Roteiro de Aula 2 – Busca (Cap. 3.1–3.5, Poole & Mackworth, AIFCA 3e)

Disciplina: BCC740 – Inteligência Artificial

4 de outubro de 2025

Sumário da Aula

1 Problem Solving as Search

Ideia central

Resolver um problema como **busca** em um espaço de estados: encontrar uma sequência de ações que transforma o estado inicial em um estado meta, minimizando (opcionalmente) um custo.

Formulação

Um problema de busca é uma quíntupla

$$\langle S, A, \gamma, c, (s_0, G) \rangle$$

onde:

- S: conjunto (possivelmente grande) de estados.
- $A(s)$: conjunto de ações aplicáveis em $s \in S$.
- $\gamma(s, a)$: função de transição (ou $T(s, a) \to s'$).
- $c(s, a, s') \geq 0$: custo do passo; custo de caminho $g(n)$ é a soma acumulada.
- $s_0 \in S$: estado inicial; $G \subseteq S$: conjunto de metas (teste de objetivo).

Solução Uma **solução** é uma sequência de ações $(a_1, \ldots, a_k)$ tal que $\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2)\ldots, a_k) \in G$. Quando há custo, buscamos solução de **custo mínimo**.

2 State Spaces

Representação de estados e ações

- **Estados**: escolhas de representação têm impacto em eficiência (ex.: tuplas imutáveis para puzzles).
- **Ações**: operadores locais; podem depender do estado (pré-condições).

Exemplos de modelagem

8-Puzzle

- S: permutações da grade 3×3 com um espaço vazio.
- A: mover o vazio $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ quando possível.
- Meta: estado ordenado (1..8; vazio).

Missionários e Canibais

- Estado: (M_E, C_E, B) com $B \in \{\text{esq}, \text{dir}\}$; restrição $M \geq C$ em cada margem quando $M > 0$.
- Ações: transportar $(\Delta M, \Delta C)$ com $1 \leq \Delta M + \Delta C \leq 2$ respeitando restrições.

3 Graph Searching

Tree search vs Graph search

- **Tree search**: ignora estados repetidos; pode reexpandir o mesmo estado inúmeras vezes.
- **Graph search**: mantém *explored set* (fechados) e/ou *visited* para evitar repetições e ciclos.

Estados repetidos Sem controle de repetição, complexidade explode. Em ambientes com *reversibility* (desfazer ação) ciclos são comuns.

4 A Generic Searching Algorithm

Nós de busca

Cada nó mantém: estado, pai, ação geradora, profundidade, custo $g(n)$.

```python
from collections import deque
import heapq

class Node:
    __slots__ = ("state","parent","action","depth","g")
    def __init__(self, state, parent=None, action=None, g=0):
        self.state  = state
        self.parent = parent
        self.action = action
        self.g      = g
        self.depth  = 0 if parent is None else parent.depth + 1

def solution(n):
    """Reconstroi a sequencia de acoes/estados a partir do no meta."""
    path = []
    while n and n.parent is not None:
        path.append(n.action)
        n = n.parent
    return list(reversed(path))
```

Algoritmo Genérico de Busca em Grafo

```python
def generic_graph_search(s0, is_goal, successors, add_to_frontier, pop_frontier):
    start = Node(s0)
    if is_goal(s0):  # teste de objetivo no no inicial
        return []
    frontier = add_to_frontier(None, start)  # inicializa fronteira
    explored = set()                         # estados ja expandidos

    while frontier:
        node, frontier = pop_frontier(frontier)  # escolhe politica de expansao
        if node.state in explored:
            continue
        explored.add(node.state)

        for (a, s_next, cost) in successors(node.state):
            child = Node(s_next, parent=node, action=a, g=node.g + cost)
            if is_goal(s_next):
                return solution(child)
            # regra de duplicatas: aqui simples; pode-se manter "best_g[state]"
            if s_next not in explored:
                frontier = add_to_frontier(frontier, child)

    return None  # falha
```

Observação A **política de fronteira** define a estratégia (seção 3.5). Para busca de custo uniforme, por exemplo, a fronteira é uma `priority queue` por $g(n)$; para BFS, uma `queue`; para DFS, uma `stack`.

5 3.5 Uninformed Search Strategies

Estratégias e Estruturas de Dados da Fronteira

- **BFS** (em largura): fila FIFO.
- **DFS** (em profundidade): pilha LIFO.
- **DLS** (depth-limited search): pilha com limite L.
- **IDS** (iterative deepening): repete DLS para $L = 0, 1, 2, \ldots$.
- **Uniform-Cost** (custo uniforme, UCS): fila de prioridade por $g(n)$.

Implementações básicas

BFS

```python
def add_fifo(frontier, node):
    if frontier is None:
        frontier = deque()
    frontier.append(node)
    return frontier

def pop_fifo(frontier):
    return frontier.popleft(), frontier
```

DFS

```
def add_lifo(frontier, node):
    if frontier is None:
        frontier = []
    frontier.append(node)
    return frontier

def pop_lifo(frontier):
    return frontier.pop(), frontier
```

Uniform-cost

```
def add_priority_by_g(frontier, node):
    if frontier is None:
        frontier = []
    heapq.heappush(frontier, (node.g, id(node), node))
    return frontier

def pop_priority(frontier):
    _, _, node = heapq.heappop(frontier)
    return node, frontier
```

Propriedades (assumindo fator de ramificação b, solução mais rasa a profundidade d, e custo de passos $\geq \epsilon > 0$)

Estratégia	Completa	Ótima	Tempo	Espaço	Observações
BFS	Sim (se b finito)	Sim (custos iguais)	$O(b^d)$	$O(b^d)$	Encontra solução mais rasa; custo de memória alto.
DFS	Não (em geral)	Não	$O(b^m)$	$O(b \cdot m)$	Pode ficar preso em ramos profundos; pouca memória.
DLS (limite L)	Não (se $L < d$)	Não	$O(b^L)$	$O(b \cdot L)$	Útil quando há limite natural/segurança.
IDS	Sim	Sim (custos iguais)	$O(b^d)$	$O(b \cdot d)$	Combina completude/ótimo da BFS com memória da DFS.
UCS	Sim ($c \geq \epsilon$)	Sim	$O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$	$\geq O(b^d)$	Expande por custo crescente; otimo para custos variados.

Tabela 1: Resumo de estrategias nao-informadas (3.5). m e a profundidade maxima, C e o custo otimo.

Tree vs Graph Search e Ótimalidade

- **BFS/IDS**: ótima em *tree* quando custo de passo é uniforme; em *graph*, manter visitados evita revisitar estados.
- **UCS**: ótima (com $c \geq \epsilon$) tanto em árvore quanto em grafo, desde que se gerencie duplicatas preservando o melhor g conhecido por estado.

Boas práticas

- **Fechados** (explored set) + **frontier** determinística.
- Tabela $best\_g[state]$ para descartar caminhos piores (UCS).
- Separar *estado* da *descrição do nó* (pai, ação, custos).

Mini-Exercícios (para fixação)

1. Modele o *Missionários e Canibais* como busca: defina S, A, γ, c, s_0, G. Liste pelo menos três ações válidas em s_0.
2. Dê um exemplo simples em que DFS encontra uma solução mais rápida que BFS; explique por quê.
3. Para o 8-Puzzle, explique a diferença prática entre *tree* e *graph* search. Em que cenário *tree search* é inviável?
4. Implemente UCS para um grid $N \times N$ com custos não uniformes e compare o caminho encontrado vs BFS.
5. Explique por que IDS repete trabalho e, ainda assim, mantém custo assintótico comparável ao da BFS.

Leitura Recomendada (AIFCA 3e)

- Cap. 3.1–3.5: definição de problema, espaços de estados, busca em grafo, algoritmo genérico e estratégias não-informadas.
- Exercícios de final de capítulo sobre BFS/DFS/IDS/UCS e repetição de estados.