SOLUÇÃO DE PROBLEMAS POR BUSCA

Esse capítulo descreve um tipo de agente baseado em objetivo chamado Agente Solucionador de Problemas – *Problem-Solving Agent.* Esse tipo de agente decide o que fazer encontrando uma seqüência de ações que o leve ao objetivo. Inicialmente, veremos os algoritmos de busca não informada (que não recebem informações sobre o problema). Posteriormente (capítulo 4) veremos os algoritmos de busca informada (que recebem alguma informação sobre onde procurar por uma solução).

3.1. AGENTE SOLUCIONADOR DE PROBLEMAS:

De forma resumida, a **formulação do objetivo** é o primeiro passo para resolver problemas. Considera-se objetivo como sendo um conjunto de estados para os quais a meta é alcançada. Assim, a tarefa do agente consiste em encontrar uma seqüência de ações que o leve ao objetivo.

Tendo feito isso, ele precisa fazer a **formulação do problema,** ou seja, definir quais as ações e os estados devem ser considerados, dado um objetivo.

Considere o seguinte exemplo: agente que deseja chegar em *Bucharest* a partir de *Arad*, tendo duas opções *Sibiu*, *Timisoara* e *Zerind*[1]*:*

- Agente não tem informação sobre o problema ⇒ o melhor que pode ser feito é escolher aleatoriamente;
- Agente com conhecimento do mapa (papel ou memória) ⇒ o agente pode usar essa informação para considerar estados subseqüentes em uma viagem hipotética (execução hipotética).

Em geral, um agente com várias opções imediatas, a partir de um estado, pode decidir o que fazer, examinando primeiramente as diferentes seqüências possíveis que o leve ao objetivo, e então escolher a melhor.

[1] Nomes de cidades da Romênia usadas como exemplos pelo autor do livro texto.

Esse processo para escolher a "melhor" sequência de ações é chamado de **BUSCA**. O algoritmo de busca recebe como entrada um problema e retorna uma solução na forma de uma seqüência de ações. Quando uma solução é encontrada é iniciado o processo de **execução**.

Assim, a figura abaixo ilustra um esquema de projeto/modelagem *"formula, busca e executa"* para um agente.

```
function SIMPLE_PROBLEM_SOLVING_AGENT (percept) returns an action
       inputs: percept, a percept
       static: seq, an action sequence, initially empty
              state, some description os the current world state
              goal, a goal, initially null
              problem, a problem formulation

       state ← UPDATE_STATE(state, percept)
       if seq is empty then do
              goal ← FORMULATE_GOAL(state)
              problem ← FORMULATE_PROBLEM (state, goal)
              seq ← SEARCH (problem)
       action ← FIRST (seq)
       seq ← REST (seq);
       return action
```

Observe que:

- Após formular o problema e o objetivo, o agente chama um processo de busca;
- Assumindo ambiente estático ⇒ não considera qualquer mudança no ambiente;
- O projeto do agente também assume que o estado inicial é conhecido ⇒ ambiente acessível;
- A idéia de enumerar as "seqüências alternativas de ações" sugere que o ambiente é discreto;

⇒ Ambiente Determinístico.

FORMULAÇÃO DE PROBLEMAS:

Durante a formulação do problema, algumas características do ambiente são desconsideradas visto que não são relevantes para o objetivo do agente. Por exemplo, no problema de encontrar uma rota entre as cidades de Arad e Bucharest, não se considera condições da estrada, tempo (clima), barreira policial, etc.... por que são irrelevantes para o problema de encontrar uma rota entre duas cidades. O processo de remover detalhes de representação é chamado de abstração. Além de abstrair a descrição dos estados, devemos abstrair também as ações. Exemplo: a ação relevante para encontrar uma rota é mudar de localização. Desconsideram-se ações como ligar o rádio, diminuir velocidade devido um radar, etc....

Um problema pode ser definido formalmente por quatro componentes:

- **Estado inicial** é o estado que o agente começa o processo de busca. Por exemplo, o estado inicial para o agente no mapa da Romênia é *In(Arad)*.

- Descrição das possíveis **ações** do agente (operadores). A formulação mais comum é utilizar a **Função Sucessor**. Dado um estado qualquer x, *SUCCESS_FN(x)* retorna um conjunto de pares <*ação, sucessor*>, onde cada *ação* é válida a partir do estado *x*, e cada *sucessor* é um estado que pode ser alcançado a partir de *x*, executando-se a ação *ação.* Por exemplo, do estado *In(Arad)*, a função sucessor retorna *{<Go(Sibiu), In(Sibiu)>,<Go(Timisoara), In(Timisoara)>, <Go(Zerind),In(Zerind)>}*.

Juntos, o estado inicial e a função sucessor definem o **espaço de estados** de um problema, ou seja, o conjunto de todos os estados alcançáveis a partir do estado inicial. **Caminho** é uma seqüência de ações que leva o agente de um estado a outro.

- **Teste de Meta** é um teste que deve ser aplicado para determinar se um dado estado é final ou não.

- **Custo do Caminho** é uma função que atribui uma medida de custo a um determinado caminho. Para um agente resolvedor de problemas escolhe-se uma função de custo que reflete sua medida de desempenho. O custo de um caminho é dado pela soma do custo de cada ação.

A **solução** para um dado problema é o caminho do estado inicial até o objetivo. **Solução ótima** é a solução com o menor custo do caminho.

3.2. EXEMPLOS DE PROBLEMAS:

PROBLEMAS REAIS X "TOY PROBLEMS":

⇓

Mais fácil, mais conciso, descrição exata e

fácil comparar algoritmos alternativos.

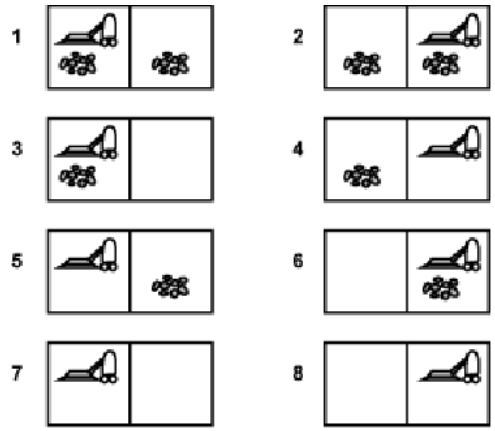
APLICAÇÕES EM PROBLEMAS REAIS:

Cálculo de rotas: rotas em redes de computadores, sistemas de planejamento de viagens, planejamento de rotas de aviões, caixeiro viajante, ...

Scheduling e projeto de VLSI.

EXEMPLOS DE "TOY PROBLEMS":

ASPIRADOR DE PÓ SIMPLIFICADO:



Formulação do Problema:

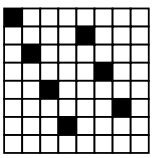
- **Estados:** 8 estados exibidos na figura anterior;
- **Operações:** mover a esquerda, mover a direita e limpar;
- **Teste de Meta:** nenhuma sujeira no tabuleiro;
- **Custo de um caminho:** cada ação custa 1.

PROBLEMA DO QUEBRA-CABEÇA DE 8 (8-PUZZLE):

Formulação do problema:

- **Estados:** Um estado define a localização de cada um dos 8 quadrados. Pode incluir também a localização do branco
- **Operações:** mover o espaço em branco para: direita, esquerda, cima e para baixo
- ***Goal test*:** Se o estado corresponde a configuração a esquerda
- **Custo de um caminho:** cada passo custa 1, o custo do caminho é dado pela soma dos passos

PROBLEMAS DAS OITOS RAINHAS:



Formulação do problema:

- **Teste de Meta:** 8 rainhas no tabuleiro, sem que nenhuma possa atacar a outra;
- **Custo de um caminho:** zero, ou seja qualquer solução tem peso igual. Custo total determinado pelo tempo de busca.

Existem diferentes maneiras de formular os estados e as operações:

Alternativa 1:

- **Estados:** um estado para cada configuração de 0 a 8 rainhas tabuleiro, logo 64^8 estados;
- **Operações:** adicionar uma rainha em algum lugar no tabuleiro.

Alternativa 2:

- **Estados:** um estado para cada configuração de 0 a 8 rainhas tabuleiro;
- **Operações:** Inserir uma nova rainha (na posição mais a esquerda) no tabuleiro sem que esta seja atacada.

Alternativa 3:

- **Estados:** configurações com 8 rainhas, uma em cada coluna;
- **Operações:** mova a rainha para outra posição na mesma coluna (mover para uma posição sem ataque quando possível).

CRYPTOARITMÉTICA:

FORTY		29786
+ TEN	solução →	850
+ TEN		850
---------		--------
SIXTY		31486

F=2, O=9, R=7, T=8,Y=6, etc...

Formulação do Problema:

- **Estados:** um tabuleiro com algumas letras substituídas por dígitos;
- **Operações:** substituir todas as ocorrências de uma letra por um dígito que não existe no tabuleiro;
- **Teste de Meta:** se o tabuleiro contém somente dígitos e representa a soma correta
- **Custo de um caminho:** zero, todas as soluções são igualmente válidas

MISSIONÁRIOS E CANIBAIS:

Formulação do Problema:

- **Estados: u**ma seqüência ordenada que representa o número de missionários, canibais e o barco. Estado inicial (3,3,1);
- **Operações:** transportar: 1 missionário; 1 canibal; 2 missionários; 2 canibais ou um de cada para outra margem através do barco;
- ***Goal test*:** estado (0,0,0);
- **Custo de um caminho:** cada ação custa 1.