Anotações - teoria #01

13 de dezembro de 2023 17:01

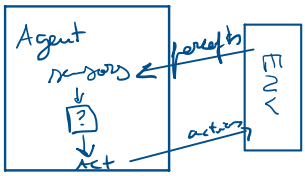
AI → study of intelligence as computing

"(Artificial Intelligence is) making a machine behave in ways that would be called intelligent if a human were so behaving."

John McCarthy.

Weak AI → systems designed and trained to solve very specific tasks
↳ chess AI / agent
→ learn to walk

Strong AI ou general AI
→ expected to learn and evolve over time



perceção → perceções do agente num dado instante

sequência de perceções → histórico total de

comportamento abstrato: função do agente

Programa ⇒ materialização da função do agente

Formulação de problemas
→ "problem solving agents"
↳ procura encontrar a seq. ação que leva a um estado desejável

- Muitos do problemas em ciências da computação podem ser formulados como:
  - Um conjunto S de ESTADOS (possivelmente infinito)
  - Um estado INICIAL s ∈ S
  - Uma relação de TRANSIÇÃO T ao longo deste espaço de estados
  - Um conjunto de estados FINAIS (objetivos): O ∈ S

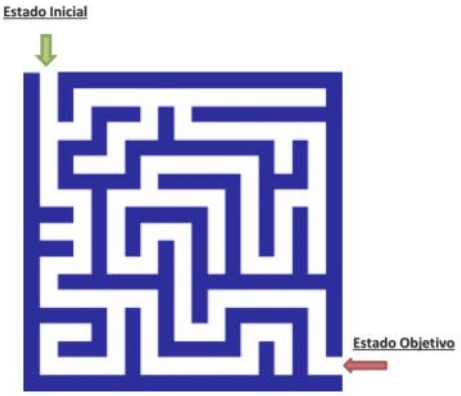
- Um problema pode ser definido formalmente em cinco componentes:
  1. Representação do Estado
  2. Estado Inicial (Atual)
  3. Estado Objetivo (define os estados desejados)
  4. Operadores (Nome, Pré-Condições, Efeitos, Custo)
  5. Custo da Solução

- Quais as ações possíveis?
- Quais os estados possíveis? (como representá-los?)
- Como avaliar os Estados

Synthetic Intelligence Lab

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



- Ambiente determinístico, totalmente observável → **problema do estado único** → Problema do aspirador
  - O agente "**sabe**" exatamente 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 contingência**
  - Perceções fornecem novas informações sobre o estado atual
  - Frequentemente intercalam procura e execução
- Espaço de estados desconhecido → **problema de exploração**

Estado ≠ Nó

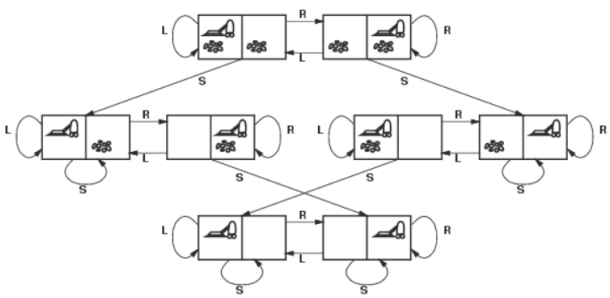
NODE depth = 3 g = 3

Exemplo: o problema do aspirador

(problema de 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



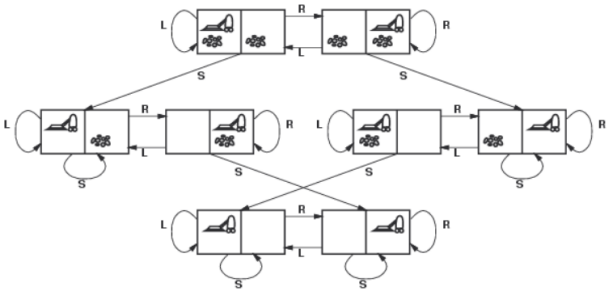
- Custo da Solução: Cada ação custa 1 (custo total =

Exemplo: o problema do aspirador

(problema de 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)



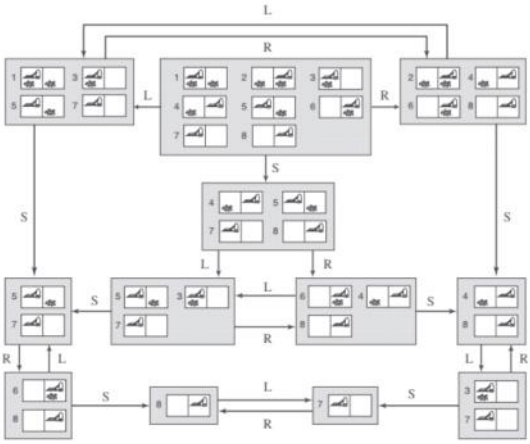
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador

(problema de estados 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



Single-state

Start in: 5

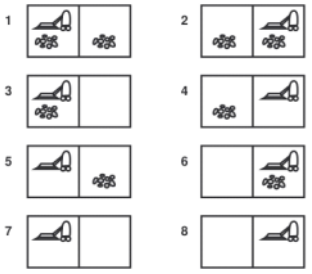
Solution: [*right*, *suck*]

Multiple-state

Start in: $\{1,2,3,4,5,6,7,8\}$

Solution: [*right*, *suck*, *left*, *suck*]

right $\rightarrow \{2,4,6,8\}$
suck $\rightarrow \{4,8\}$
left $\rightarrow \{3,7\}$
suck $\rightarrow \{7\}$



Single-state problem
- observable (at least the initial state)
- deterministic
- static
- discrete

Multiple-state problem
- partially observable (initial state not observable)
- deterministic
- static
- discrete

Contingency problem
- partially observable (initial state not observable)
- non-deterministic

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.

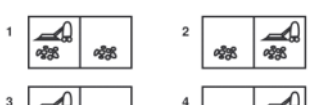
Portanto, a sequência de ações deve ser planeada mediante cada perceção do ambiente, ou seja, por cada ação realizada o agente recolhe informação através do seu sensor e só depois decide a próxima ação a executar.

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]

Contingency

Murphy's Law:



Contingency

Murphy's Law:
suck **can dirty a clean carpet**

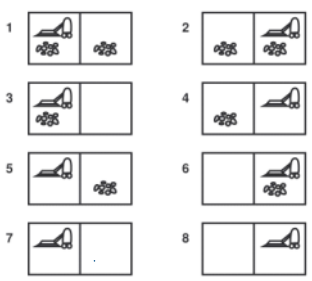
Local sensing:
dirty*/*not dirty **at location only**

Start in: $\{1,3\}$

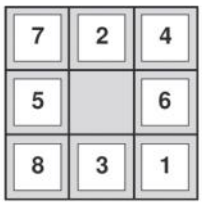
Solution: [***suck***, ***right***, ***suck***]

suck $\rightarrow \{5,7\}$
right $\rightarrow \{6,8\}$
suck $\rightarrow \{6,8\}$

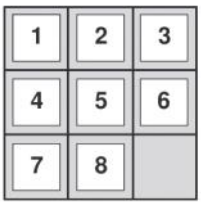
Improvement: [***suck***, ***right***, if ***dirt*** then ***suck***]
(decide whether in 6 **or** 8 **using local sensing)**



Example: The 8-puzzle



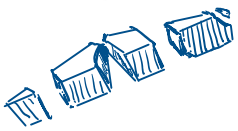
Start State



Goal State

States	integer locations of tiles
Actions	*left*, *right*, *up*, *down*
Goal test	= goal state?
Path cost	1 per move

- 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



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.

Qualidade da solução:

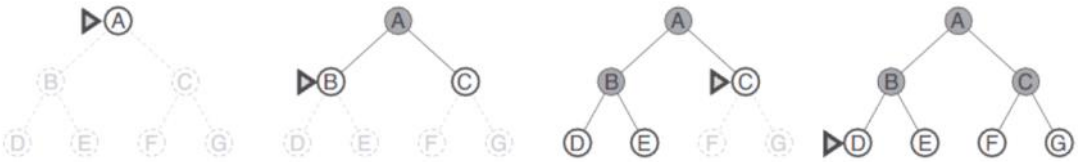
- Uma **solução satisfatória** se é uma qualquer solução;
- uma **solução semi-óptima** é aquela que tem aproximadamente o menor custo entre todas as soluções;
- uma **solução óptima** é aquela que tem o menor custo entre todas as soluções.

Estratégia de procura

- A estratégia: a ordem da expansão do nó
- Critérios de avaliação:
  - Completude: Está garantido que encontra a solução?
  - Otimização: Encontra a melhor solução?
  - Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
  - Complexidade no Espaço: Quanta memória necessita para fazer a procura?
- 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 procura
  - d: a profundidade da melhor solução
  - m: a máxima profundidade do espaço de estados

53

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



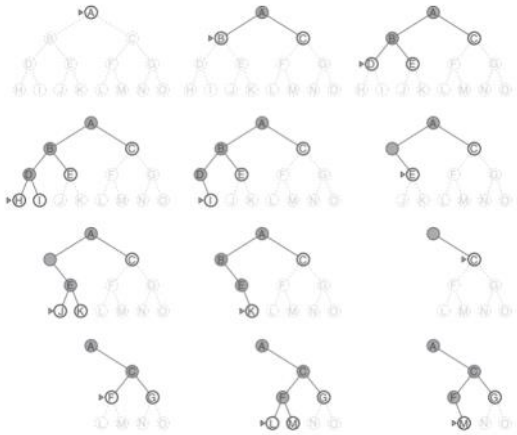
Procura Primeiro em Profundidade (Depth-First Search)

- 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 em 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
  - Ótima: Não (em princípio devolve a 1ª solução que encontra)

- b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
- 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 Procura com Profundidade Limitada



Problema de Custo Uniforme

- Estratégia:
  - Para cada nó da lista de estados não expandidos, guardar 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

- Estratégia:
  - Para cada nó da lista de estados não expandidos, guardar 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)
- Procura Primeiro em Largura é igual a Procura de Custo Uniforme se g(N)=Depth(N)
- Equivalente a **Procura Primeiro em Largura (Breadth-first search)**, se os custos forem todos iguais
- Implementação: lista de estados não expandidos é uma lista prioritária ordenada pelo custo do caminho
- Temos de garantir que g(sucessor)>= 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.

O algoritmo de Dijkstra (Edsger Dijkstra, 1956), resolve o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso não negativo.

Procura Iterativa

Procura Iterativa
Aprofundamento Progressivo

Se não conhecermos 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 principio da procura limitada fazendo variar esse limite entre zero e infinito.

Usar **Procura Primeiro em Profundidade (Depth-First Search)** como uma sub-rotina

- Verificar a raiz
- Desenvolver um DFS procurando um caminho de comprimento 1
- Se não houver um caminho de comprimento 1, desenvolver um DFS procurando um caminho de comprimento 2
- Se não houver um caminho de comprimento 2, desenvolver um DFS procurando um caminho de comprimento 3...

Procura Bidirecional:

- Estratégia: Executar uma procura 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 procuras? Que tipo de procura fazer nas duas metades?
- Eg., Para encontrar uma rota na Romênia, existe apenas um estado objetivo, portanto, a procura para trás é muito parecida com a procura para a frente; Mas se o objetivo é uma descrição abstrata, como o de que "nenhuma rainha ataca outra rainha" no problema das rainhas n, a procura bidirecional é de difícil uso.

Comparação entre Estratégias de Procura

- Avaliação das estratégias de procura:
  - B é o fator de ramificação
  - d é a profundidade da solução
  - m é a máxima profundidade da árvore
  - l é a profundidade limite da procura

falha em espaços de profundidade infinita, com repetições (loops)

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

↳ primeira que encontra

Procura Informada

↳ utiliza informação sobre o problema

função de avaliação

greedy : $f(n) = h(n)$

A* : $f(n) + g(n) + h(n)$

Greedy → Não é completa : pode entrar em ciclo

$O(b^m)$ ← espaço e tempo

A* → ótimo e completo

complexidade no tempo é exponencial

(em erro relativo de h^* comprimento da solução)

Heurísticos - 8 puzzle

7	2	4
5		6
8	3	1

$H_1(n)$ = Nº de peças fora do lugar

$H_2(n)$ = Soma das distâncias das peças até às suas posições corretas

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

Gulosa	Não	Não	no pior caso : $O(b^m)$ No melhor caso: O(bd)
A*	Sim	Sim (se a heurística for admissível)	Nº de nodos com $g(n)+h(n) \leq C^*$

↳ should never overestimate

e consistente

$\forall$ node n e sucessor n', com custo c

$h(n) \leq h(n') + c$