Inteligência Artificial - Notas de aula

Raoni F. S. Teixeira

Aula 2 - Busca Exaustiva

1 Introdução

Este curso trata do projeto de agentes que interagem com ambientes. Em ambientes controlados, os agentes podem ser mais simples. Nesta aula, abordamos um ambiente tão controlado que os sensores podem ser ignorados, permitindo projetar o agente com busca exaustiva.

2 Busca exaustiva

O ambiente possui estas características:

- *Observável*: o agente conhece seu estado atual.
- *Discreto*: há ações finitas em cada estado.
- *Conhecido*: o estado resultante de cada ação é previsível.
- *Determinístico*: cada ação gera um único resultado.

Com essas características, o agente não precisa de sensores, pois sabe o estado atual e o que ocorrerá após cada ação. Podemos modelar esse ambiente como um grafo, onde os vértices representam estados e as arestas, ações possíveis.

Por exemplo, no grafo da Figura 1, o vértice s representa o estado atual. As arestas a_1, a_2 e a_3 são as ações possíveis, que levam o agente a novos estados b, d e e. O objetivo do agente é encontrar o caminho de menor custo entre o estado inicial s e o objetivo o.

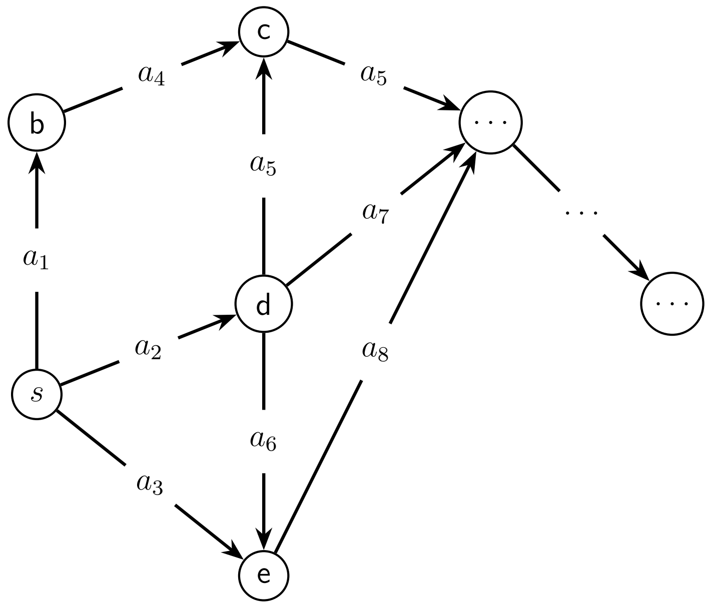


Figura 1: Ambiente visto como um grafo cujos vértices e arestas correspondem, respectivamente, aos estados do ambiente e às ações do agente. Apenas as ações do agente afetam o ambiente.

A solução para esse problema é o caminho de menor custo $\delta(s, o)$, definido por:

$$\delta(s,o) = \begin{cases} \min\big(\sum_{i=1}^{k} w(a_i) : \forall \text{ caminho } s \rightsquigarrow o \big), & \text{se existe caminho } s \rightsquigarrow o \\ \infty, & \text{caso contrário,} \end{cases} \tag{1}$$

em que a_i representa uma ação do caminho, e w é a função de custo associada a cada ação.

Muitos problemas reais podem ser modelados como grafos. Um exemplo é o quebra-cabeça de oito peças apresentado na Figura 2. Esse quebra-cabeça consiste em uma matriz 3×3 com 8 (oito) células quadradas rotuladas de 1 a 8 e uma célula vazia. O objetivo é colocar as células em ordem. É permitido mover as células horizontalmente ou verticalmente mas não diagonalmente. A resposta procurada é uma sequência de movimentos a ser executada para encontrar um estado objetivo. No caso apresentado na Figura 2, a sequência de resposta é 5 e 6.

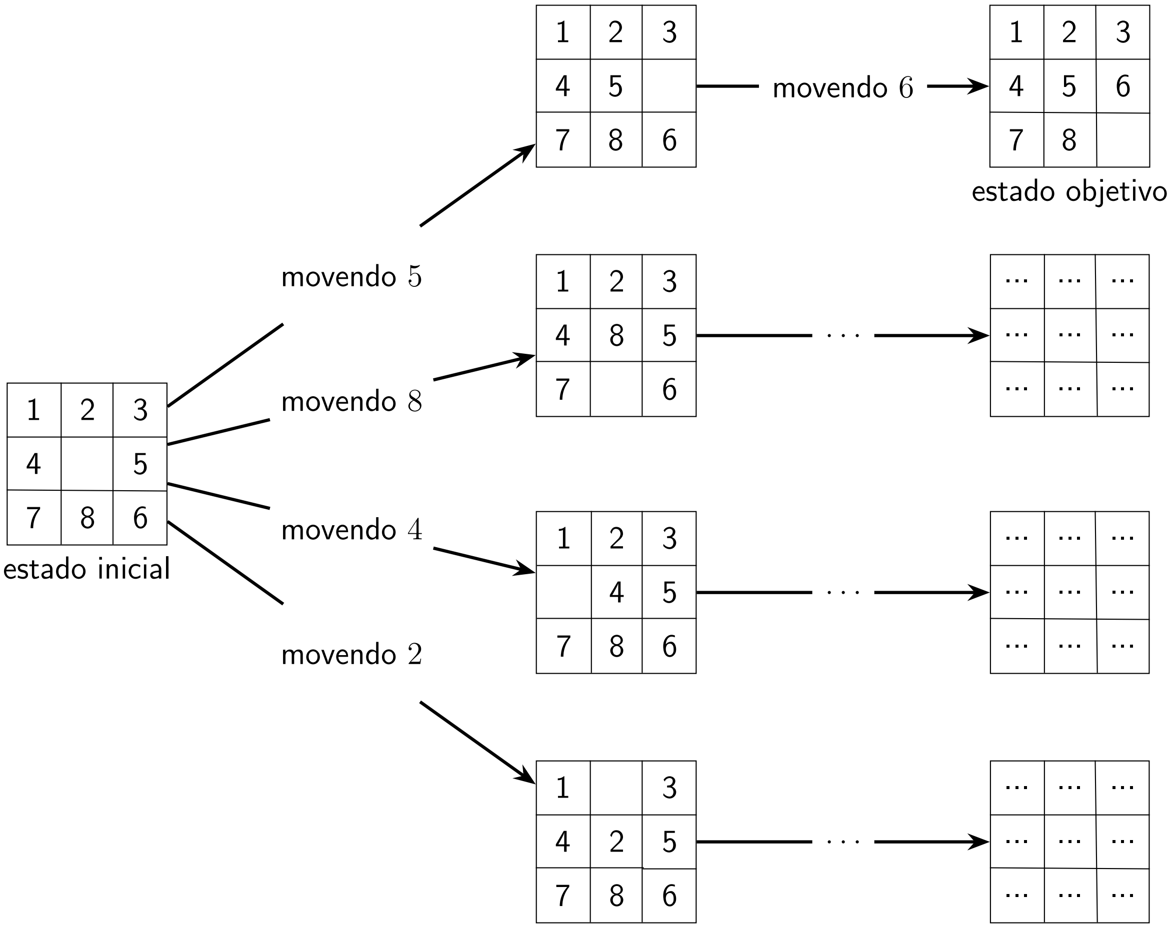


Figura 2: Grafo do quebra-cabeça de 8 peças.

Dado um grafo $G = (\mathcal{S}, E)$, um estado inicial s, um conjunto de estados objetivo $\mathcal{O}$, $\mathcal{O} \subset \mathcal{S}$, e uma função w, $w(a_1, a_2, \ldots, a_n) \mapsto [-\infty, \infty]$, que calcula o custo de uma sequência de ações, um algoritmo de busca exaustiva ou de caminho mínimo encontra uma solução para o problema. Esse tipo de algoritmo segue uma sequência de ações, descobrindo novos estados até ele visite todos os estados ou encontre uma solução. O algoritmo de busca informada $\mathrm{A}^\star$ é apresentado a seguir.

3 $\mathrm{A}^\star$

O algoritmo $\mathrm{A}^\star$ foi proposto por Hart, Nilsson e Raphael em 1968 [HNR68] como uma extensão do algoritmo de Dijkstra [CLRS09, Dij59]. Essa extensão incorpora uma heurística h que *prevê* (estimativa ou intuição) a distância de um vértice para o objetivo. Seu uso em IA remonta a trabalhos pioneiros como os de [NSS59] sobre resolução heurística de problemas.

O algoritmo funciona explorando primeiro os estados que aparentam estar mais próximos do objetivo. Em cada iteração, ele descobre novos estados e os armazena em uma fila de prioridades. A prioridade de um estado v, $c(v)$, é a soma do custo acumulado até v, $d(v)$, e da estimativa do custo restante até o objetivo, $h(v)$:

$$c(v) = \underbrace{d(v)}_{\text{custo acumulado até } v} + \underbrace{h(v)}_{\text{estimativa do custo restante até o objetivo}} .$$

Etapas do algoritmo:

1. **Inicialização:** Adicione o estado inicial s à fila de prioridade Q com custo zero.
2. **Defina os valores iniciais:** Defina o custo acumulado para o estado inicial como $d[s] = 0$.
3. **Marque o início da busca:** Defina o predecessor de s como nulo ($\pi[s] \leftarrow \perp$).
4. **Explore a fila:** Enquanto a fila não estiver vazia, retire o estado v com menor prioridade. Esse será o próximo estado a ser explorado.
5. **Verifique se o objetivo foi alcançado:** Se o estado atual for o objetivo, encerre a busca.
6. **Atualize os vizinhos** Calcule o custo acumulado e atualize o predecessor de cada vizinho.
7. **Atualize a fila de prioridade:** Insira cada estado vizinho u na fila Q com prioridade $c(u) = d[u] + h(u)$. A prioridade considera o custo acumulado $d[u]$ e a estimativa do custo restante $h(u)$.

O algoritmo continua até encontrar o estado objetivo ou visitar todos os estados. Para reconstruir o caminho, o vetor de predecessores π é percorrido recursivamente, a partir do objetivo até o estado inicial. O pseudo-código dos algoritmos $\mathrm{A}^\star$ e $\textsc{Reconstrução}$ são apresentados a seguir.

$\text{A}^\star(s, h, o)$

$Q \leftarrow \emptyset$ //Inicializa a fila de prioridade
$\text{INSERT}(Q, s, 0)$ //Adiciona o estado inicial com custo zero
$d[s] \leftarrow 0$ //Define o custo acumulado do estado inicial
$\pi[s] \leftarrow \perp$ //Predecessor inicial é nulo
while $Q \neq \emptyset$ **do**
    $v \leftarrow \text{EXTRACT-MIN}(Q)$ //Extrai o estado com menor prioridade
    if $v = o$ **then**
        return v, π //Se o objetivo for alcançado, retorna o resultado
    for each vertex $u \in Adj(v)$ **do**
        $d[u] \leftarrow d[v] + w(v, u)$ //Atualiza o custo acumulado do vizinho
        $\pi[u] \leftarrow v$ //Atualiza o predecessor do vizinho
        $\text{INSERT}(Q, u, d[u] + h(u))$ //Insere o vizinho na fila com prioridade

$\text{RECONSTRUÇÃO}(\pi, v)$

if $\pi[v] \neq \perp$ **then**
    $\text{RECONSTRUÇÃO}(\pi, \pi[v])$
print v

A operação $\text{INSERT}(Q, v, p)$ insere o vértice v na fila Q com prioridade p. A operação $\text{EXTRACT-MIN}(Q)$ retira de Q o vértice com menor prioridade. A função de adjacência, $Adj(v)$, devolve todos os estados alcançados com a execução de uma ação em v e $w(v, u)$ devolve o custo da ação (v, u). O símbolo $\perp$ indica a fim da lista (nulo).

A Figura 4 mostra a fila de prioridade Q durante a execução do algoritmo para o grafo da Figura 2. Inicialmente, a fila contém apenas o estado inicial. Em cada iteração, novos estados são descobertos e adicionados a Q. O algoritmo escolhe o estado com a menor prioridade para prosseguir, explorando primeiro os mais promissores (de acordo com a heurística), enquanto os demais permanecem em Q para futuras iterações. O processo continua até que o estado objetivo seja encontrado. Na Figura 4, o estado com menor prioridade na terceira fila Q é o estado objetivo.

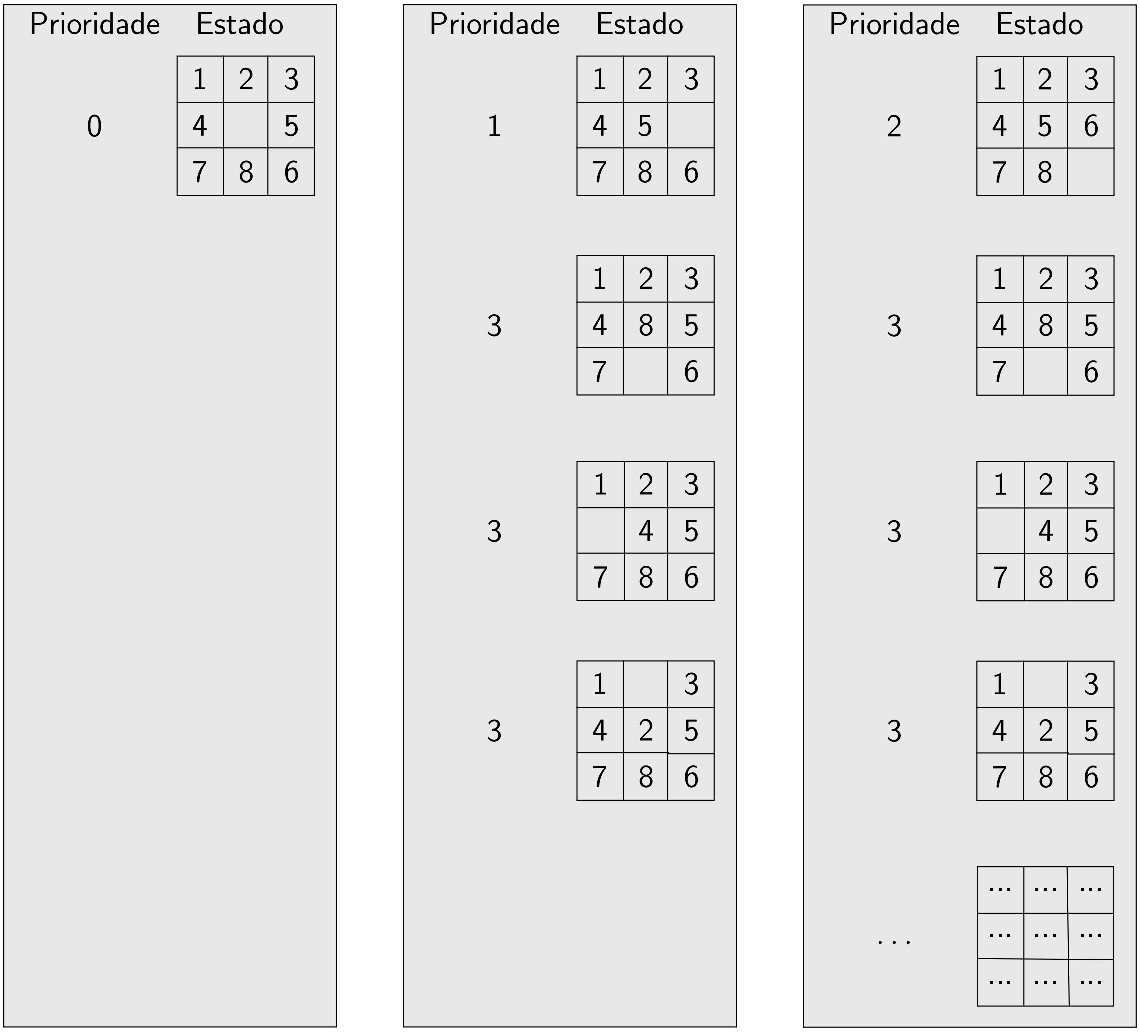
3.1 Propriedades de Heurísticas no A*

Para garantir que o algoritmo A* encontre o caminho de menor custo, a função heurística $h(n)$ deve ser **admissível**. Isso significa que ela nunca superestima o custo real até o estado objetivo. Formalmente:

$$h(n) \leq \delta(n, o) \quad \forall\ n \in S,$$

em que $\delta(n, o)$ é o custo real mínimo de n até o objetivo o.

No quebra-cabeça de 8 peças, duas heurísticas admissíveis comuns são:



Q na primeira iteração Q na segunda iteração Q na terceira iteração

Figura 4: Fila de Prioridade Q durante a execução do Algoritmo $A^{\star}$ para o grafo da Figura 2.

- **Número de peças fora do lugar**: Subestima o custo real, pois cada movimento pode corrigir apenas uma peça por vez.
- **Distância Manhattan**: Soma das distâncias horizontais e verticais de cada peça até sua posição correta. Como movimentos diagonais são proibidos, essa heurística é otimista.

Além disso, se $h(n)$ for **consistente** (ou monotônica), ou seja, satisfizer a desigualdade triangular:

$$h(n) \leq c(n, n') + h(n') \quad \forall\, n, n',$$

o A* será eficiente, expandindo cada estado apenas uma vez. Heurísticas consistentes são automaticamente admissíveis, mas o inverso nem sempre é verdade.

Se $h(n)$ não for admissível, o A* pode perder a garantia de otimalidade, retornando um caminho subótimo.

4 Exercícios

1. Calcule a heurística "número de células na posição errada" para os estados a, b e c da Figura 5.

8	1	3
4		2
7	6	5

a

4	1	3
8	2	5
7	6	

b

4	2	3
5	1	8
7		6

c

Figura 5: Exemplos de estados do quebra-cabeça de 8-peças.

2. Simule a execução do algoritmo $\mathrm{A}^\star$ para o grafo da Figura 2.

3. Determine o custo do caminho entre os estados s e v na Figura 6.

1	2	3
4	5	6
8	7	

s

$\longrightarrow \cdots \longrightarrow$

1	2	3
4	5	6
7	8	

v

Figura 6: Caminho $s \rightsquigarrow v$.

4. Mostre que a distância Manhattan é mais *informativa* que o número de peças fora do lugar. Isso é sempre verdade?

Referências

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. Received: 11 June 1959.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[NSS59] Allen Newell, John C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.