sequência de percepções numa ação.
- **Agente : arquitetura* + programa****

* dispositivo computacional com sensores e atuadores

** materialização da função do agente

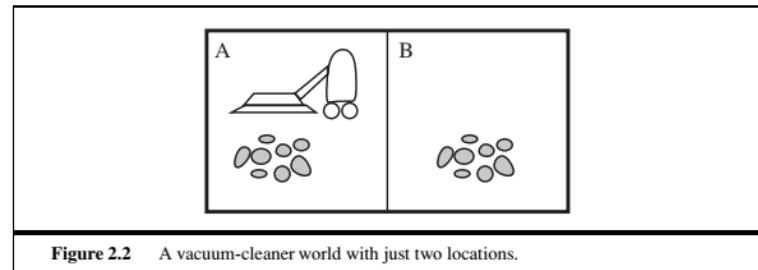


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
:	:

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Como é que um agente pode agir, estabelecendo objetivos e considerando possíveis sequências de ações para atingir esses objetivos?
- A concretização de um simples agente que resolve problemas necessita primeiramente de formular um **objetivo** e um **problema**, procurar a sequência de ações que resolvam o problema, executando-as uma de cada vez. Quando completo, formulará um outro objetivo e executará de novo.
- Portanto, considera-se que neste contexto a resolução de problemas apresenta-se como a formulação de um problema como um **problema de pesquisa**.

Formulação de problemas

- “Problem Solving Agent”: Procura encontrar a sequência de ações que leva a um estado desejável!
- Formulação do Problema:
 - Quais as ações possíveis? (qual o seu efeito sobre o estado do mundo?)
 - Quais os estados possíveis? (como representá-los?)
 - Como avaliar os Estados
- Problema de pesquisa:
 - Solução: sequência de ações
- Fase final é a execução!
- Formular → Pesquisar → Executar

Formulação de problemas

- Muitos dos problemas em ciências da computação podem ser formulados como:
 - Um conjunto S de ESTADOS (possivelmente infinito)
 - Um estado INICIAL $s \in S$
 - Uma relação de TRANSIÇÃO T ao longo deste espaço de estados
 - Um conjunto de estados FINAIS (objetivos): $O \subseteq S$
- Um problema pode ser definido formalmente em cinco componentes:
 1. Representação do Estado
 2. Estado Inicial (Atual)
 3. Estado Objetivo (define os estados desejados)
 4. Operadores (Nome, Pré-Condições, Efeitos, Custo)
 5. Custo da Solução

Resolução de problemas

- O problema pode ser resolvido através de pesquisa e um caminho entre o estado inicial e um estado objetivo
- Em suma, a formulação do problema envolve decidir que ações e estados a considerar tendo em conta um objetivo

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

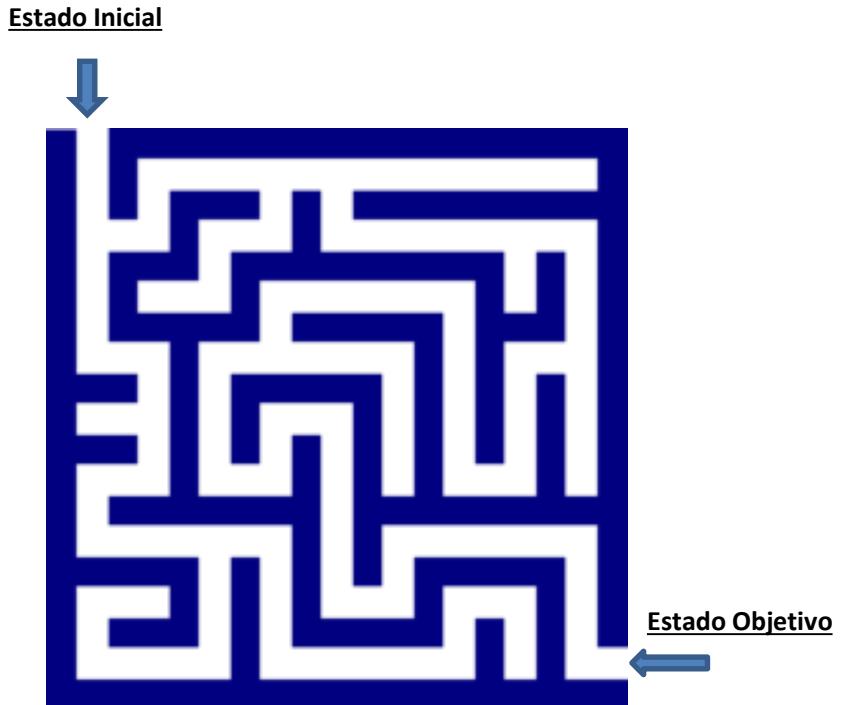
  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then do
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
  return action
```

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Formulação de problemas

Componentes de um problema de procura

- Estado inicial
- Ações
- Modelo de transição
 - Que estado resulta da execução de uma determinada ação em um determinado estado?
- Estado Objetivo
- Custo do caminho
 - Suponha que seja uma soma dos custos não negativos de cada etapa
- A solução ideal é a sequência de ações que fornece o menor custo de caminho para alcançar a meta



Formulação de problemas

Exemplo: Viajar de Arad para Bucharest

Estado Inicial: Arad;

Estado Objetivo: Bucharest

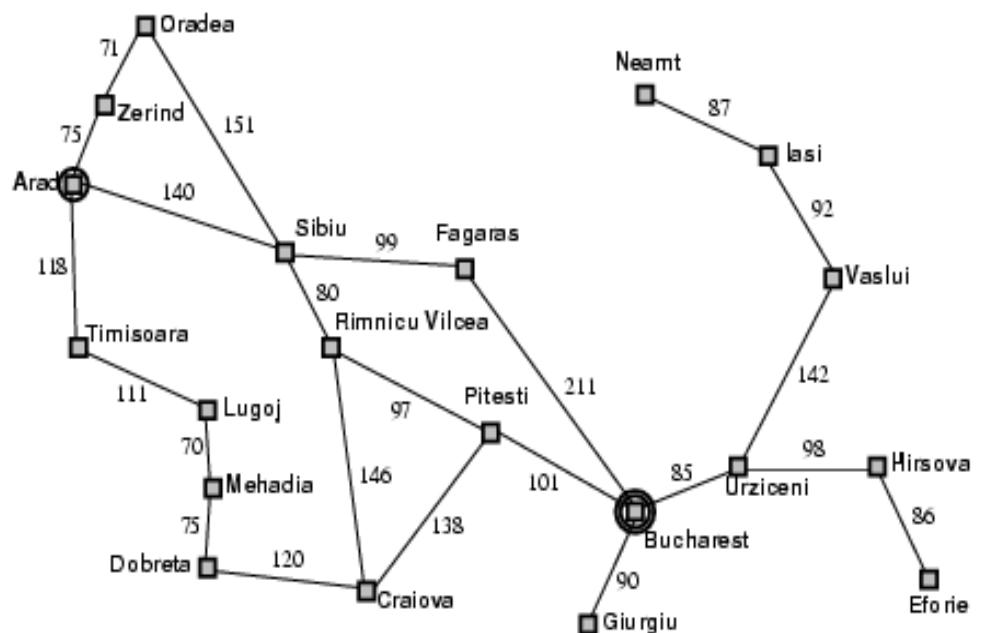
Estados: as várias cidades

Operações: conduzir entre as cidades

Solução:

- Um caminho*: sequência de cidades, e.g., Arad, Sibiu, Fagaras, Bucharest
 - O caminho mais curto
 - O caminho mais rápido

* O custo da solução é determinado pelo custo de cada caminho, que reflete uma medida de desempenho da solução (neste caso o custo será a distância entre cidades).



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Formulação de problema

- Uma solução para um problema é o caminho do estado inicial para o estado do objetivo;
- A qualidade da solução é medida pela função de custo do caminho e uma solução “ideal” tem o menor custo do caminho entre todas as possíveis soluções;
- O mundo “real” é obviamente mais complexo:
 - o espaço de estado é um abstração para a solução de problemas;
- Estado (abstrato) = conjunto de estados reais;
- Ação (abstrata) = combinação complexa de ações reais;
 - Eg., "Arad \Rightarrow Zerind" representa (uma abstração) um conjunto mais complexo de rotas possíveis, desvios, paragens, etc.
- Para garantia de realização, qualquer estado "em Arad" deve chegar a algum estado "em Zerind";
- Solução (abstrata) = conjunto de caminhos reais que são soluções no mundo real;
- Cada ação abstrata deve ser "mais fácil" do que o problema original.

Tipos de problemas

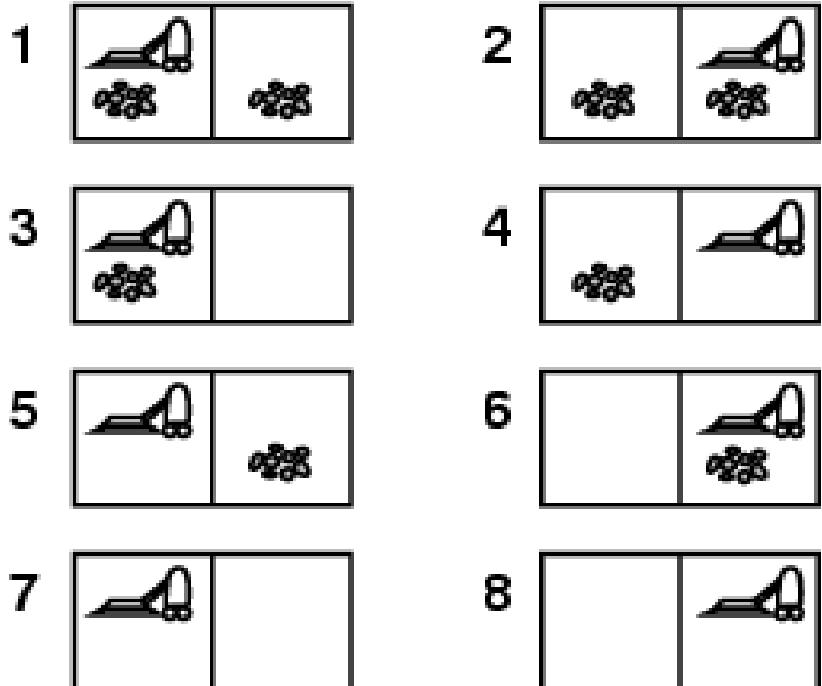
- Ambiente determinístico, totalmente observável → **problema do estado único**
 - O agente “sabe” exactamente o estado em que estará; a solução é uma sequência.
- Ambiente determinístico, não acessível → **problema de múltiplos estados**
 - O agente não “sabe” onde está; a solução é uma sequência
- Ambiente não determinístico e/ou parcialmente acessível → **problema de contigência**
 - Percepções fornecem novas informações sobre o estado actual
 - Frequentemente intercalam pesquisa e execução
- Espaço de estados desconhecido → **problema de exploração**

Tipos de problemas

Exemplo: o problema do aspirador

- 2 localizações (lixo e não lixo)
- 3 ações (left, right, suck)
- Objectivo: limpar o lixo
- Custo: uma unidade por ação

- Problema de:
 - Estado Único se...
 - Múltiplos Estados se...
 - Contingência se...
 - Exploração se...



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Tipos de problemas

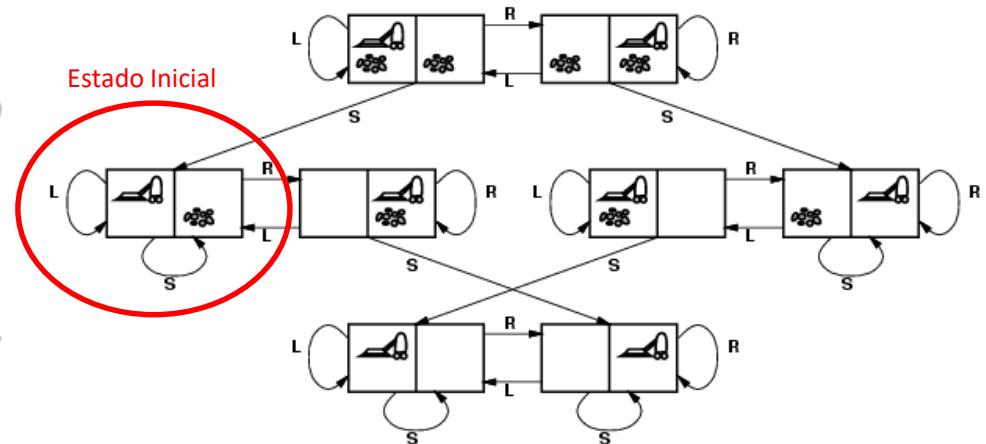
Exemplo: o problema do aspirador

(problema do estado único)

Se o ambiente é completamente observável, o aspirador saberá sempre onde está e onde está o lixo.

A solução é então reduzida à procura de um caminho do estado inicial até ao estado objectivo.

Solução? [Right, Suck]



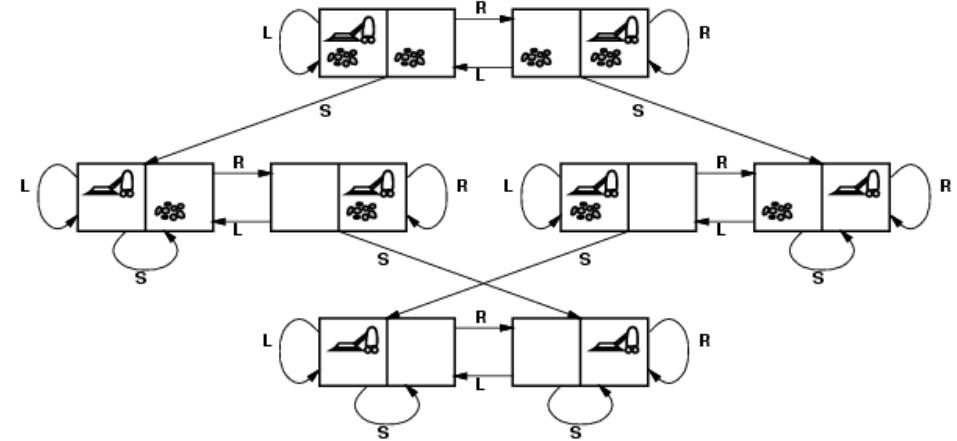
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador

(problema do estado único)

Formulando o problema:

- Estado: 8 estados representados (definidos pela posição do robô e lixo)
- Estado inicial: Qualquer um
- Operadores: esquerda, direita, aspirar
- Teste Objetivo: Não há lixo em nenhum dos quadrados
- Custo da Solução: Cada ação custa 1 (custo total = número de passos da solução)



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

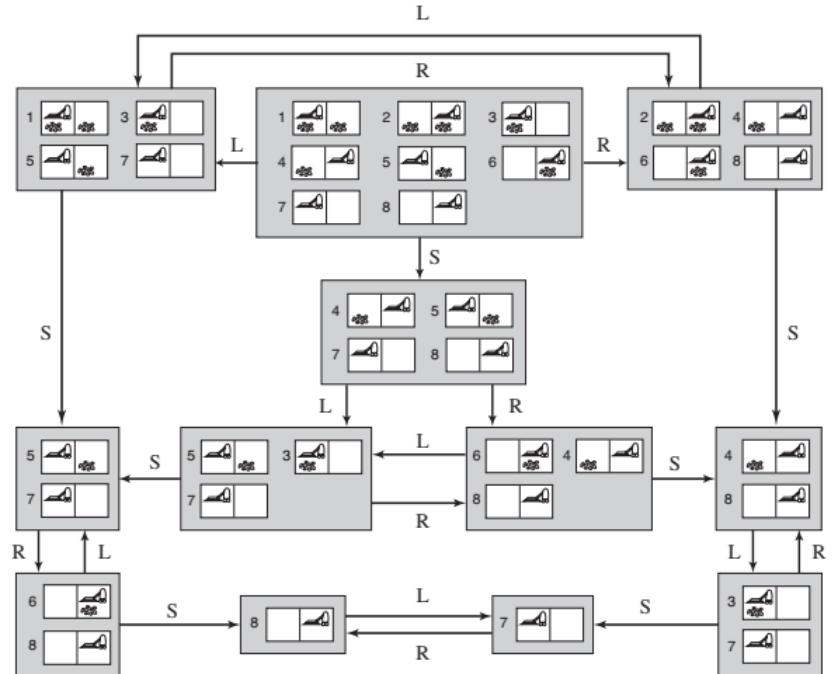
Tipos de problemas

Exemplo: o problema do aspirador (problema do estados múltiplos)

Se o aspirador não tem sensores, então não saberá onde está e onde está o lixo. Mesmo assim, conseguirá resolver o problema.

Solução? {1,2,3,4,5,6,7,8}

- Right – {2,4,6,8}
- Suck – {4,8}
- Left – {3,7}
- Suck – {7}



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

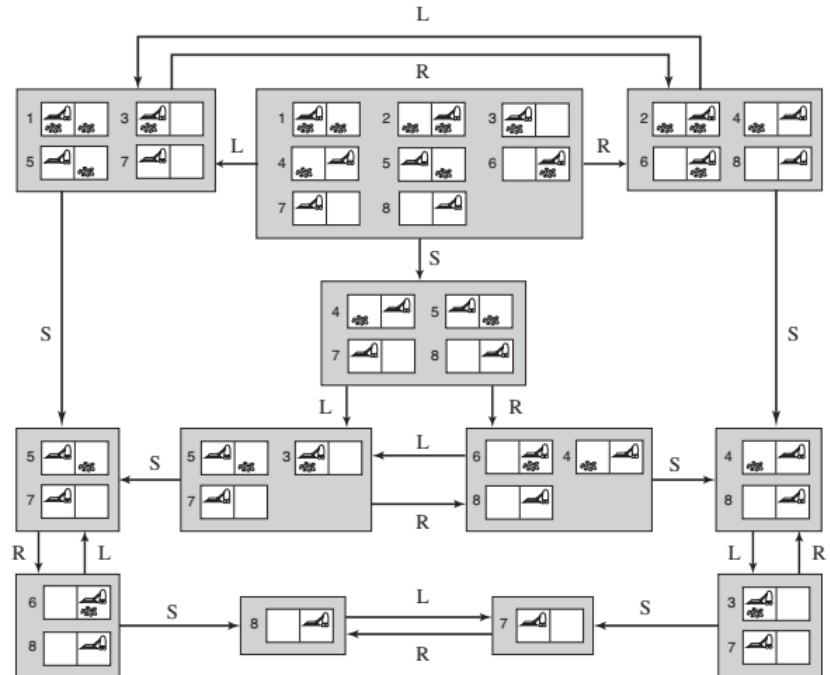
Tipos de problemas

Exemplo: o problema do aspirador

(problema do estado múltiplos)

Formulando o problema:

- Conjunto de Estados: subconjunto dos estados representados
- Operadores: esquerda, direita e aspirar
- Teste Objetivo: Todos os estados do conjunto não podem ter lixo
- Custo da Solução: Cada ação custa 1



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador

(problema de contingência)

O agente não sabe qual o efeito que terá suas ações, isto devido ao ambiente ser parcialmente observável.

Portanto, a sequência de ações deve ser planeada mediante cada percepção do ambiente, ou seja, por cada ação realizada o agente recolhe informação através do seu sensor e só depois decide a próxima ação a executar.

A solução será uma lista de ações condicionais que intercalará procura e execução.

Exemplo: Iniciar em {5} ou {7} – [Right, se tiver lixo então Suck]

Exemplo: o problema do aspirador

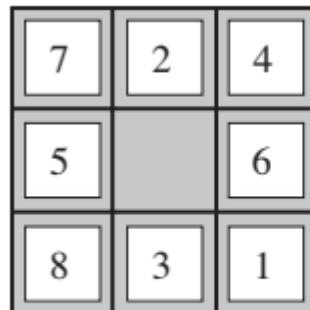
(problema de exploração)

Este é um caso extremo de um problema de contingência que é resolvido considerando informação adicional para uma solução que intercale procura e execução.

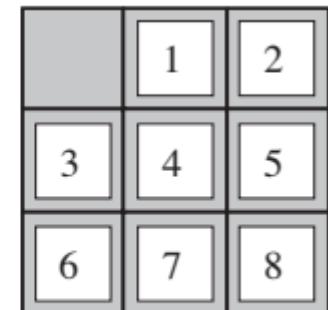
Exemplo de Problemas do Mundo Real

Problema puzzle de 8 peças

- Estados: descrição da localização das oito peças e quadrado em branco
- Estado Inicial: a configuração inicial do puzzle
- Ações: Movimentar o quadrado branco para esquerda, direita, cima e para baixo
- Estado Objetivo: estado que corresponde à configuração do puzzle à direita
- Custo: cada passo custa 1 unidade, o custo da solução é o número de passos para resolver o problema



Start State



Goal State

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo de Problemas do Mundo Real

Problema de Rotas/Caminhos

- Encontrar o melhor caminho de um ponto a outro (aplicações: google maps, redes de computadores, planeamento militar, viagens aéreas)
- Visitar cada ponto pelo menos uma vez num dado espaço (Ex: Caixeiro viajante visitar cada cidade exatamente uma vez, encontrar o caminho mais curto)

Outros:

- Navegação autónoma (com alguns graus de liberdade)
 - A dificuldade é incrementada rapidamente com o número de graus de liberdade. Possíveis complicações incluem: erros de percepção do ambiente, ambientes desconhecidos, etc.
- Sequênciação da montagem automática
 - Planeamento da montagem de objectos complexos (por robôs)
 - etc.

- Uma solução para um dado problema é definida como a sequência de ações desde o estado inicial até ao estado objetivo. A “qualidade” da solução é medida através da função do custo do caminho e pode ser:
 - Uma **solução satisfatória** se é uma qualquer solução;
 - uma **solução semi-óptima** é aquela que tem aproximadamente o menor custo entre todas as soluções;
 - uma **solução óptima** é aquela que tem o menor custo entre todas as soluções.

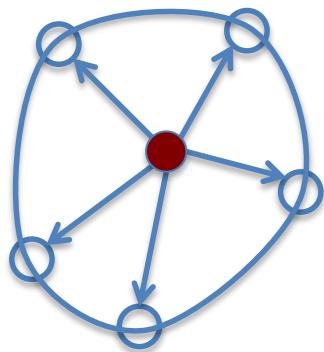
Procura Ideia básica

- Começando no estado inicial (nodo) e expandi-lo, fazendo uma lista de todos os possíveis estados sucessores;
- Manter uma lista de estados (nodos) não expandidos;
- Em cada etapa, escolha um estado da lista de estados não expandidos para expandir;
- Continue até atingir o estado objetivo;
- Tente expandir o menor número possível de estados.

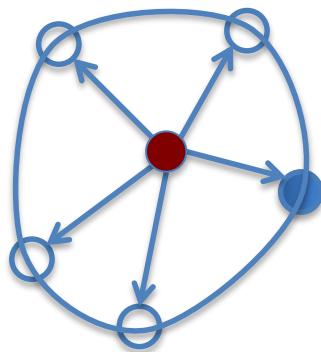


start

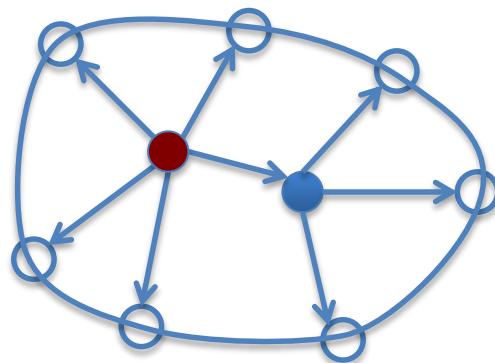
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



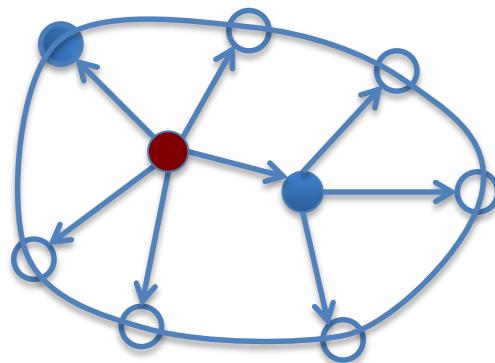
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



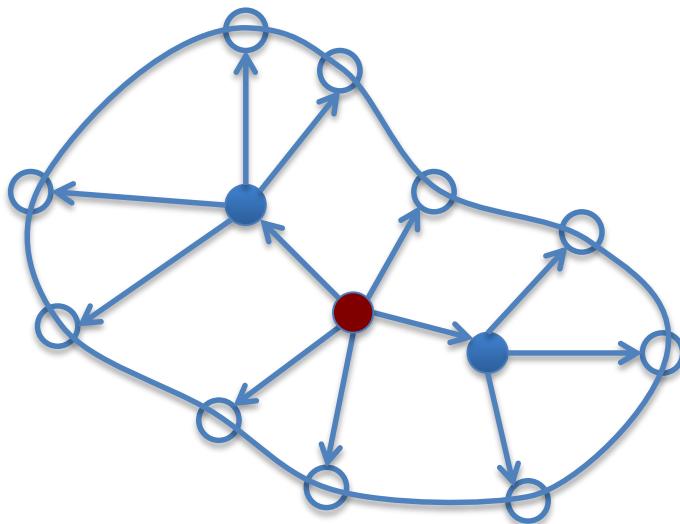
Fonte: Svetlana Lazebnik, 2017 Artificial Intelligence, University of Illinois.



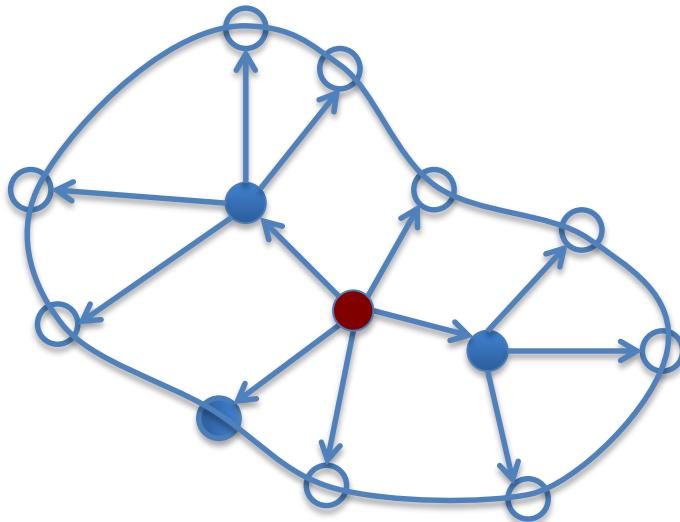
Fonte: Svetlana Lazebnik, 2017 Artificial Intelligence, University of Illinois.



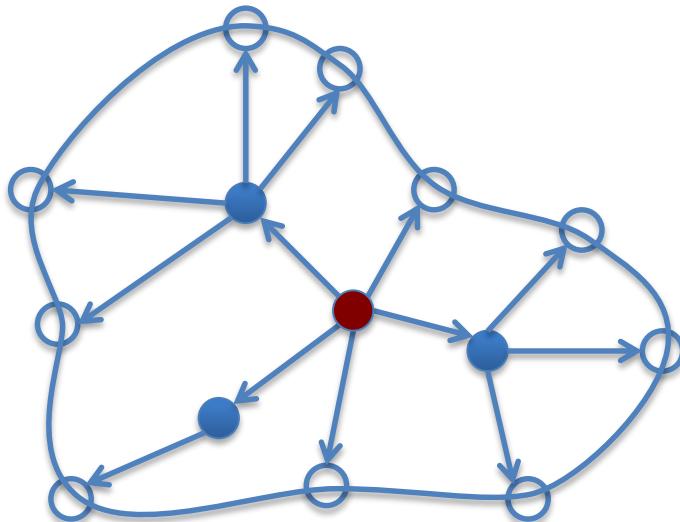
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



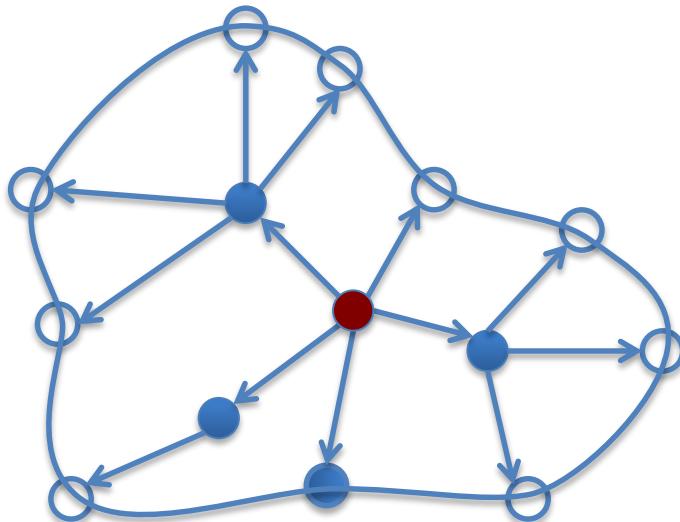
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

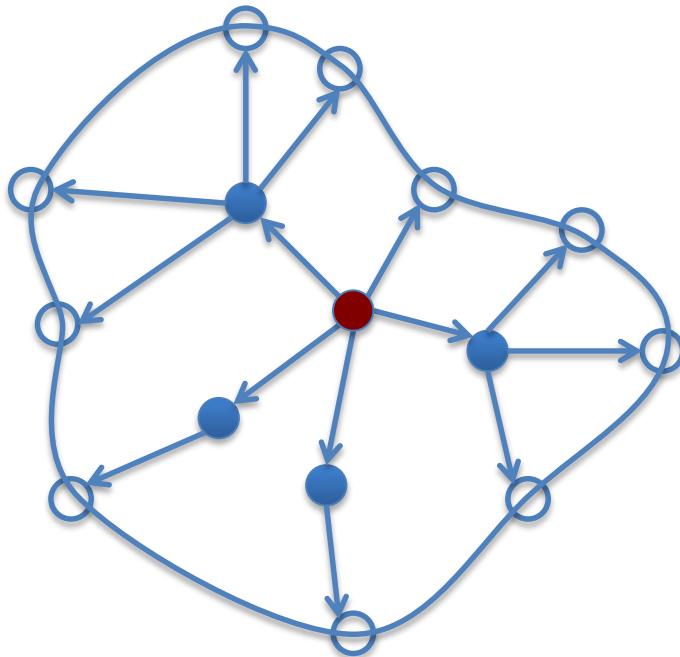


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



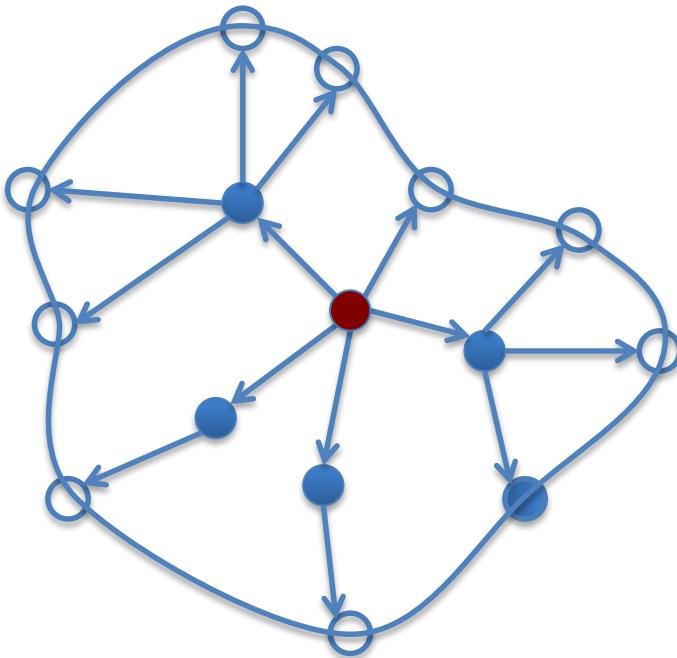
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search: Basic idea

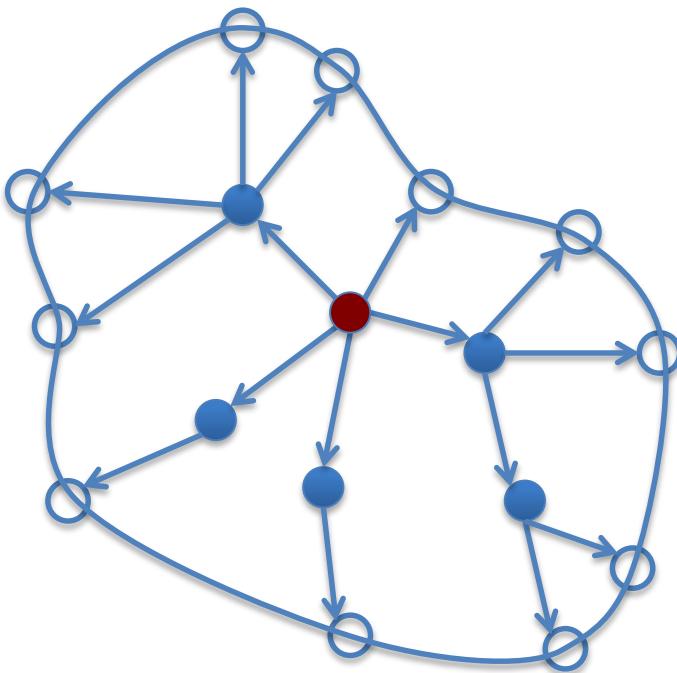


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search: Basic idea



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Metodologia para realizar a Pesquisa da Solução:

1. Começar com o estado inicial
2. Executar o teste do objetivo
3. Se não foi encontrada a solução, usar os operadores para expandir o estado atual gerando novos estados - sucessores (expansão)
4. Executar o teste do objetivo
5. Se não tivermos encontrado a solução, escolher qual o estado a expandir a seguir (estratégia de pesquisa) e realizar essa expansão
6. Voltar a 4)

Pesquisa da Solução Árvore de Pesquisa

- Através do **espaço de estados** do problema podemos formar uma **árvore de pesquisa** que nos auxilie a encontrar a solução.
- O **estado inicial** forma o **nó raiz** da árvores e os **ramos** de cada nó são as ações possíveis a partir do nó (estado) para as folhas (próximos estados)
- Portanto, é uma árvore de pesquisa composta por nós. Nós folhas, ou não têm sucessores ou ainda não foram expandidos.
- Importante distinguir entre a árvore de pesquisa e o espaço de estados!

Pesquisa da Solução - Árvore de Pesquisa

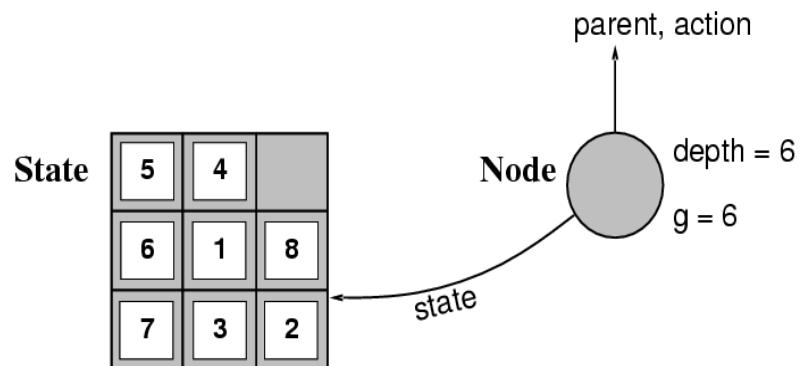
Estados versus Nós

- Um **estado** é a representação de uma configuração física
- Um **nó** é uma estrutura de dados constituído por parte da árvore de pesquisa que inclui **estado**, **nó pai***, **ação****, **custo do caminho $g(x)$ *****, **profundidade**.

*nó que lhe deu origem

** operador aplicado para o gerar

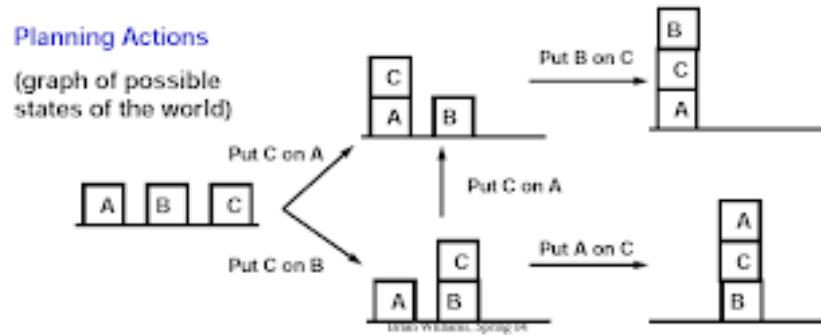
*** custo do caminho desde o nó inicial



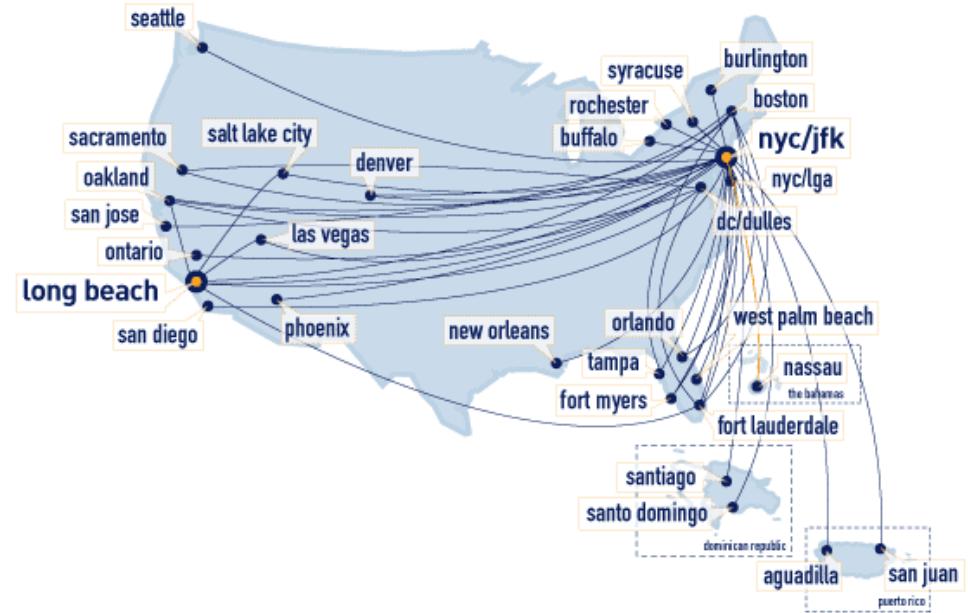
Ao expandir a árvore cria-se novos nós, preenchendo os vários campos e utilizando os “sucessores” do problema para criar os estados correspondentes

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplos de Grafos



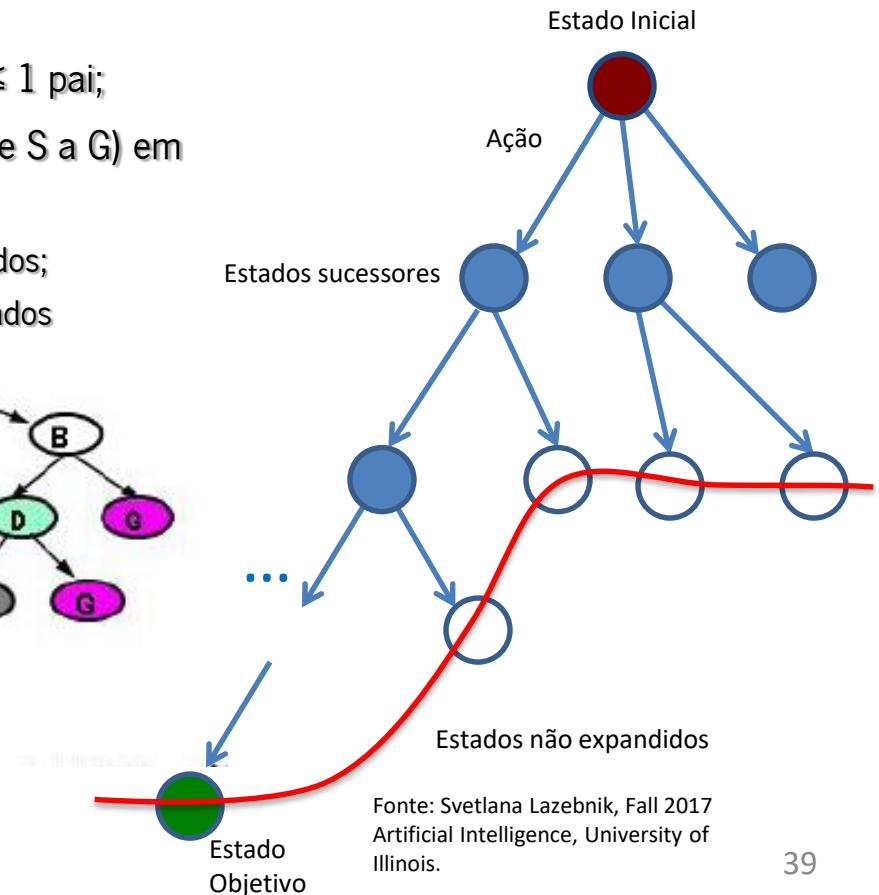
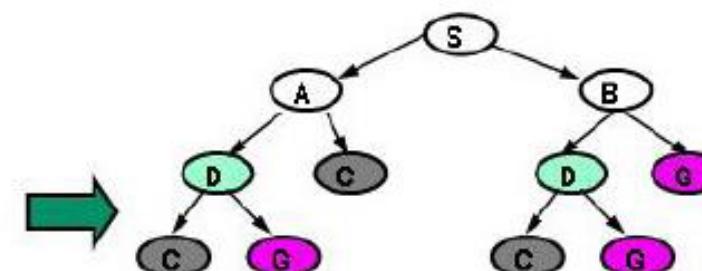
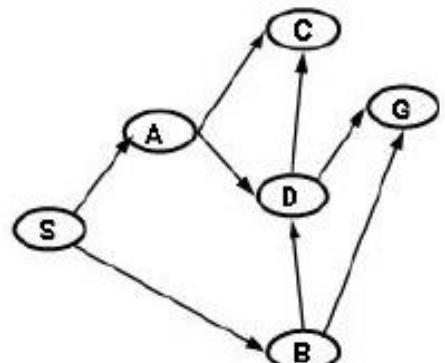
Fonte: Brian Williams, 2014



Fonte: Jet Blue airlines: <http://www.jetblue.com/travelinfo/routemap.html>

Grafos de procura como Árvores de procura

- Árvores são grafos sem ciclos e em que os nodos têm ≤ 1 pai;
- Podemos transformar problemas de pesquisa gráfica (de S a G) em problemas de pesquisa em árvore:
 - substituindo links não direcionados por 2 links direcionados;
 - evitando loops no caminho (ou monitorando os nós visitados)



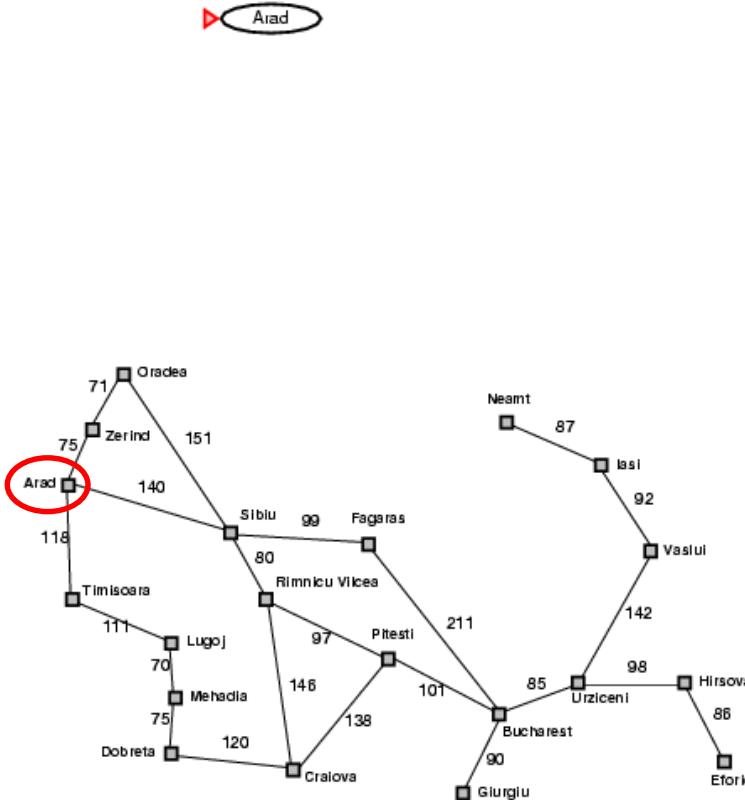
Fonte: https://ocw.mit.edu/courses/health-sciences-and-technology/hst-947-medical-artificial-intelligence-spring-2005/lecture-notes/ch2_search1.pdf

Algoritmo de pesquisa em árvore

- Inicialize a lista de estados não expandidos usando o estado inicial
- Enquanto a lista de estados não expandidos não estiver vazia
 - Escolha um nó de lista de estados não expandidos de acordo com a estratégia de pesquisa e remova-o da lista
 - Se o nó contiver o estado do objetivo, devolva a solução
 - Senão, expanda o nó e inclua seus filhos na lista de estados não expandidos

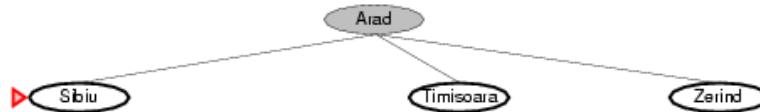
Exemplo de pesquisa em árvore

Inicial: Arad
Objetivo: Bucharest

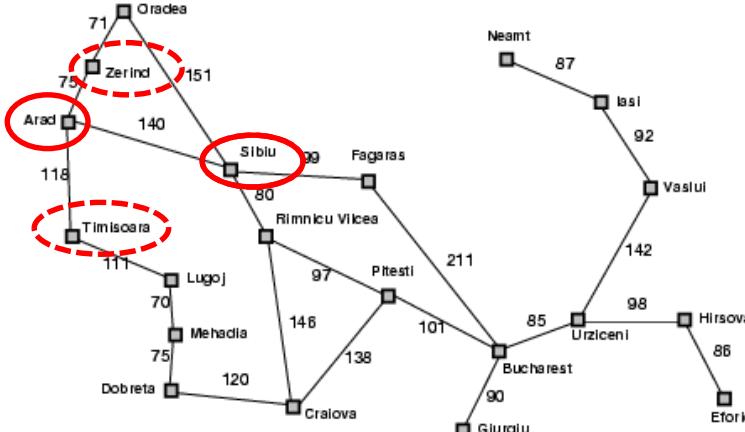


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore

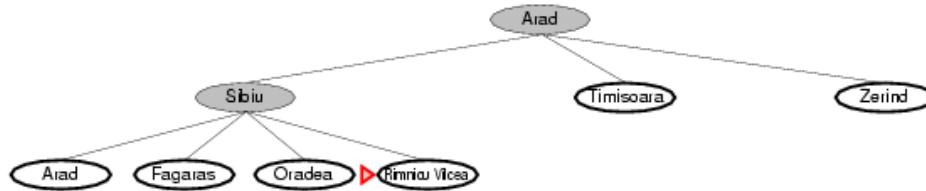


Inicial: Arad
 Objetivo: Bucharest

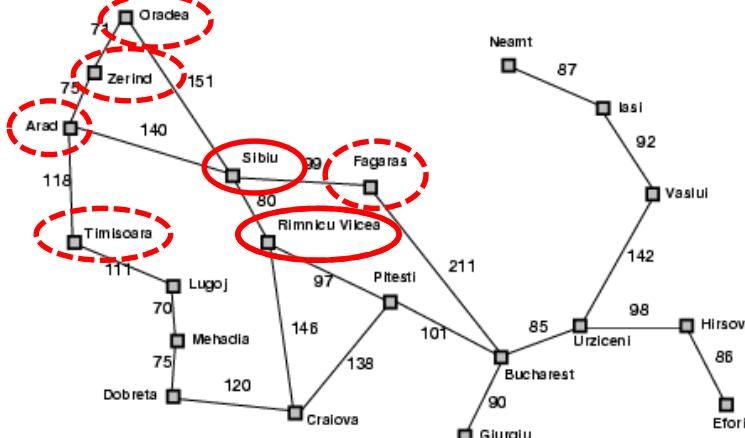


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore

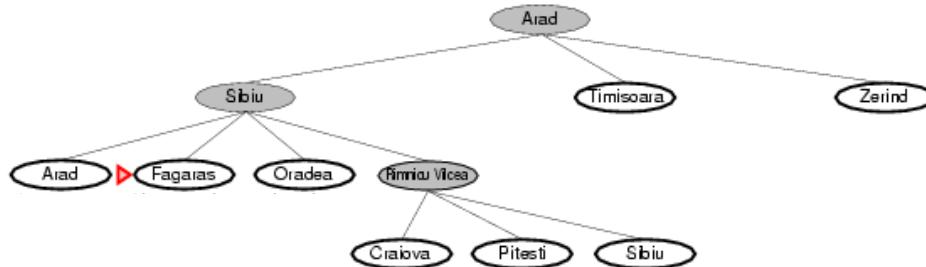


Inicial: Arad
 Objetivo: Bucharest

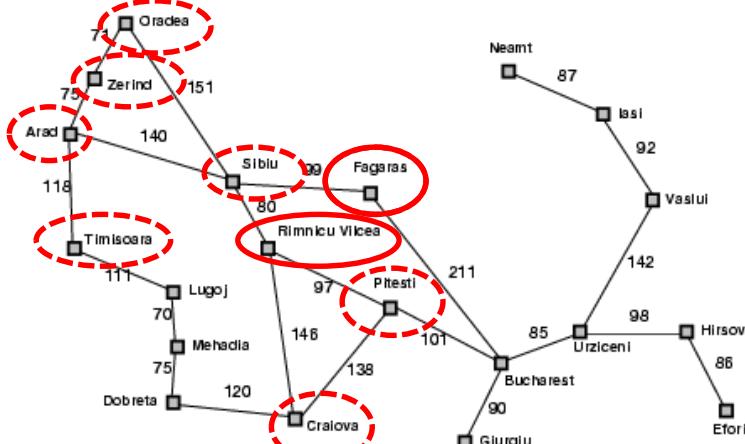


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore

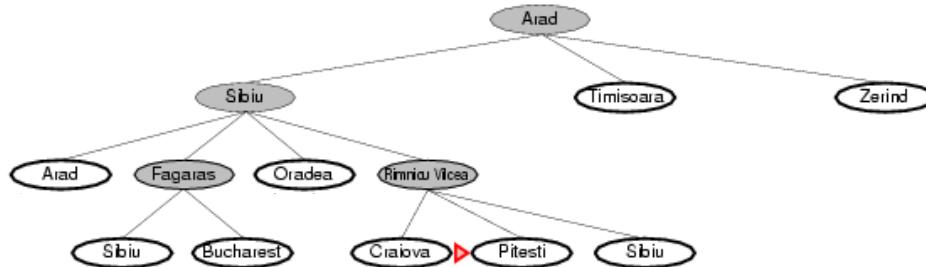


Inicial: Arad
 Objetivo: Bucharest

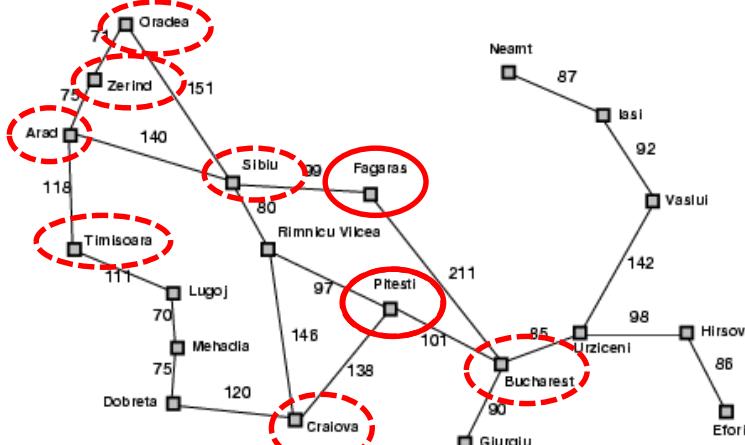


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore

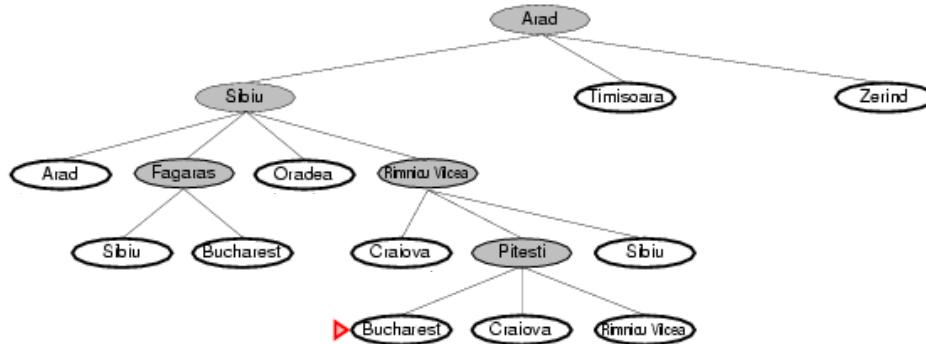


Inicial: Arad
 Objetivo: Bucharest

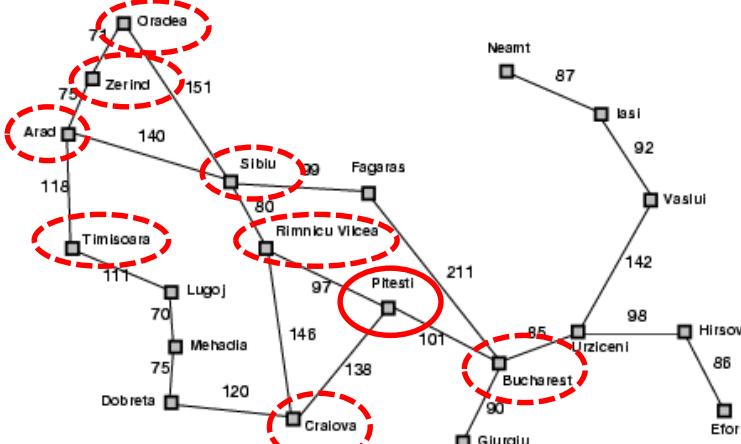


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore

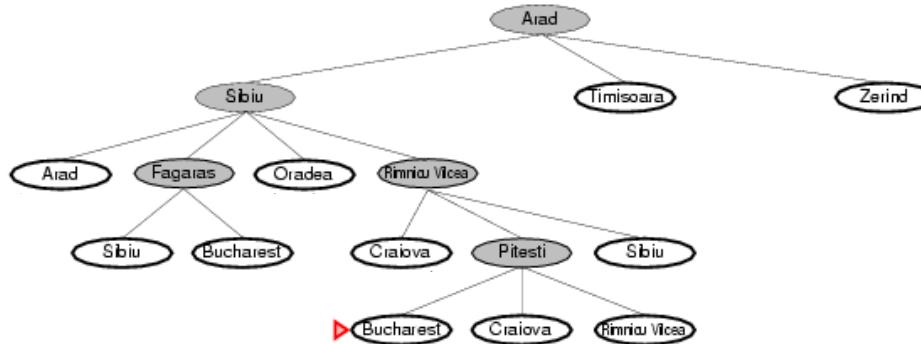


Inicial: Arad
 Objetivo: Bucharest

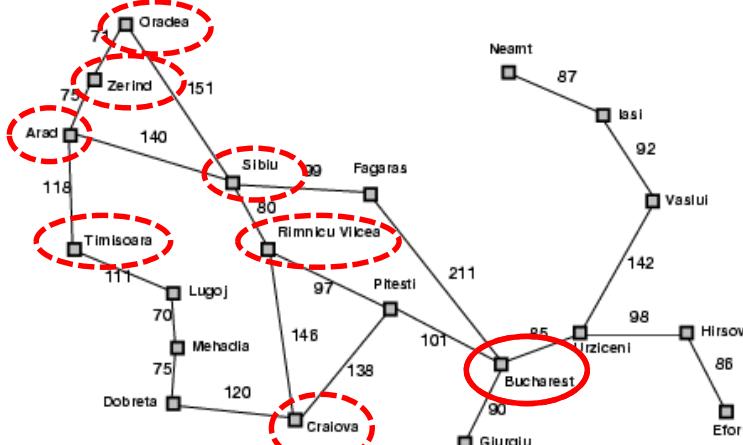


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Exemplo de pesquisa em árvore



Inicial: Arad
 Objetivo: Bucharest

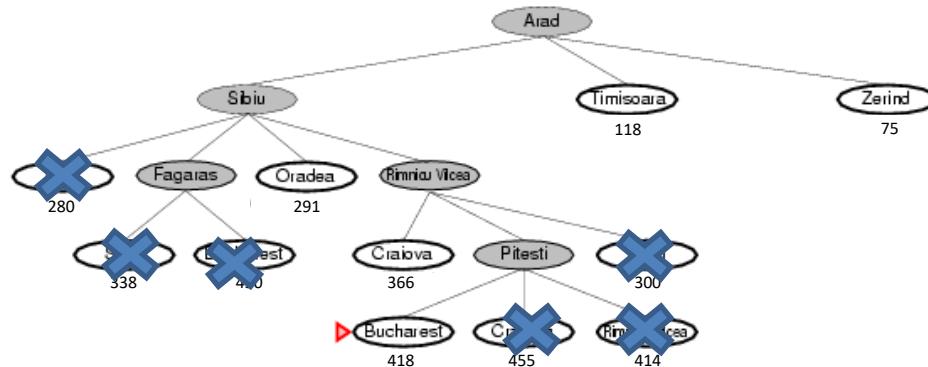


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

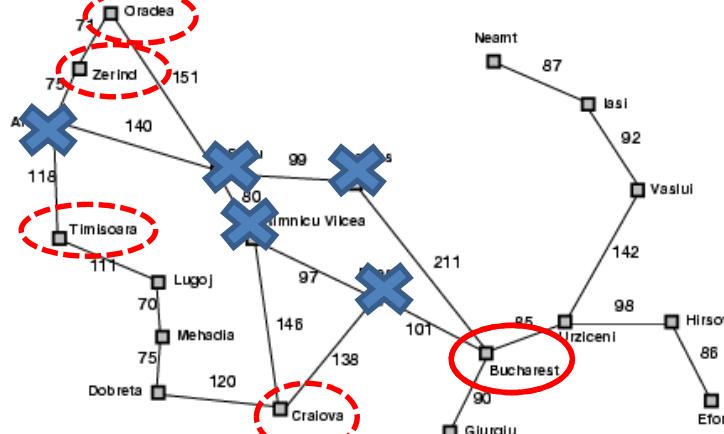
Algoritmo de pesquisa em árvore Manipulação de estados repetidos

- Inicialize a lista de estados não expandidos usando o estado inicial
- Enquanto a lista de estados não expandidos não estiver vazia
 - Escolha um nó de lista de estados não expandidos de acordo com a estratégia de pesquisa e remova-o da lista
 - Se o nó contiver o estado do objetivo, devolva a solução
 - Senão, expanda o nó e inclua seus filhos na lista de estados não expandidos
- Para lidar com estados repetidos:
 - Toda vez que expandir um nó, adicione esse estado ao conjunto explorado; não coloque estados explorados na lista de estados não expandidos novamente
 - Sempre que você adicionar um nó à lista de estados não expandidos, verifique se ele já existe na lista de estados não expandidos com um custo de caminho mais alto e, se sim, substitua esse nó pelo novo.

Pesquisa sem estados repetidos



Inicial: Arad
Objetivo: Bucharest



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Podemos avaliar o desempenho do algoritmo de pesquisa através dos seguintes critérios:

- Completude: Está garantido que encontra a solução?
- Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
- Complexidade no Espaço: Quanta memória necessita para fazer a pesquisa?
- Optimalidade: Encontra a melhor solução?

O tempo e a complexidade do espaço são sempre considerados tendo em conta a medição da dificuldade do problema (e.g. o tamanho do grafo e do espaço de estados).

- A estratégia: a ordem da expansão do nó
- Critérios de avaliação:
 - Completude: Está garantido que encontra a solução?
 - Otimalidade: Encontra a melhor solução?
 - Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
 - Complexidade no Espaço: Quanta memória necessita para fazer a pesquisa?
- O tempo e a complexidade do espaço são medidos em termos de:
 - b : o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa
 - d : a profundidade da melhor solução
 - m : a máxima profundidade do espaço de estados

Estratégias de Pesquisa

- **Tipos de Estratégias de Pesquisa:**

- Pesquisa Não-Informada (cega)

- Primeiro em Largura (BFS)
 - Primeiro em Profundidade (DFS)
 - Custo Uniforme
 - Pesquisa Iterativa
 - Pesquisa Bidirecional

- Pesquisa Informada (heurística)

- Pesquisa Gulosa
 - Algoritmo A*

- Uma **estratégia de pesquisa** é definida escolhendo a ordem da expansão do nó;
- **Estratégias de pesquisa não informadas** usam apenas as informações disponíveis na definição do problema;
- Nas **estratégias de pesquisa informadas** dá-se ao algoritmo “dicas” sobre a adequação de diferentes estados.

■ Pesquisa Primeiro em Largura (Breadth-first search)

- Estratégia: Todos os nós de menor profundidade são expandidos primeiro
- Bom: Pesquisa muito sistemática
- Mau: Normalmente demora muito tempo e sobretudo ocupa muito espaço

■ Propriedades:

- Completa: Sim, se b (fator de ramificação) for finito
- Tempo: supondo fator de ramificação b então $n=1+b+b^2+b^3+\dots+b^m = O(b^m)$ é exponencial em d
- Espaço: Guarda cada nó em memória $O(b^d)$
- Otimal: Sim, se o custo de cada passo for 1

■ Em geral só pequenos problemas podem ser resolvidos assim!

- b : o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa
- d : a profundidade da melhor solução
- m : a máxima profundidade do espaço de estados

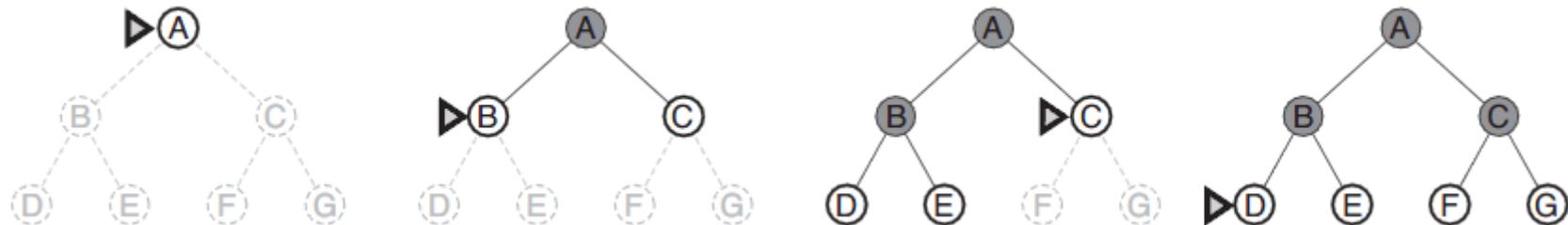
Pesquisa Primeiro em Largura

▪ Pesquisa Primeiro em Largura (Breadth-first search) (2)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier  $\leftarrow$  a FIFO queue with node as the only element
    explored  $\leftarrow$  an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child  $\leftarrow$  CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier  $\leftarrow$  INSERT(child, frontier)
```

Pesquisa Primeiro em Largura

- Pesquisa Primeiro em Largura (Breadth-first search) (3)



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

■ Pesquisa Primeiro em Profundidade (Depth-First Search)

- Estratégia: Expandir sempre um dos nós mais profundos da árvore
- Bom: Muito pouca memória necessária, bom para problemas com muita soluções
- Mau: Não pode ser usada para árvores com profundidade infinita, pode ficar presa em ramos errados

■ Propriedades:

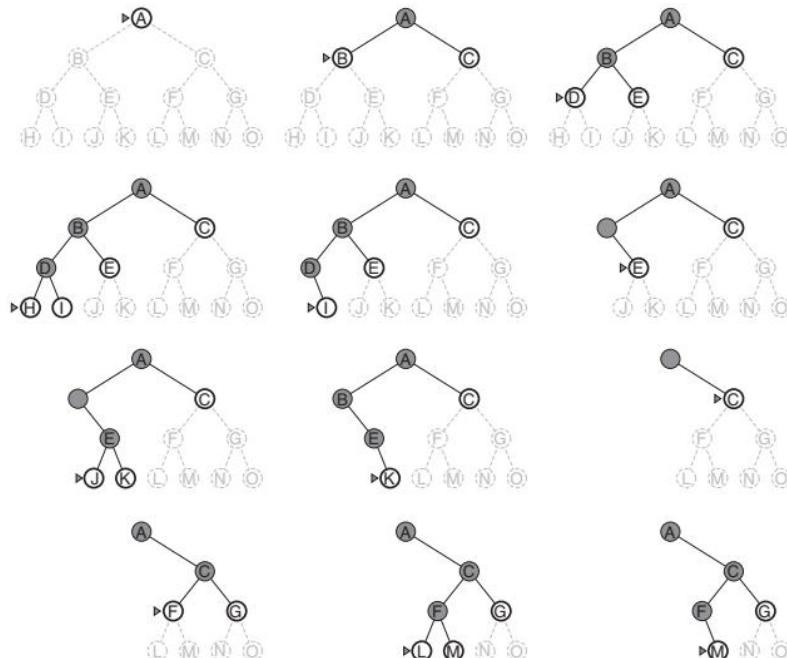
- Completa: Não, falha em espaços de profundidade infinita, com repetições (loops)
 - Modifique para evitar estados repetidos ao longo do caminho
- Tempo: $O(b^m)$, mau se $m > d$
- Espaço: $O(bm)$, espaço linear
- Otimal: Não (em princípio devolve a 1^a solução que encontra)

• b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa
• d: a profundidade da melhor solução
• m: a máxima profundidade do espaço de estados

■ Por vezes é definida uma profundidade limite (l) e transforma-se em Pesquisa com Profundidade Limitada

Pesquisa Primeiro em Profundidade

- Pesquisa Primeiro em Profundidade (Depth-First Search) (2)**



Para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que procura em largura;

No entanto, esta estratégia deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos.

Pesquisa Primeiro em Profundidade (Depth-First Search) Depth-limited search

= pesquisa em profundidade com limite de profundidade l , ie,
nós em profundidade l não têm sucessores

```

function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred?  $\leftarrow$  false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred?  $\leftarrow$  true
        else if result  $\neq$  failure then return result
    if cutoff-occurred? then return cutoff else return failure

```

Um dos problemas da pesquisa Primeiro em Profundidade prende-se com a incapacidade desta lidar com caminhos infinitos;

A Pesquisa em Profundidade com limite de profundidade procura evitar este problema fixando o nível máximo de procura.

▪ **Pesquisa Primeiro em Profundidade (Depth-First Search)**

- Escolha o primeiro elemento da lista de estados não expandidos
- Adicione extensões de caminho à frente da lista de estados não expandidos

▪ **Pesquisa Primeiro em Largura (Breadth-first search)**

- Escolha o primeiro elemento de lista de estados não expandidos
- Adicione extensões de caminho ao final da lista de estados não expandidos

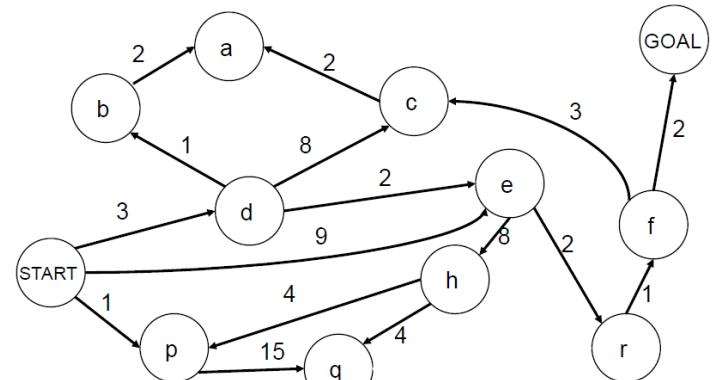
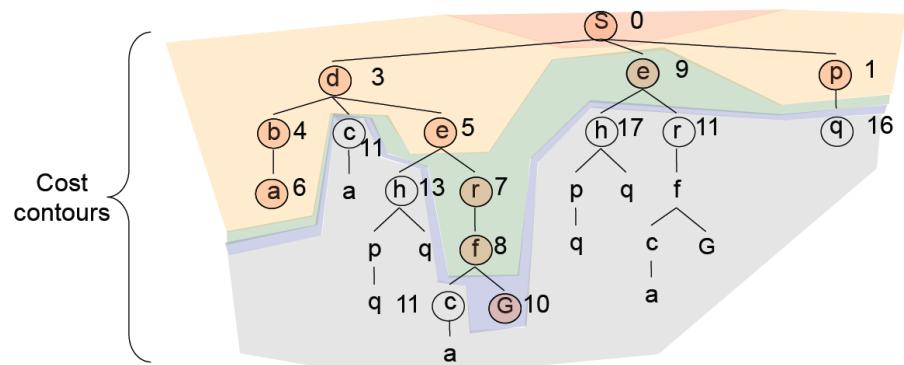
Pesquisa de Custo Uniforme

- Estratégia:
 - Para cada nó da lista de estados não expandidos, salve o custo total do caminho do estado inicial para esse nó
 - Expandir sempre o nó com menor custo da lista de estados não expandidos (medido pela função de custo da solução)
- Pesquisa Primeiro em Largura é igual a Pesquisa de Custo Uniforme se $g(N) = \text{Depth}(N)$
- Equivalente a **Pesquisa Primeiro em Largura (Breadth-first search)**, se a os custos todos iguais
- Implementação: de lista de estados não expandidos é uma fila prioritária ordenada pelo custo do caminho
- Temos de garantir que $g(\text{sucessor}) \geq g(N)$
- Equivalente ao algoritmo de Dijkstra em geral

Em todos os nós N , $g(N)$ é o custo conhecido de ir da raiz até ao nó N .

Pesquisa de Custo Uniforme (2)

- Ordem de expansão:
(S,p,d,b,e,a,r,f,e,G)



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Pesquisa de Custo Uniforme (3)



Fonte https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs

Pesquisa de Custo Uniforme

- Propriedades:

- Completa: Sim, se o custo da etapa for maior que alguma constante positiva ε (não queremos sequências infinitas de etapas com um custo total finito)
- Tempo:
 - *Número de nós com custo de caminho \leq custo da solução ideal (C^*)*, $O(b^{C^*/\varepsilon})$
 - *Isso pode ser maior que O(bd): a pesquisa pode explorar caminhos longos que consistem em pequenos passos antes de explorar caminhos mais curtos que consistem em passos maiores*
- Espaço: $O(b^{C^*/\varepsilon})$
- Otimal: Sim
 - b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa
 - d: a profundidade da melhor solução
 - m: a máxima profundidade do espaço de estados

Pesquisa Iterativa Aprofundamento Progressivo

Se não conhecemos o valor limite máximo, estaremos condenados a uma estratégia de procura em profundidade primeiro e temos que lidar com o problema de eventuais caminhos infinitos. A resposta passa pela alteração do princípio da procura limitada fazendo variar esse limite entre zero e infinito.

Use **Pesquisa Primeiro em Profundidade (Depth-First Search)** como uma sub-rotina

- Verifique a raiz
- Faça um DFS procurando um caminho de comprimento 1
- Se não houver um caminho de comprimento 1, faça um DFS procurando um caminho de comprimento 2
- Se não houver um caminho de comprimento 2, faça um DFS procurando um caminho de comprimento 3...

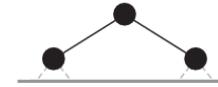
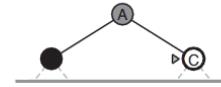
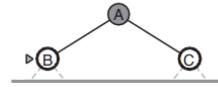
Pesquisa Iterativa Aprofundamento Progressivo

■ Pesquisa em Profundidade Iterativa (2)

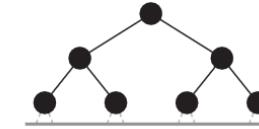
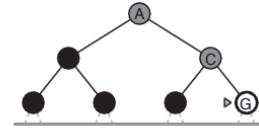
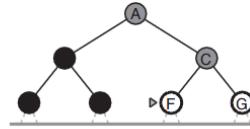
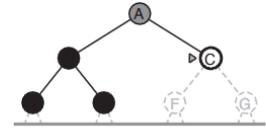
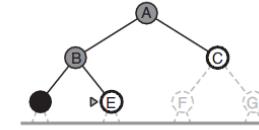
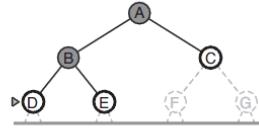
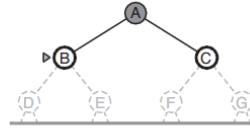
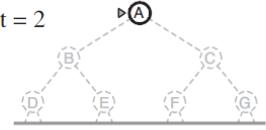
Limit = 0



Limit = 1

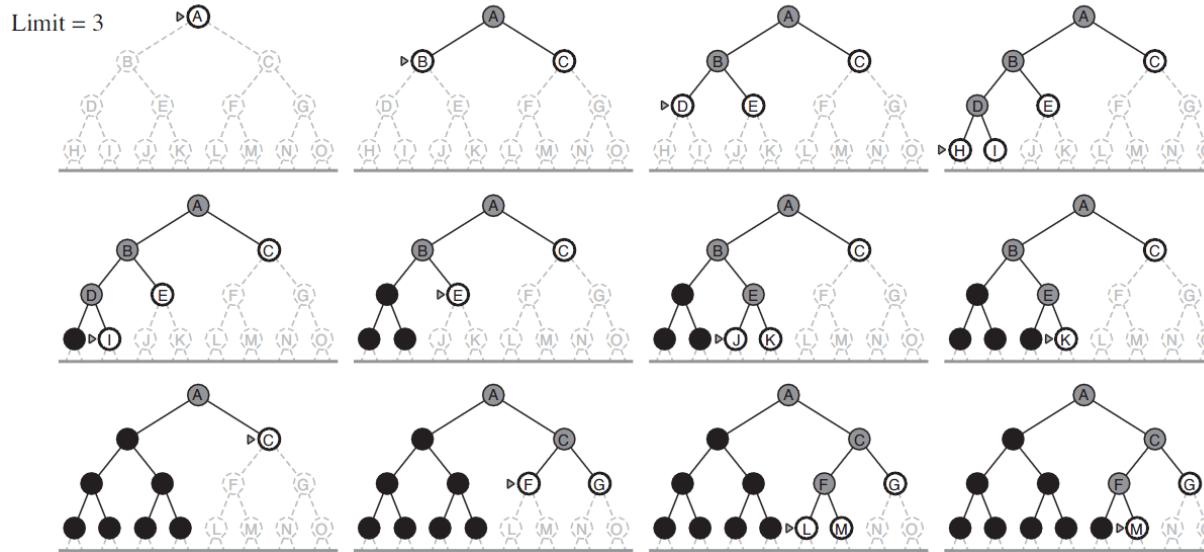


Limit = 2



Pesquisa Iterativa Aprofundamento Progressivo

■ Pesquisa em Profundidade Iterativa (3)



O algoritmo de procura por iterativa é uma boa opção para problemas em que somos obrigados a recorrer a um método cego.

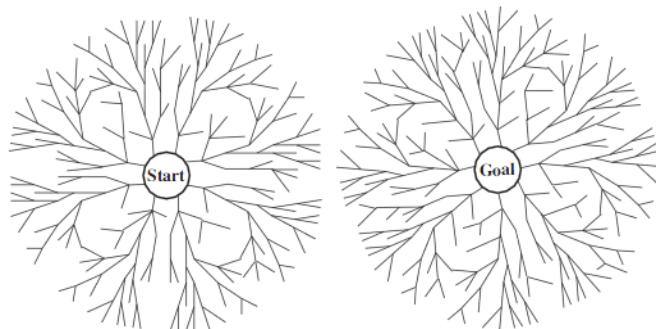
O espaço de procura é grande, mas não sabemos qual é o nível máximo em que pode estar uma solução.

Pesquisa Iterativa Aprofundamento Progressivo

- Estratégia: Executar pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade
 - Propriedades:
 - Completa: Yes
 - Tempo: $(d+1)b^0 + d b^1 + (d-1)b^2 + \dots + b^d = O(b^d)$
 - Espaço: $O(b^d)$
 - Ottimal: Sim, se o custo for 1.
 - Em geral é a melhor estratégia (não-informada ou cega) para problemas com um grande espaço de pesquisa e em que a profundidade da solução não é conhecida
- b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa
• d: a profundidade da melhor solução
• m: a máxima profundidade do espaço de estados

Pesquisa Bidirecional

- Estratégia: Executar pesquisa para a frente desde o estado inicial e para trás desde o objetivo, simultaneamente
 - Bom: Pode reduzir enormemente a complexidade no tempo $O(b^{d/2})$
 - Problemas: Será possível gerar os predecessores? E se existirem muitos estados objetivo? Como fazer o “matching” entre as duas pesquisas? Que tipo de pesquisa fazer nas duas metades?
- Eg., Para encontrar uma rota na Romênia, existe apenas um estado objetivo, portanto, a pesquisa para trás é muito parecida com a pesquisa para frente; Mas se o objetivo é uma descrição abstrata, como o de que “nenhuma rainha ataca outra rainha” no problema das rainhas n, a pesquisa bidirecional é difícil de usar.



Comparação entre Estratégias de Pesquisa

- Avaliação das estratégias de pesquisa:

- B é o fator de ramificação
- d é a profundidade da solução
- m é a máxima profundidade da árvore
- l é a profundidade limite de pesquisa

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

- **Pesquisa informada (Heurística)**

- Utiliza informação do problema para evitar que o algoritmo de pesquisa fique “perdido vagueando no escuro”

- **Estratégia de Pesquisa:**

- Definida escolhendo a ordem de expansão dos nós

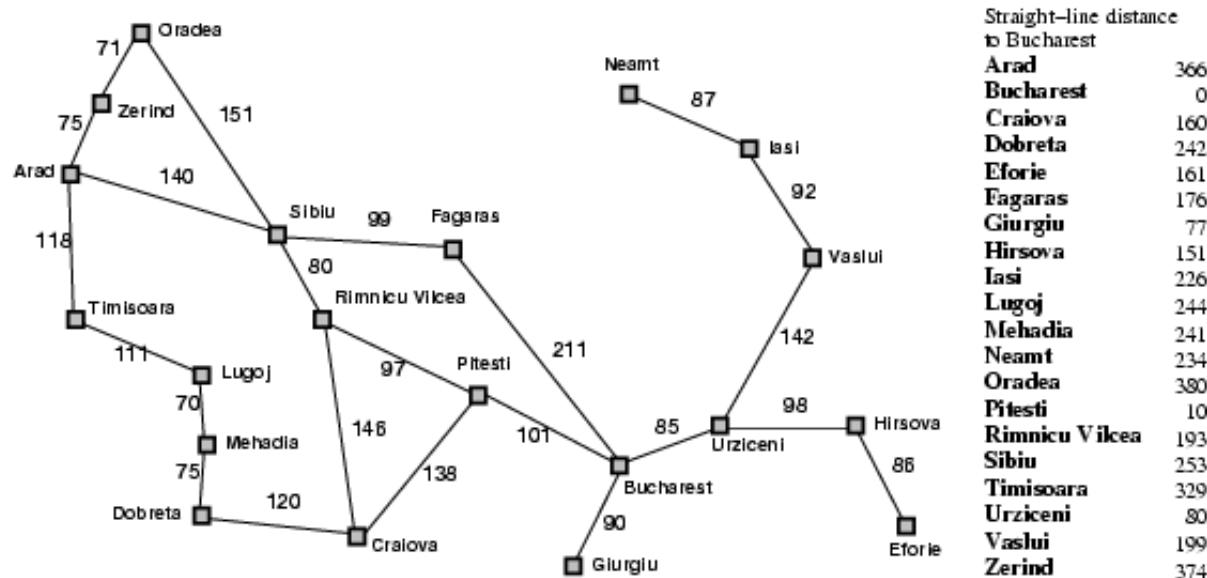
- **Pesquisa do Melhor Primeiro (Best-First Search)**

- Utiliza uma função de avaliação que retorna um número indicando o interesse de expandir um nó
 - Pesquisa Gulosa (Greedy-Search) – $f(n) = h(n)$ (função que estima distância à solução)
 - Algoritmo A* - $f(n) = g(n) + h(n)$ (estima o custo da melhor solução que passa por n)

- Como técnica de procura para a obtenção de metas em problemas não algorítmicos que geram “explosões” combinatórias;
 - Como um método aproximado de resolução de problemas utilizando funções de avaliação de tipo heurística (dica);
 - Como um método de poda (corte) para estratégias de programas de jogos.
-
- Numa procura, podemos aplicar dois tipos genéricos de heurísticas, sobre a decisão sobre qual nó será feita a expansão e sobre a decisão sobre quais os nós que devem ser descartados.
 - Se o universo é totalmente conhecido, a heurística será realizada através da atribuição de números;
 - Se o universo não é totalmente conhecido, no qual a heurística será realizada através da aplicação de regras.
-
- As heurísticas são específicas para cada problema. Estas funções podem ser pouco certas ou podem não encontrar a melhor resposta, no entanto a sua utilização permite a libertação do uso das análises combinatórias.
 - A implementação de uma heurística é difícil! É difícil medir precisamente o valor de uma determinada solução e é difícil medir determinados conhecimentos de forma a permitir que seja efetuada uma análise matemática do seu efeito no processo de busca.

Heurística para o problema da Romênia

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



▪ Pesquisa informada (Pesquisa Gulosa – Greedy-Search)

- **Estratégia:**

- Expandir o nó que parece estar mais perto da solução

- $h(n)$ = custo estimado do caminho mais curto do estado n para o objetivo (função heurística)

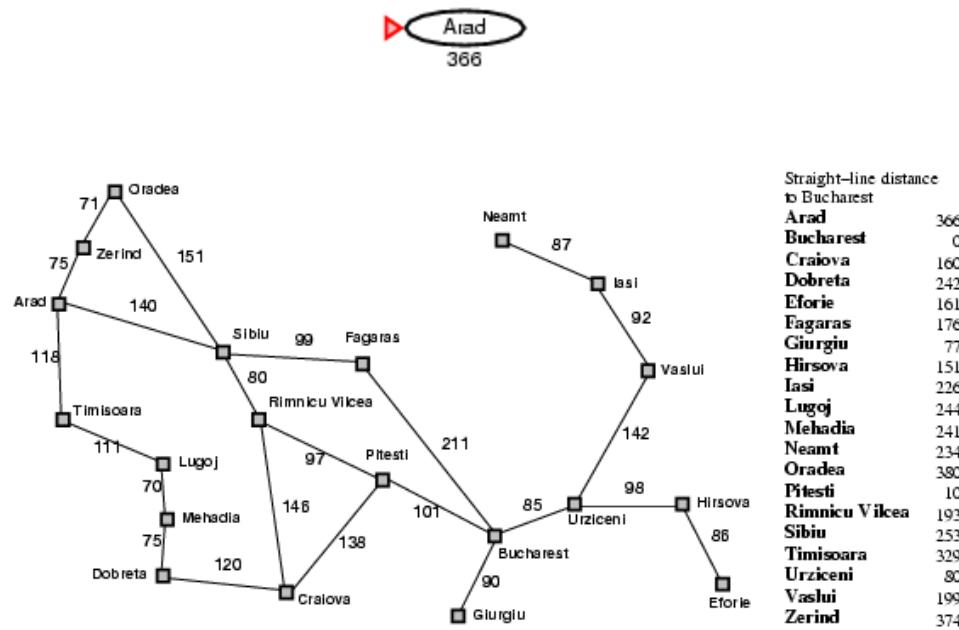
- function GREEDY-SEARCH(problem) returns a solution or failure
 - return BEST-FIRST-SEARCH(problem,h)

- **Exemplo:**

- $h_{SLD}(n)$ = distância em linha reta entre n e o objetivo

Pesquisa Gulosa Greedy-Search

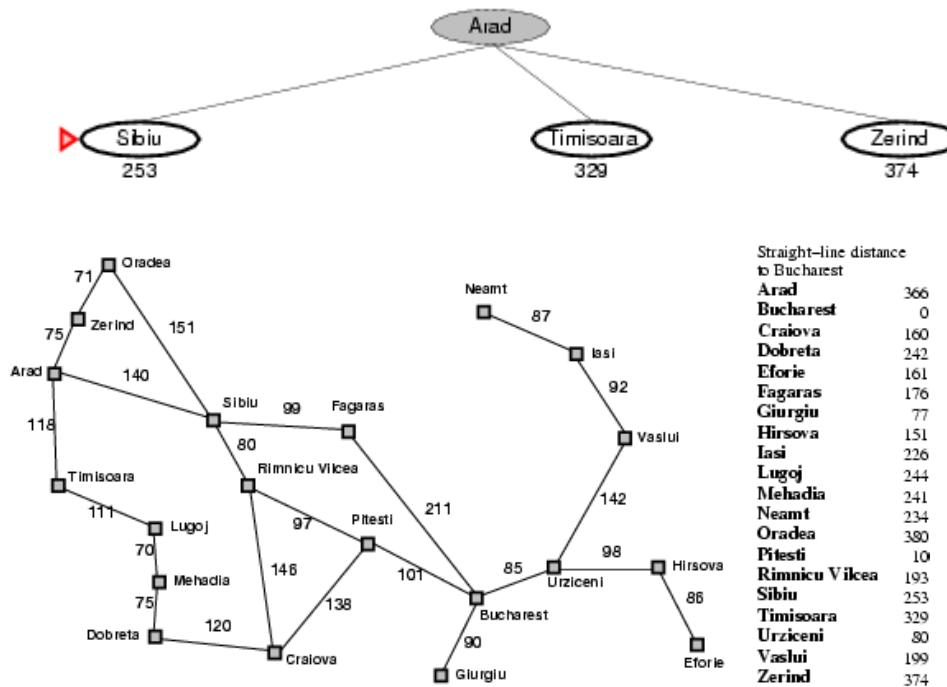
- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Pesquisa Gulosa Greedy-Search

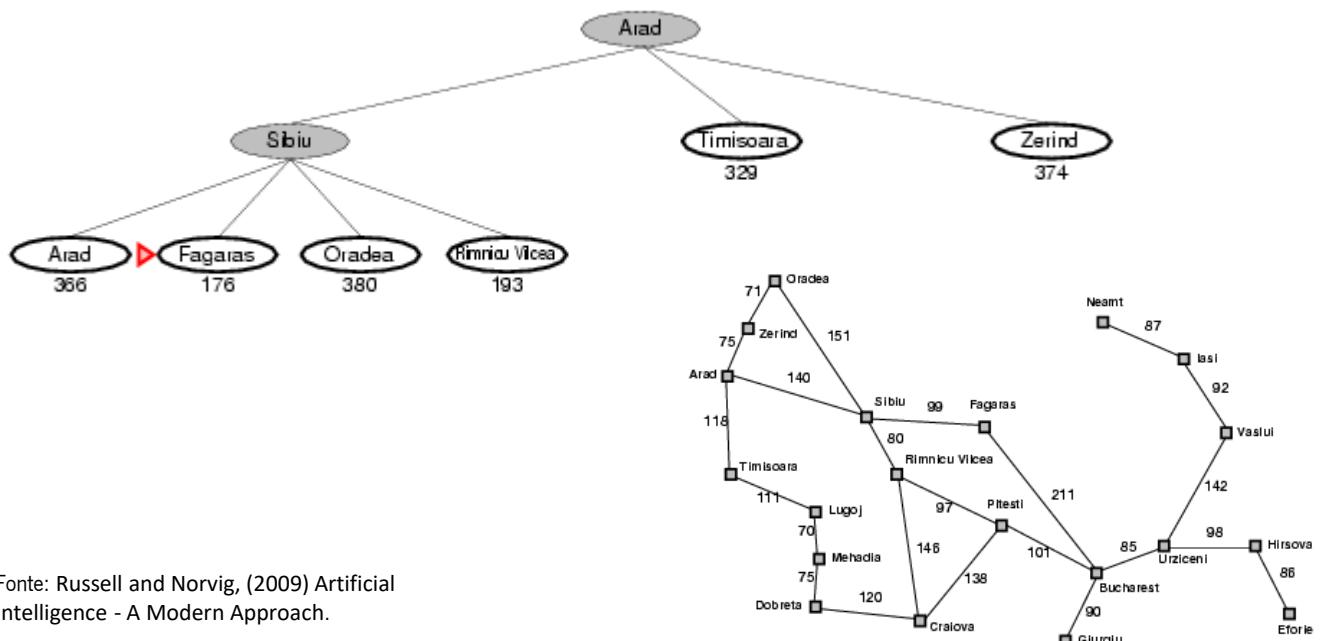
- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

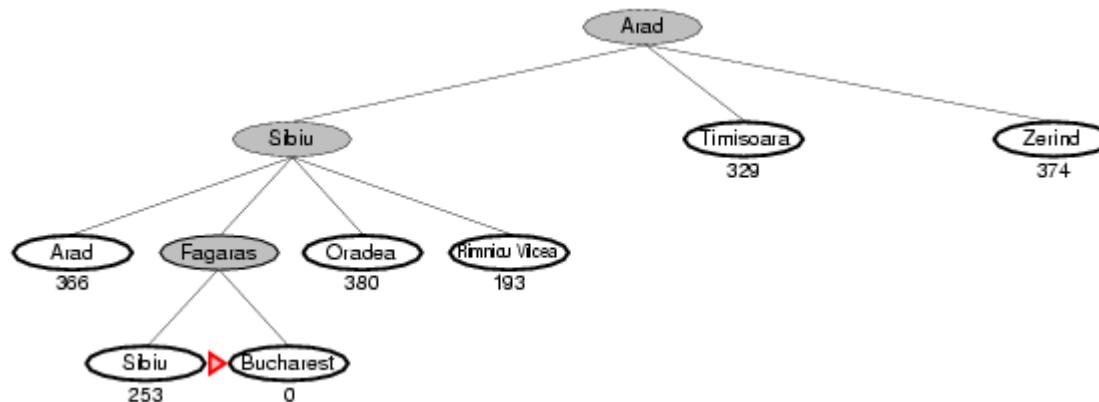
Pesquisa Gulosa Greedy-Search

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

■ Propriedades

- Completa? Não, pode entrar em ciclos.
 - Suscetível a falsos começos
- Complexidade no tempo? $O(b^m)$
 - mas com uma boa função heurística pode diminuir consideravelmente
- Complexidade no espaço? $O(b^m)$
 - Mantém todos os nós na memória
- Ótima? Não, não encontra sempre a solução ótima.
- Necessário detetar estados repetidos.

- **Pesquisa informada (Pesquisa A*)**

- **Estratégia:**

- evite expandir caminhos que são caros
- O algoritmo A* combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. Usa a função:

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

- $g(n)$ = custo total, até agora, para chegar ao estado n (custo do percurso)
- $h(n)$ = custo estimado para chegar ao objetivo (não deve sobreestimar o custo para chegar à solução (heurística))

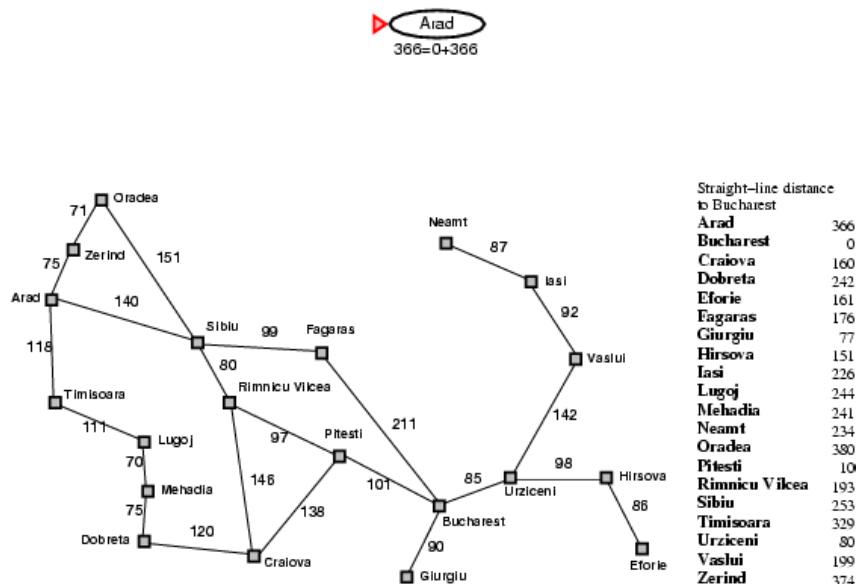
$f(n)$ = custo estimado da solução mais barata que passa pelo nó n

- **function** A*-SEARCH(problem) **returns** a solution or failure

return BEST-FIRST-SEARCH(problem,g+h)

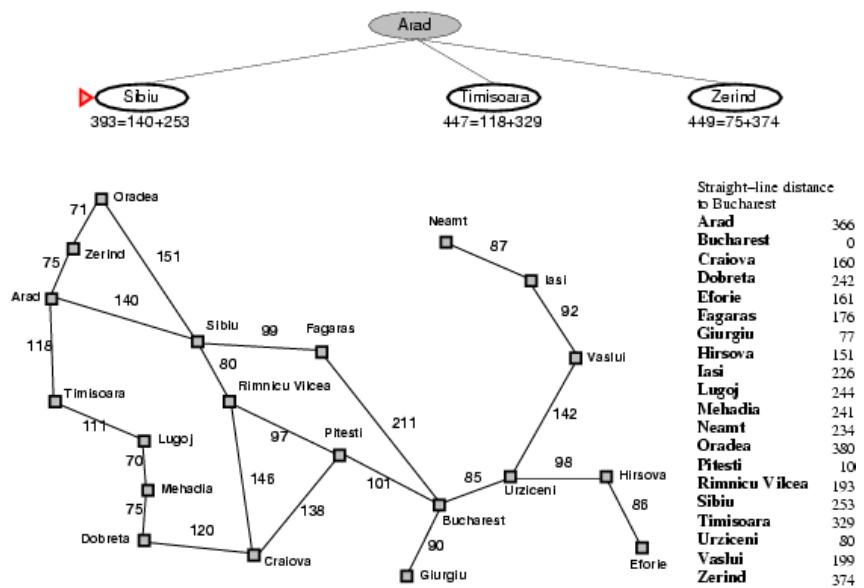
- Algoritmo A* é ótimo e completo.
- Complexidade no tempo exponencial em (erro relativo de h^* comprimento da solução)
- Complexidade no espaço: Mantém todos os nós em memória.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



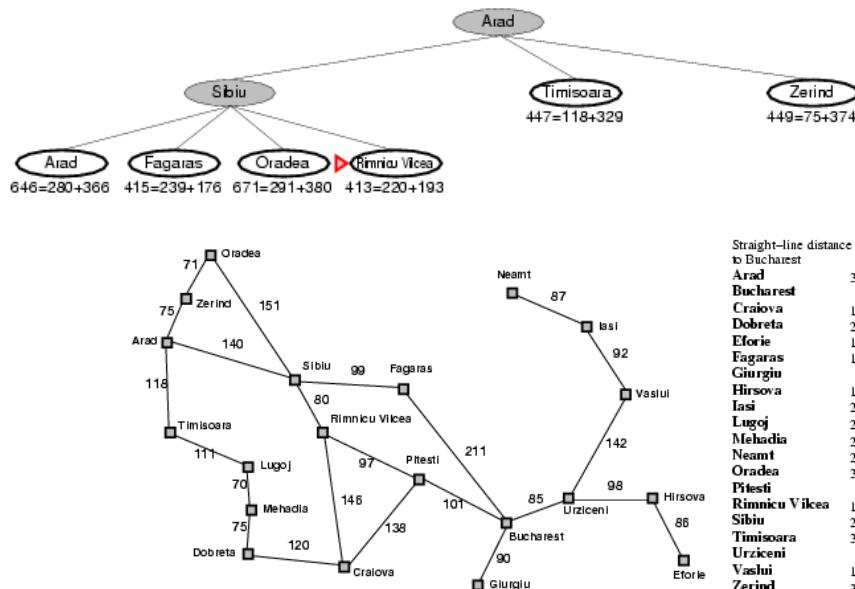
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
 - $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



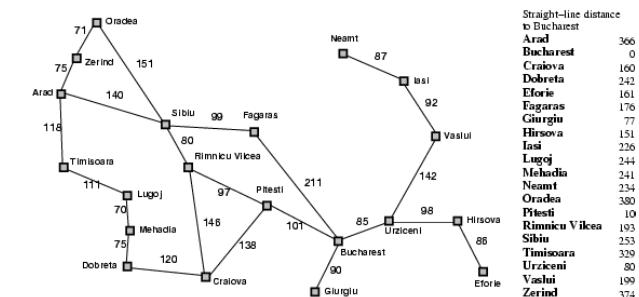
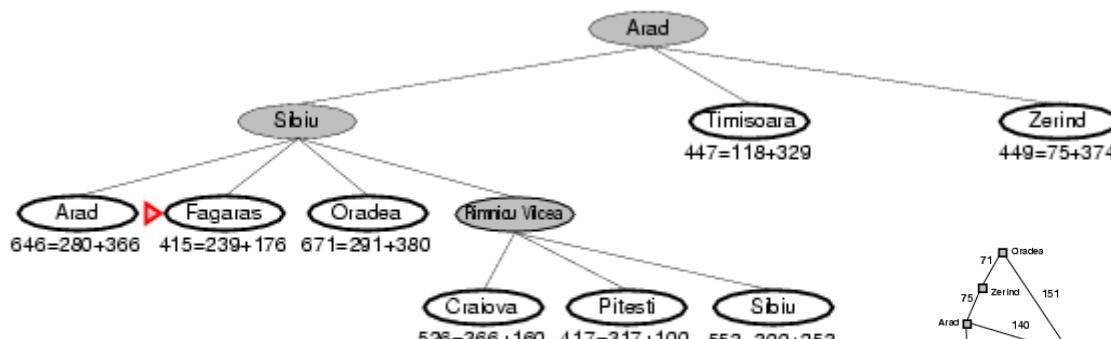
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



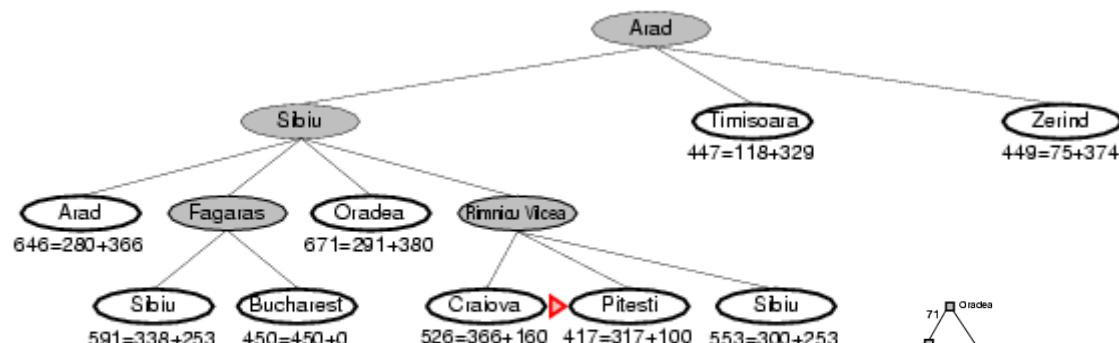
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo

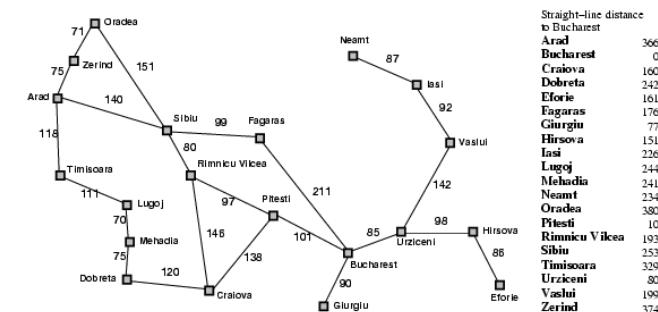


Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

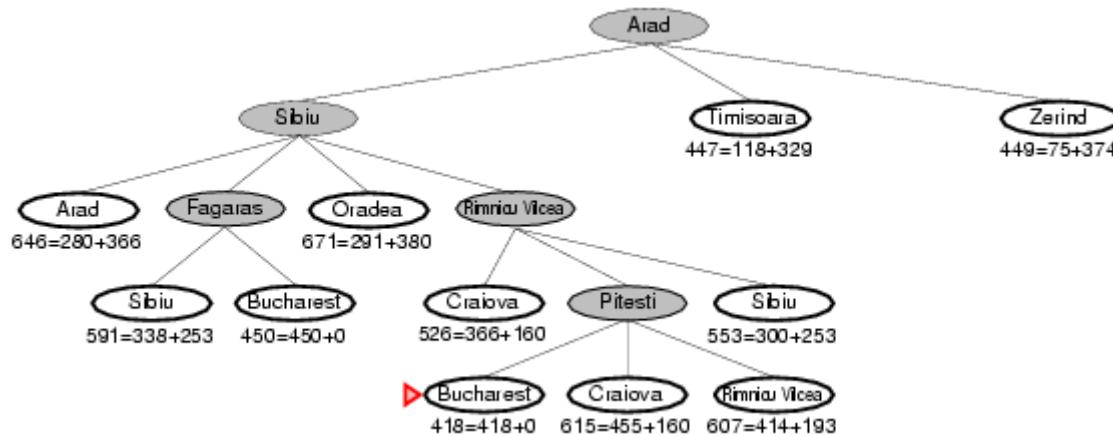
- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.



- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Pesquisa A* Exemplo (2)



•

Fonte:

https://en.wikipedia.org/wiki/A*_search_algorithm

Projetando funções heurísticas

■ Heurísticas – 8 Puzzle

- Solução típica em 20 passos com fator de ramificação médio: 3
- Número de estados: $320 = 3.5 \times 10^9$
- N° Estados (sem estados repetidos) = $9! = 362880$
- Heurísticas:
 - $H_1(n)$ = N° de peças fora do sítio
 - $H_2(n)$ = Soma das distâncias das peças até às suas posições corretas

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Projetando funções heurísticas

▪ Heurísticas – 8 Puzzle (2)

o Relaxamento de Problemas como forma de inventar heurísticas:

- Peça pode-se mover de A para B se A é adjacente a B e B está vazio
- a) Peça pode-se mover de A para B se A é adjacente a B
- b) Peça pode-se mover de A para B se B está vazio
- c) Peça pode-se mover de A para B

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

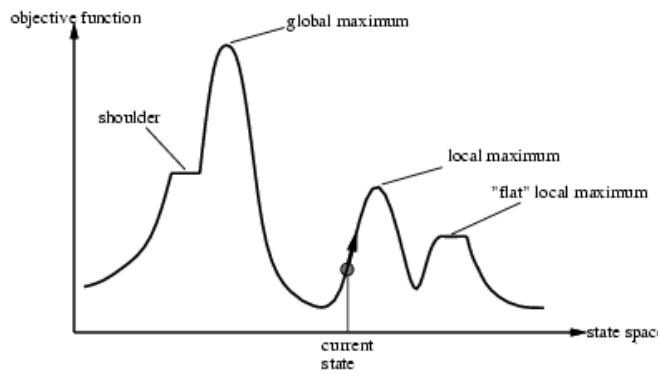
Goal State

- **Pesquisa com Memória Limitada - IDA*/SMA***
- IDA* - Pesquisa com Profundidade Iterativa (Iterative Deepening Search)
 - Estratégia: Utilização de um custo limite em cada iteração e realização de pesquisa em profundidade iterativa
 - Problemas em alguns problemas reais com funções de custo com muitos valores
- SMA* - Pesquisa Simplificada com Memória Limitada (Simplified Memory Bounded A*)
 - IDA* de uma iteração para a seguinte só se lembra de um valor (o custo limite)
 - SMA* utiliza toda a memória disponível, evitando estados repetidos
 - Estratégia: Quando necessita de gerar um sucessor e não tem memória, esquece um nó da fila que pareça pouco prometedor (com um custo alto).

Algoritmo	Completo	Otimal	Tempo complexidade	Espaço complexidade
Primeiro em Largura (BFS)	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(b^d)$
Primeiro em Profundidade (DFS)	No	Não	$O(b^m)$	$O(bm)$
Pesquisa Iterativa	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(bd)$
Custo Uniforme	Yes	Sim	Number of nodes with $g(n) \leq C^*$	
Greedy	No	Não	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

Algoritmos de Pesquisa Local e Problemas de Otimização

- Até agora abordamos, na essência, uma única categoria de problemas: ambientes observáveis, determinísticos e conhecidos, onde a solução é uma sequência de ações.
- Nem todos os ambiente são assim!
- Algoritmos que executam pesquisa puramente local no espaço de estados, avaliando e modificando um ou mais estados atuais, em vez de explorar sistematicamente caminhos a partir de um estado inicial.
- Esses algoritmos são adequados para problemas nos quais tudo o que importa é o estado da solução, não o custo do caminho para alcançá-lo.



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Para além da pesquisa Clássica Algoritmos de Melhoria Iterativa

- Em muitos problemas de otimização, o caminho para o objetivo é irrelevante.
- Espaço de Estados = conjunto das configurações completas.
- Algoritmos Iterativos mantém um único estado (corrente) e tentam melhorá-lo.
- Algoritmos de Melhoria Iterativa:
 - Pesquisa Subida da Colina (Hill-Climbing Search)
 - Arrefecimento Simulado (Simulated Annealing)
 - Pesquisa Tabu (Tabu Search)
 - Algoritmos Genéticos (Genetic Algorithms)
 - Ant Colony Optimization
 - Particle Swarm Optimization
- **Estratégia:** Começar como uma solução inicial do problema e fazer alterações de forma a melhorar a sua qualidade

Pesquisa Subida da Colina (Hill-Climbing Search)

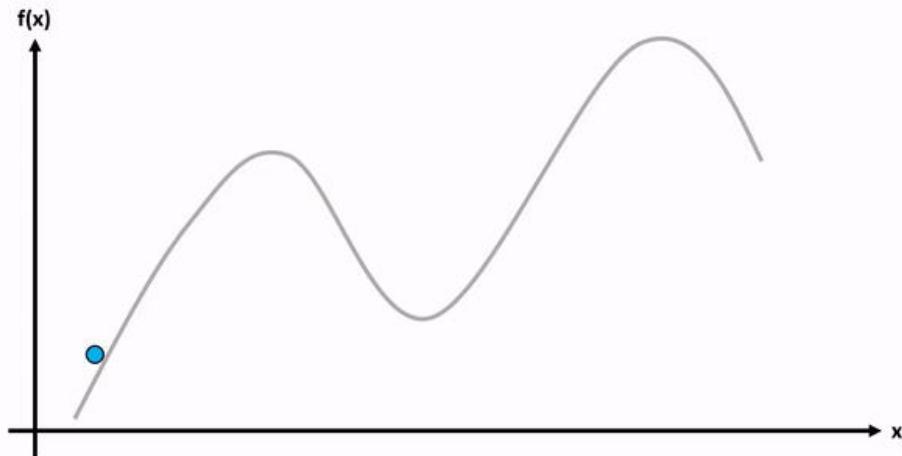
- **O Hill Climbing** é um algoritmo clássico, sendo bastante eficiente na tarefa de encontrar máximos ou mínimos locais, pela exploração.
- **Estratégia:**
 - iniciamos num ponto aleatório X e fazemos a sua avaliação.
 - movemos do nosso ponto X original para um novo ponto vizinho ao que estamos o Y.
 - Se esse novo ponto Y for uma solução melhor do que nosso ponto original X, ficamos nele e fazemos esse processo novamente, porém caso seja inferior, voltamos para nosso ponto inicial X e tentamos visitar um outro vizinho.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if VALUE[neighbor]  $\leq$  VALUE[current] then return STATE[current]
    current  $\leftarrow$  neighbor
```

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Pesquisa Subida da Colina (Hill-Climbing Search)

- Hill Climbing é ótimo em encontrar as boas soluções (mínimo/máximo locais) mas dificilmente vai encontrar a melhor solução (a menos que você tenha sorte na inicialização do ponto inicial).



Fonte: <https://www.globalsoftwaresupport.com/wp-content/uploads/2018/04/ezgif.com-video-to-gif-47.gif>

Pesquisa Subida da Colina (Hill-Climbing Search)



- Nos casos apresentados, o algoritmo chega a um ponto de onde não tem mais progresso.
- **Solução:** *reinício aleatório (random restart)*
 - O algoritmo realiza uma série de procura a partir de estados iniciais gerados aleatoriamente.
- Cada procura é executada
 - Até que um número máximo estipulado de iterações seja alcançado, ou
 - Até que os resultados encontrados não apresentem uma melhoria significativa.
- O algoritmo escolhe o melhor resultado obtido com as diferentes procura.

- O sucesso depende muito do morfologia (formato) da superfície do espaço de estados:
 - Se há poucos máximos locais, o reinício aleatório encontra uma boa solução rapidamente
 - Caso contrário, o custo de tempo é exponencial.

Arrefecimento Simulado (Simulated Annealing)

- O **Simulated Annealing** é um dos algoritmos inspirados pela natureza, assim como as redes neurais artificiais e os algoritmos genéticos.
- **Estratégia:**
 - De forma parecida com o Hill Climbing iniciamos de um ponto aleatório X e fazemos sua avaliação
 - o algoritmo faz um movimento até um dos seus vizinhos Y e avalia esse novo ponto.
 - Se os resultados melhoraram no nosso ponto Y então nos movemos até ele e refazemos o processo anterior, todavia se o ponto Y for inferior nós só iremos nos mover para ele caso nossa **probabilidade** de ir para um ponto negativo seja superior a um número aleatório.

```

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to "temperature"
  local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  schedule[t]
    if T = 0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  VALUE[next] - VALUE[current]
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 

```

$$\text{probabilidade}(p) = \text{Exp}(Y - X / T)$$

Essa função exponencial calcula a diferença do nosso ponto Y subtraída pelo ponto X em que estávamos anteriormente, dividido pela temperatura (variável T).

Com isso, nas primeiras iterações, quando nossa temperatura T está mais elevada, nós temos uma probabilidade maior de aceitar valores negativos e conforme são decorridas as iterações essa probabilidade diminui.

Com o tempo (diminuição da temperatura), este algoritmo passa a funcionar como Subida de Colinas

- A Tabu Search é uma Meta-heurística e um procedimento adaptativo auxiliar, que guia um algoritmo de pesquisa local na exploração contínua dentro de um espaço de pesquisa.
- A ideia básica da Pesquisa Tabu é penalizar movimentos que levam a solução para espaços de pesquisa visitados anteriormente (também conhecidos como tabu). A Pesquisa Tabu, no entanto, aceita de forma determinística soluções que não melhoraram para evitar ficar preso em mínimos locais.
- **Estratégia:**
 - Ideia chave: manter a sequência de nós já visitados (Lista tabu)
 - Partindo de uma solução inicial, a pesquisa move-se, a cada iteração, para a melhor solução na vizinhança, não aceitando movimentos que levem a soluções já visitadas, esses movimentos conhecidos ficam armazenados numa lista tabu.
 - A lista permanece na memória guardando soluções já visitadas (tabu) durante um determinado espaço de tempo ou certo número de iterações (prazo tabu). Como resultado final é esperado que se encontre um ótimo global ou próximo do ótimo global.

Algorithm 1 Tabu search algorithm

```
Set  $x = x_0$ ;                                ▷ Initial candidate solution
Set  $\text{length}(L) = z$ ;                        ▷ Maximum tabu list length
Set  $L = \{\}$ ;                                ▷ Initialize the tabu list
repeat
    Generate a random neighbor  $x'$ ;
    if  $x' \notin L$  then
        if  $\text{length}(L) > z$  then
            Remove oldest solution from L;          ▷ First in first out queue
            Set  $x' \in L$ ;
        end if
    end if
    if  $x' < x$  then
         $x = x'$ ;
    end if
until (Stopping criteria satisfied)           ▷ e.g. Number of iterations
return  $x$ ;                                    ▷ Best found solution
```

Population-based metaheuristic

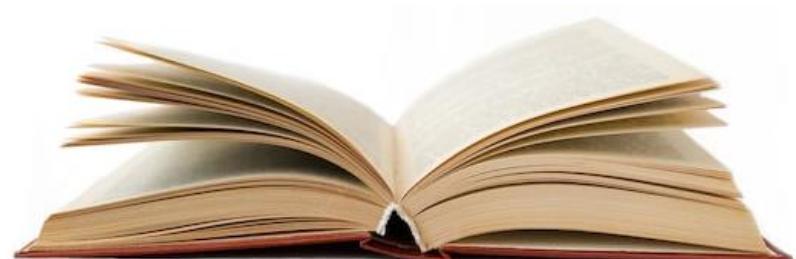
- Algoritmos Genéticos (Genetic Algorithms)
- Ant Colony Optimization
- Particle Swarm Optimization
- ...

Bibliografia Recomendada

- Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594.
- E.Costa, A.Simões (2008), Inteligência Artificial-Fundamentos e Aplicações, FCA, ISBN: 978-972-722-340-4, 2008.

Outro material

- Svetlana Lazebnik, Lecture notes Fall 2017 Artificial Intelligence, University of Illinois.
- Luís Paulo Reis, Lecture notes Artificial Intelligence (2019), Universidade do Porto



Universidade do Minho

Escola de Engenharia

Departamento de Informática

MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial