Parte II: IA Simbólica ou Good Old-Fashioned Artificial Intelligence (GOFAI).

Prof. Fabiano Araujo Soares, Dr. / FGA 0221 - Inteligência Artificial

Universidade de Brasília

2025

Resolvendo problemas por busca

Como agentes inteligentes podem tomar decisões autônomas em ambientes complexos sem conhecer previamente todas as soluções possíveis?

Resolvendo problemas por busca

Como agentes inteligentes podem tomar decisões autônomas em ambientes complexos sem conhecer previamente todas as soluções possíveis?

Algoritmos de busca são o motor da inteligência artificial clássica: eles transformam problemas aparentemente insolúveis em trajetórias concretas rumo à solução, pavimentando o caminho da tomada de decisões inteligentes.

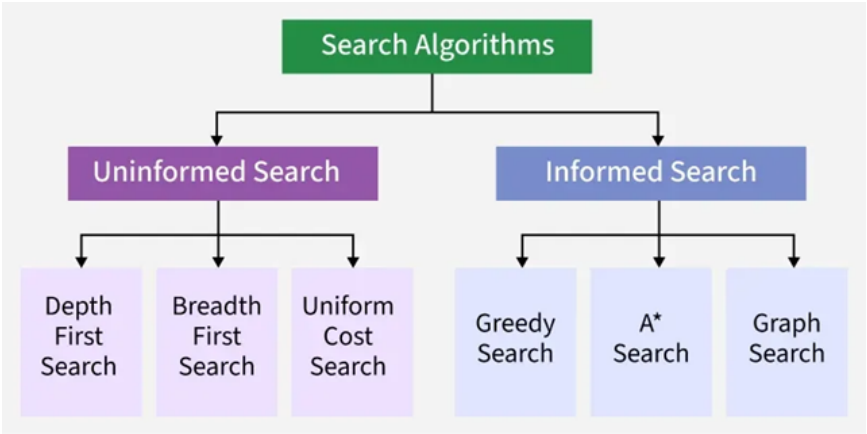
Motivação

- Buscas são fundamentais para agentes inteligentes baseados em objetivos.
- Resolução de problemas vista como navegação em um espaço de estados.

Formulação de Problemas de Busca

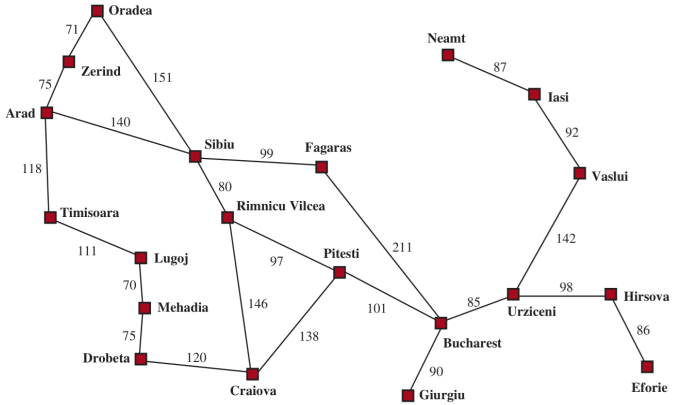
- **Estado Inicial:** Ponto de partida.
- **Ações:** Operadores disponíveis.
- **Objetivo:** Estado(s) desejado(s).
- **Custo do caminho:** Soma dos custos individuais das ações.

Busca cega (uninformed) ou busca informada (informed)?



Exemplo Clássico: Mapa da Romênia

- Problema: encontrar caminho mínimo entre cidades.
- Demonstração visual do espaço de estados e aplicação dos algoritmos.

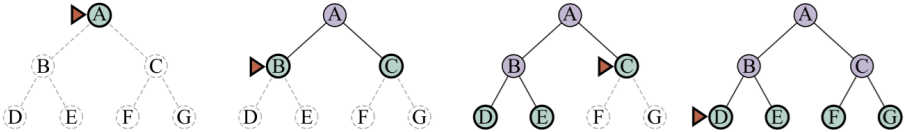


Busca Cega: Conceito

- Também chamada de **busca não informada**, pois não utiliza nenhuma informação além da definição do problema.
- Explora o espaço de estados apenas com base nas regras do problema, sem estimativas ou dicas sobre a localização da solução.
- Aplica-se a qualquer problema que possa ser formulado como um grafo ou árvore de estados.
- Métodos comuns incluem **Busca em Largura (BFS)**, **Busca em Profundidade (DFS)** e **Busca de Custo Uniforme**.
- Geralmente garantem completude (encontram uma solução se existir), mas podem ser ineficientes em termos de tempo e memória.
- São essenciais para entender conceitos básicos de exploração em IA antes de avançar para buscas mais eficientes com heurísticas.

Busca em Largura (Breadth-First Search)

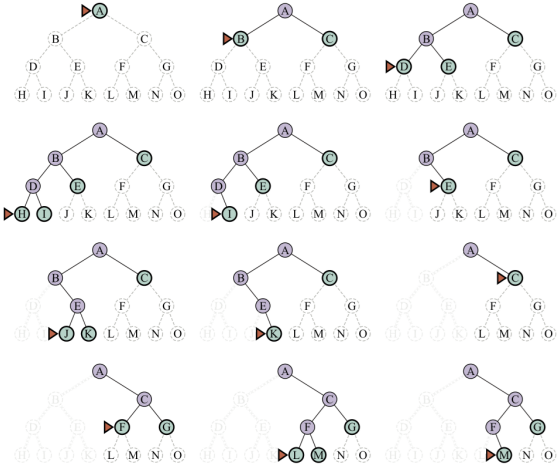
- Explora todos os nós em cada nível antes de aprofundar.
- Garante otimalidade se custos são iguais.
- Requer muita memória.



Busca em Profundidade (Depth-First Search)

- Explora sempre o filho mais profundo primeiro.
- Baixo consumo de memória.
- Pode ficar presa em caminhos sem solução.
- Não garante otimalidade.

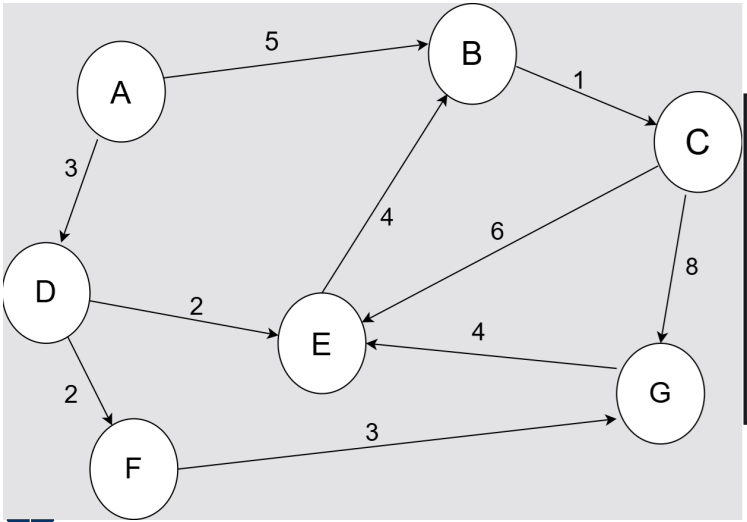
Busca em Profundidade (Depth-First Search)



Busca de Custo Uniforme (Uniform Cost Search)

- Sempre expande o nó de menor custo acumulado.
- Garante otimalidade se todos os custos forem positivos.
- Mais eficiente que BFS se os custos variam.

Busca de Custo Uniforme (Uniform Cost Search)



	Frontier List	Expand List	Explored List
1.	{(A,0)}	A	NULL
2.	{(A-**D**, 3), (A-B, 5)}	D	{A}
3.	{(A-**B**, 5), (A-D-E, 5), (A-D-F, 5)}	B	{A, D}
4.	{(A-D-**E**, 5), (A-D-F, 5), (A-B-C, 6)}	E	{A, D, B}
5.	{(A-D-**F**, 5), (A-B-C, 6), ~~(A-D-E-B, 9)~~} *here **B** is **already explored**	F	{A, D, B, E}
6.	{(A-B-**C**, 6), (A-D-F-G,8)}	C	{A, D, B, E, F}
7.	{(A-D-F-**G**,8), ~~(A-B-C-E,12)~~, (A-B-C-G, 14)} *here **E** is **already explored**	G	{A, D, B, E, F, C}
8.	{(A-D-F-G,8)}	NULL	{A, D, B, E, F, C, **G**} # GOAL Found!

Busca Informada: Conceito

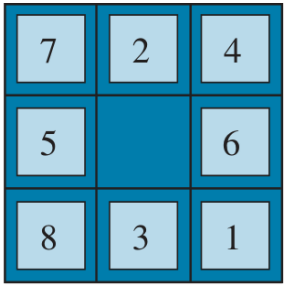
- A busca informada utiliza uma função heurística $h(n)$ que estima o custo restante do nó atual n até o objetivo.
- Essa heurística agrega conhecimento específico do problema para guiar a busca, tornando o processo mais eficiente.
- Ao contrário da busca cega, que explora estados sem direcionamento, a busca informada prioriza os caminhos que parecem mais promissores com base nessa estimativa.
- O resultado é a redução do tempo e do espaço necessários para encontrar uma solução, especialmente em grandes espaços de estados.
- Exemplos de algorítimos de busca informada: **Busca Gananciosa (Greedy Search)**, que usa só a heurística $h(n)$, e **Busca A***, que combina o custo do caminho já percorrido $g(n)$ com $h(n)$ para uma decisão balanceada.
- Para garantir soluções ótimas, a heurística deve ser admissível (nunca superestimar o custo real) e idealmente consistente (monótona).

Funções Heurísticas: Conceito

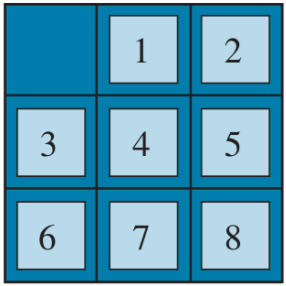
- Heurísticas são funções que trazem informação sobre o custo para alcançar o objetivo a partir de um estado.
- Elas guiam algoritmos de busca informada, permitindo explorar o espaço de estados de forma eficiente.
- Para problemas complexos, uma boa heurística pode reduzir drasticamente o tempo de busca.

Exemplo: Quebra-cabeça deslizante

- Peças movem-se horizontal ou verticalmente.
- Com 8 posições, temos $9! = 362.880$ estados alcançáveis.
- Com 15 posições, o número de estados cresce exponencialmente.
- Importante utilizar uma **heurística admissível** para garantir otimalidade.



Start State



Goal State

Heurísticas comuns para o quebra-cabeça

- h_1: Número de peças fora do lugar.
  - Exemplo: $h_1 = 8$.
  - Heurística admissível: Nunca superestima o custo real, *e.g.* cada peça fora do lugar requer ao menos um movimento.
- h_2: Soma das distâncias (Manhattan Distance) das peças até sua posição final.
  - Exemplo: $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$.
  - Também admissível e mais informativa.

Exemplo: Custo da solução

- Custo real da solução para o exemplo mostrado: 26.
- Ambas as heurísticas h_1 e h_2 são menores ou iguais ao custo real, garantindo admissão.

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Busca Gananciosa (Greedy Search)

- **Usa apenas a função heurística** $h(n)$: A Greedy Search prioriza expandir o nó que parece estar mais próximo do objetivo, baseando-se unicamente na estimativa heurística $h(n)$.
- **Decisões localmente ótimas**: Em cada passo, ela faz a escolha que parece ser a melhor no momento, sem considerar as consequências futuras dessa escolha ou o custo do caminho já percorrido.
- **Rápida, mas não garante otimalidade**: É um algoritmo rápido pois foca em avançar rapidamente para o objetivo, mas pode levar a um caminho subótimo ou até mesmo a um beco sem saída, já que não explora alternativas que poderiam ser melhores a longo prazo.

Exemplo Simples: Mapa da Romênia (de Arad a Bucareste)

- **Custo Heurístico $h(n)$ (distância em linha reta até Bucareste):**
  - Arad: 366
  - Sibiu: 253
  - Fagaras: 178
  - Oradea: 380
- Partindo de Arad ($h(Arad) = 366$):
  - Vizinhos: Sibiu ($h(Sibiu) = 253$), Timisoara ($h(Timisoara) = 329$), Zerind ($h(Zerind) = 374$).
  - Greedy escolhe **Sibiu** ($h = 253$) pois é o menor valor heurístico.
- De Sibiu ($h(Sibiu) = 253$):
  - Vizinhos: Arad ($h = 366$), Fagaras ($h = 178$), Oradea ($h = 380$), Rimnicu Vilcea ($h = 193$).
  - Greedy escolhe **Fagaras** ($h = 178$).
- De Fagaras ($h(Fagaras) = 178$):
  - Vizinhos: Sibiu ($h = 253$), **Bucareste** ($h = 0$).
  - Greedy escolhe **Bucareste** ($h = 0$), alcançando o objetivo.

Exemplo Simples: Mapa da Romênia (de Arad a Bucareste) [cont.]

- O caminho encontrado é Arad $\rightarrow$ Sibiu $\rightarrow$ Fagaras $\rightarrow$ Bucareste.
- **Problema:** Pode não ser o caminho mais curto em termos de custo real (distância percorrida), pois a Greedy ignora o custo real acumulado $g(n)$.

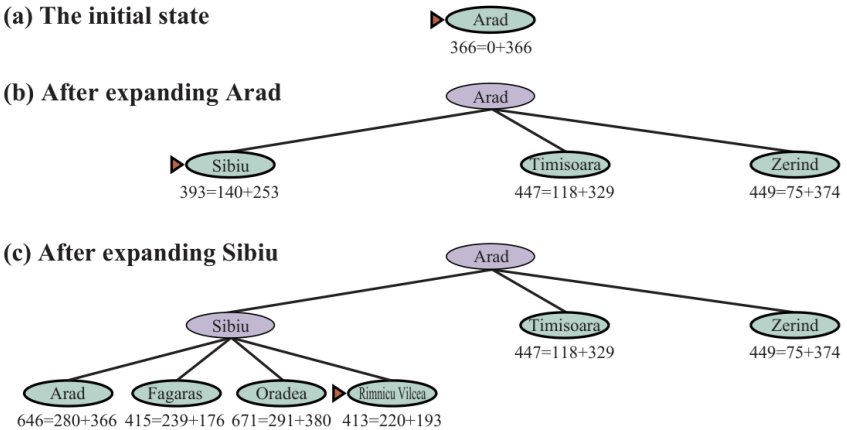
Busca A*

- Utiliza a função de avaliação $f(n) = g(n) + h(n)$, onde:
  - $g(n)$: custo do caminho desde o nó inicial até o nó n.
  - $h(n)$: estimativa heurística do custo do nó n até o objetivo.
- Combina o melhor de duas abordagens: custo real acumulado e estimativa heurística.
- É completa e ótima se a heurística $h(n)$ for admissível (nunca superestimar o custo real).
- Amplamente utilizada em aplicações práticas, como rotas de GPS e jogos.

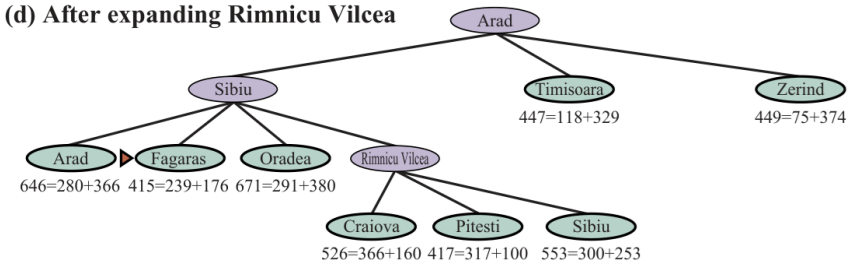
Exemplo simplificado no Mapa da Romênia

- Desejamos ir de Arad a Bucareste.
- $h(n)$: distância em linha reta (heurística) até Bucareste.
- $g(n)$: distância real percorrida até o nó n.
- Passos do algoritmo:
  - Começa em Arad: $f(Arad) = g(Arad) + h(Arad) = 0 + 366 = 366$.
  - Expande vizinhos de Arad, calculando $f(n)$ para cada um e escolhe o nó com menor $f(n)$.
  - Continua expandindo nós, atualizando custos e escolhendo sempre o nó com menor $f(n)$.
  - Chega em Bucareste pelo caminho de menor custo total.
- Caminho típico encontrado: Arad $\rightarrow$ Sibiu $\rightarrow$ Fagaras $\rightarrow$ Bucareste.

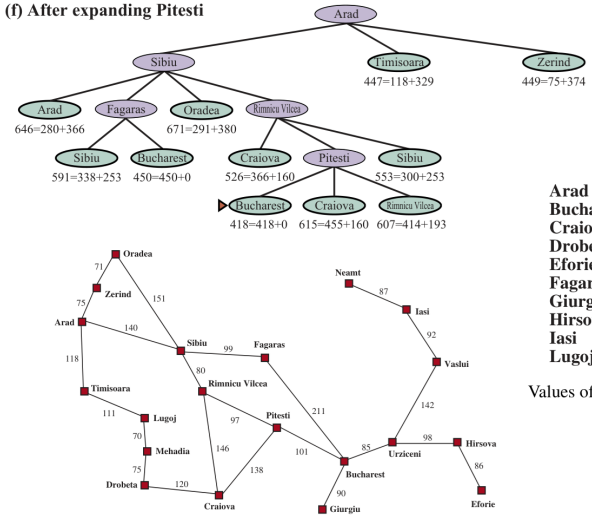
Exemlo: Busca A*



Exemlo: Busca A*



Exemlo: Busca A*



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

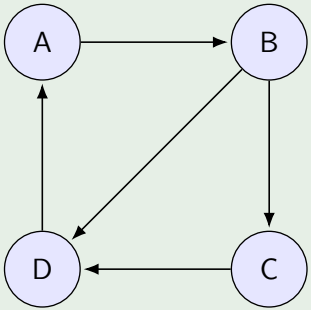
Values of h_{SLD}—straight-line distances to Bucharest.

Graph Search

- Variante dos algoritmos de busca que evita a expansão redundante de estados já visitados, armazenando os nós explorados.
- Essencial para problemas com muitos ciclos ou grandes espaços de estados, onde visitar um mesmo nó várias vezes seria ineficiente.
- Mantém uma **lista de nós visitados** para não revisitar estados já processados.

Exemplo

Considere um grafo com ciclos simples:



Ao explorar a partir do nó A:

- O algoritmo adiciona A à lista de visitados e expande B.
- Ao chegar em D, sem a lista de visitados, o algoritmo poderia voltar para A criando um ciclo infinito.
- Com Graph Search, A já está na lista, então D não expande para A novamente.
- Isso evita loops infinitos e redundância na busca.

Comparação

Algoritmo	Completo	Ótimo	Complexidade Tempo	Complexidade Espaço
BFS	Sim	Sim*	Alta	Alta
DFS	Não	Não	Baixa	Baixa
Uniform Cost	Sim	Sim	Variável	Alta
Greedy	Não	Não	Variável	Variável
A*	Sim	Sim**	Variável	Variável

*Se todos os custos forem iguais
**Se heurística for admissível

Conclusão

- Algoritmos de busca são fundamentais em IA.
- Busca informada supera limitações das buscas cegas pela eficiência.
- Estudo de heurísticas e implementação prática são importantes.

Referências

1. Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. 4ª ed.
2. Materiais da disciplina e aulas complementares.

Obrigado!

E-mail: fabianosoares@unb.br

LinkedIn: https://www.linkedin.com/in/fabiano-soares-06b6a821a/

Site do curso: https://www.fabianosoares.eng.br/fga0221-inteligência-artificial