Capítulo 3: Solução de Problemas por Busca

Agente Solucionador de Problemas

Agente baseado em meta cujo objetivo é encontrar uma **sequência de ações** que leva a um estado desejado.

- **Formulação do problema:** definição do mesmo como um problema de busca em um espaço de **estados**.
- Deve-se definir:
  - **estados**: representações do "mundo".
  - **Operadores**: causam transições entre estados.
- Para ativar o processo de busca por uma solução, deve-se especificar estados **inicial** e **meta**.

Um exemplo:

Figura 3.3

Busca:

- processo de pesquisa pela "melhor" sequência de ações que leve a um estado meta.

Algoritmo de busca:

- Recebe como entrada um problema e retorna uma solução (sequência de ações).

Fase de execução:

- Execução da solução proposta pelo algoritmo de busca.

Figura 3.1

Formulação de Problemas:

Há 4 tipos de problemas:

- De único estado.
- De múltiplos estados.
- De contingências.
- De exploração.

Figura 3.2

Problemas de único estado

Ao executar uma ação, o agente sabe o estado resultante (tem sensores suficientes).

Exemplo:

$$5 \xrightarrow{\text{Right}} 6 \xrightarrow{\text{Suck}} 8$$

Problemas de múltiplos estados

- Ao executar uma ação, o agente tem conhecimento incompleto sobre o estado resultante (os sensores não captam tudo). Exemplo:

$$\begin{tikzcd}
\{1,2,\ldots,8\} \arrow[r, "\text{Right}"] & \{2,4,6,8\} \arrow[r, "\text{Suck}"] & \{4,8\} \arrow[d, "\text{Left}"] \\
 & \{7\} & \{3,7\} \arrow[l, "\text{Suck}"']
\end{tikzcd}$$

Problemas de contingências

- O agente requer uso de sensor durante a execução: busca e execução alternadas.

Problemas de exploração

- O agente desconhece completamente os efeitos de suas ações: aprendizagem requerida.

Formulação de problemas de único estado

Um problema é definido especificando-se:

- **Estado inicial**. Ex: "in Arad".
- **Operadores**. Ex: Arad→Zerind, Arad→Sibiu, etc.
- **Teste de meta** explícito (ex: "in Bucharest") ou implícito (ex: nodirt(x)).
- **Função de custo de caminhos**. Ex: somas das distâncias.

Problemas Exemplo

- *Toy problem* × problema real

  ↓

  mais fácil

  mais conciso

  descrição exata

  fácil comparar algoritmos alternativos

O quebra-cabeças de 8 (*8-puzzle*)

Figura 3.4

Formulação:

- Estado: locais dos ladrilhos e do vazio.
- Operadores: mover **vazio** para esquerda, direita, para cima ou para baixo.
- Teste de meta: estado meta dado.
- Custo de caminho: 1 para cada movimento.

Obs: solução ótima de problemas da família *n-puzzle* é NP-difícil.

O problema das 8 rainhas (n rainhas)

Figura 3.5

Formulação:

- Teste de meta: 8 rainhas não ameaçadas no tabuleiro.
- Custo de caminho: 0.

⇒ Várias alternativas para estados e operadores.

Alternativa 1:

- Estado: um arranjo qualquer de 0 a 8 rainhas no tabuleiro.
- Operadores: adicione uma rainha ao tabuleiro.

$\Rightarrow 64^8$ sequências a investigar.

Alternativa 2:

- Estado: um arranjo qualquer de 0 a 8 rainhas sem ameaças.
- Operadores: adicione uma rainha na coluna desocupada mais à esquerda, de forma que ela não seja ameaçada.

$\Rightarrow$ 2057 sequências a investigar.

Criptoaritmética

```
FORTY
+ TEN
+ TEN
-----
SIXTY
```

Solução: F=2, O=9, R=7, T=8, Y=6, E=5, N=0, S=3, I=1, X=4.

Formulação:

- Estado: esquema com algumas letras substituídas por números.
- Operadores: substituir toda ocorrência de uma letra por um dígito ainda não utilizado.
- Teste de meta: toda letra foi substituída.
- Custo de caminho: 0.

O mundo do aspirador

Formulação com percepção total:

- Estado: um dos 8 mostrados na figura.
- Operadores: ir para a esquerda, ir para a direita, aspirar.
- Teste de meta: nenhum lugar tem sujeira.
- Custo de caminho: 1 para cada ação.

Figura 3.6

Formulação com percepção nula:

- Estado: um subconjunto dos 8.
- Operadores: ir para a esquerda, ir para a direita, aspirar.
- Teste de meta: nenhum lugar em qualquer estado do conjunto tem sujeira.
- Custo de caminho: 1 para cada ação.

Figura 3.7

Missionários e canibais

Formulação:

- Estado: (m, c, b), onde:
  - m: número de missionários na margem de partida;
  - c: número de canibais na margem de partida;
  - b: número de barcos na margem de partida.
- Operadores: atravessar:
  - 1 missionário; ou
  - 1 canibal; ou
  - 2 missionários; ou
  - 2 canibais; ou
  - 1 missionário e 1 canibal.
- Teste de meta: atingiu o estado $(0, 0, 0)$.
- Custo de caminho: número de cruzamentos.

Problemas reais:

- Determinação de rotas.
  Exemplo: redes de computadores, planejamento de viagens aéreas.
- O problema do caixeiro-viajante.
- Leiaute de VLSI.
- Navegação de robôs.
- Montagem por robôs.

Buscando Soluções

- **Expansão** de um estado: geração de seus sucessores.
- **Estratégia** de busca: determina a <u>ordem</u> das expansões.
- **Árvore de busca**:

Figura 3.8

Estrutura de um algoritmo de busca:

Figura 3.9

Componentes de um nodo de uma árvore de busca:

- estado;
- enderêço do nodo pai;
- enderêços dos nodos filhos;
- operador usado na geração do estado;
- profundidade do nodo;
- custo do caminho desde a raiz.

Algoritmo de pesquisa:

- Idéia básica: expansão sucessiva de nodos da árvore de busca.

Figura 3.10

Estratégias de Busca

Critérios para avaliação:

- **Completeza**: sempre encontra uma solução quando existe uma?
- **Complexidade de tempo**: número de nodos gerados/expandidos.
- **Complexidade de espaço**: número máximo de nodos na memória.
- **Otimalidade**: sempre encontra uma solução de menor custo?

Complexidades de tempo e espaço medidas em termos de:

- b: fator de ramificação máximo da árvore de busca.
- d: profundidade da solução de menor custo.
- m: profundidade máxima (pode ser ∞).

Busca cega (não informada):

- só usa informações obtidas diretamente da formulação do problema.

Busca heurística (informada):

- usa informações "aproximadas", usualmente relacionadas a custos.

Estratégias de busca não informadas:

- Busca por amplitude (*breadth-first*)
- Busca por custo uniforme
- Busca por profundidade (*depth-first*)
- Busca por profundidade limitada
- Busca por aprofundamento iterativo (*iterative deepening*)

Busca por amplitude

- Expande um nodo de menor profundidade.

Figura 3.11

```
function BREADTH-FIRST-SEARCH(problem) returns a solution or failure
    GENERAL-SEARCH(problem,ENQUEUE-AT-END)
```

Propriedades:

- Completo: sim (se b é finito).
- Tempo: $1 + b + b^2 + \cdots + b^d = O(b^d)$.
  $\Rightarrow$ exponencial
- Espaço: $O(b^d)$.
  (nodos do último nível na memória)
- Ótimo: sim, se custo fixo por expansão.

Busca por custo uniforme

- Expande nodo n de menor custo ($g(n)$) associado.

Figura 3.13

Propriedades:

- Completo: sim, se custo por passo $\geq \epsilon$.
- Tempo: $|\{n \mid g(n) \leq$ custo da solução ótima$\}|$.
- Espaço: idem.
- Ótimo: sim, se $\forall n\ g(suc(n)) \geq g(n)$.

Busca por profundidade

- Expande nodo de maior nível.

Figura 3.14

function DEPTH-FIRST-SEARCH(*problem*) **returns** a solution or failure
GENERAL-SEARCH(*problem*,ENQUEUE-AT-FRONT)

Outra opção: **algoritmo recursivo**.

Propriedades:

- Completo: não.
  pode falhar se a árvore tem ramos infinitos.
  evitar estados repetidos...
- Tempo: $O(b^m)$.
  O problema é se $m >> d$...
- Espaço: $O(bm)$; linear!
- Ótimo: não.

Busca por profundidade limitada

- Busca por profundidade com limite de profundidade l.
- **Implementação:** nodos de profundidade l não têm sucessores.

Propriedades:

- Completo: não.
- Tempo: $O(b^l)$.
- Espaço: $O(bl)$.
- Ótimo: não.

Problema: descobrir l.

Busca por aprofundamento iterativo

- Busca por profundidade com limites de profundidade $0, 1, 2, \ldots$

Figura 3.15

Propriedades:

- Completo: sim.
  $\Rightarrow$ como busca por amplitude.
- Tempo: $(d+1)b^0+db^1+(d-1)b^2+\cdots+b^d = O(b^d)$.
- Espaço: $O(bd)$.
  $\Rightarrow$ como busca por profundidade!
- Ótimo: sim, se custo fixo por expansão.
  $\Rightarrow$ como busca por amplitude.
  como modificá-lo para "custo uniforme"?

Busca bidirecional

- Busca simultânea para frente e para trás.

Figura 3.17

Propriedades:

- Completo: sim.
- Tempo: $2O(b^{d/2}) = O(b^{d/2})$.
- Espaço: $O(b^{d/2})$.
- Ótimo: sim.

Comparando as estratégias:

Figura 3.18

Evitando Estados Repetidos

Como lidar com estados repetidos:

- Gerá-los (se existirem).
- Não gerar nodo sucessor com mesmo estado que o pai.
- Não gerar nodo sucessor com mesmo estado que ancestral (não criar caminhos com ciclos).
- Não gerar nodo com estado já gerado.