3-Representação de Conhecimento e Solução de Problemas

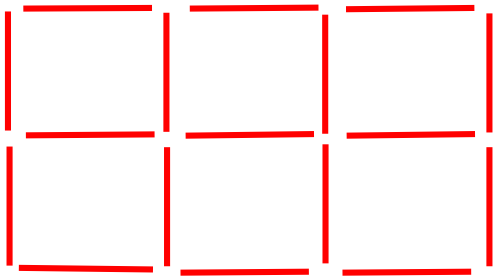
Introdução

- Conhecimento
  - informação armazenada ou modelos usados para interpretar prever, responder apropriadamente ao ambiente
  - forma e conteúdo do conhecimento e.g. tabela × calculadora

- Problema
  - coleção de informações que um agente utiliza para decidir

- Interesse da IA
  - representações que podem ser definidas formalmente
  - apropriadas para mecanização e computação

Exemplo: 3 quadrados

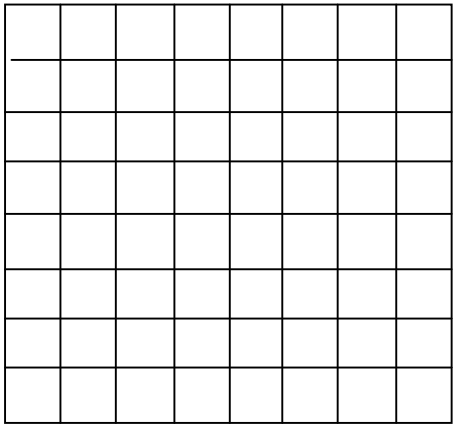


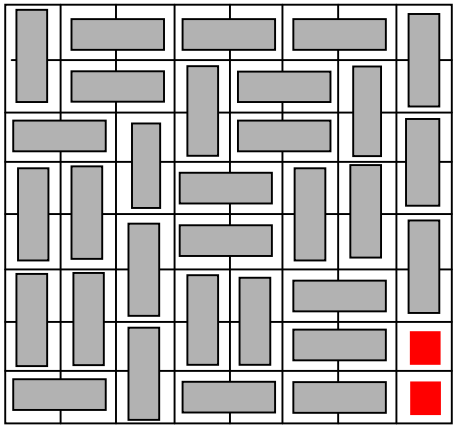
1 - Elemento primitivo é o palito; removemos 5 palitos/vez; verificar resultado (6188)

2 - Elemento primitivo é um quadrado; removemos 3/vez; escolher configuração com exatamente 5 palitos a serem removidos (20)

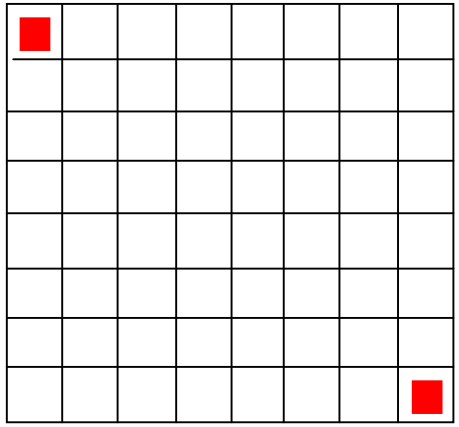
3 - Notar que existem 17 palitos e que, depois de remover 5, sobram 12. Estes 12 só podem constituir 3 quadrados se eles não tiverem arestas em comum (2)

Exemplo: problema dos 31 dominós





OK



?

Sempre existe arranjo onde dois quadrados arbitrários não são cobertos por uma peça?

Representação de conhecimento

- Princípio da representação
  - representa todas as situações de interesse
  - estável
  - identifica redundância
  - estruturas de dados e operações permitem algoritmos simples

- Quatro partes fundamentais de uma representação
  - léxica: símbolos do vocabulário da representação
  - estrutural: restrições nos arranjos de símbolos
  - procedimental: procedimentos para criar, modificar e obter soluções
  - semântica: associação de significado às descrições

- Propósitos da representação
  - interpretar o ambiente
    - informação sensorial usada com representações internas (modelos) de objetos
    - informação interpretada comparando dado sensorial com descrições de objetos
  - organizar para identificar
    - similaridades e diferenças entre objetos
    - similaridades e diferenças entre eventos

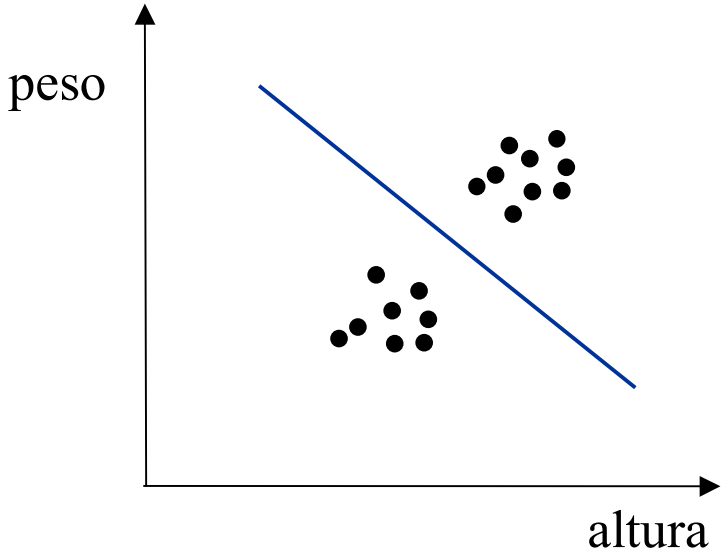
- questionar
  - modelos internos permitem elaborar questões sobre eventos e objetos
  - porque um certo evento ocorre apesar do modelo prever o contrário ?
  - revisão de modelos

- prever
  - modelo interno permite prever eventos que resultam de ações
  - modelo matemático permite prever o comportamento do sistema

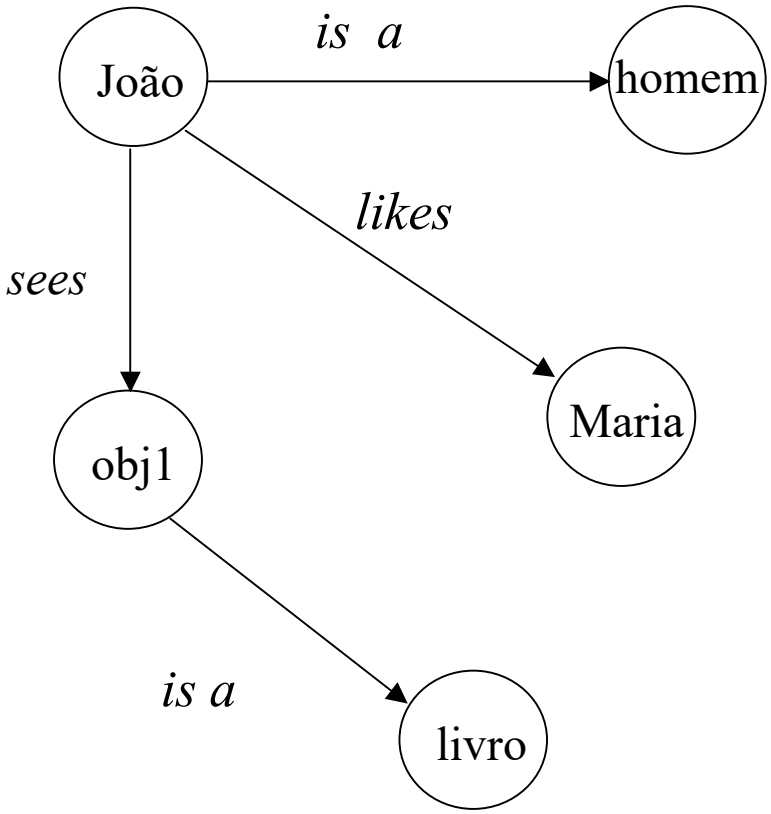
- deduzir
  - representações usadas para explicitar um novo conhecimento
  - deduções realizadas com o conhecimento original
  - todo homem é mortal. João é um homem. João é mortal.

Representação de conhecimento na IA

- Espaço de atributos (*feature/decision space*)
  - associa cada dimensão de um espaço a um atributo

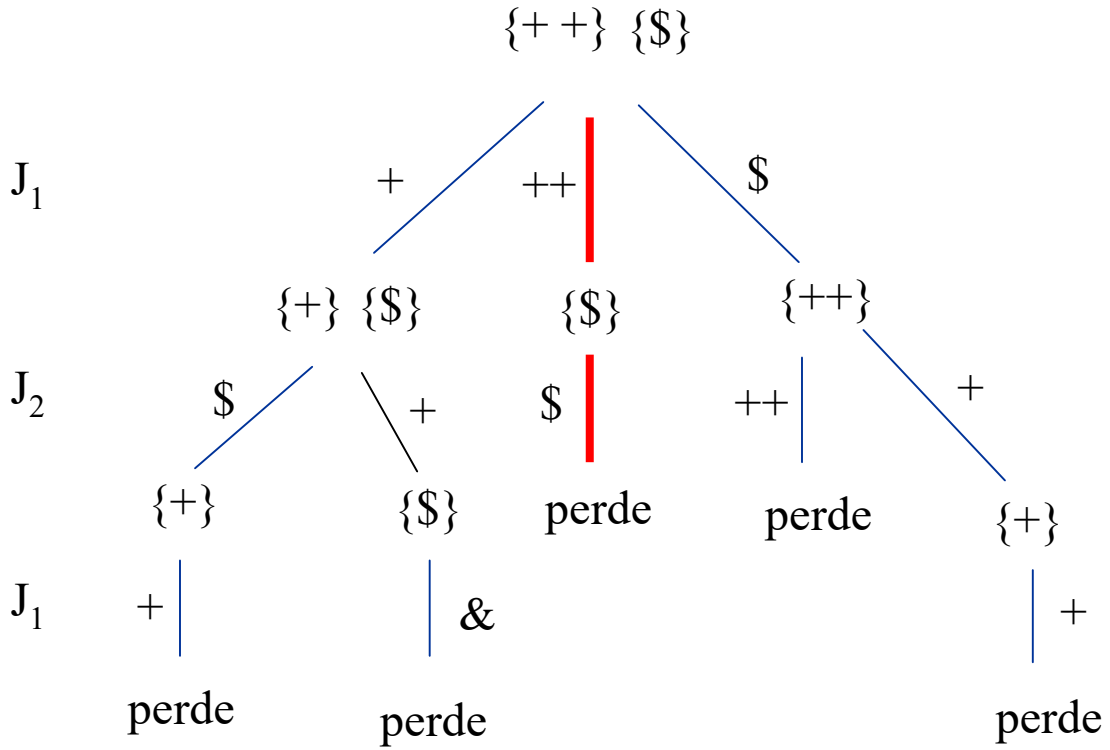


- Rede semântica (grafo relacional)
  - descreve relações entre objetos



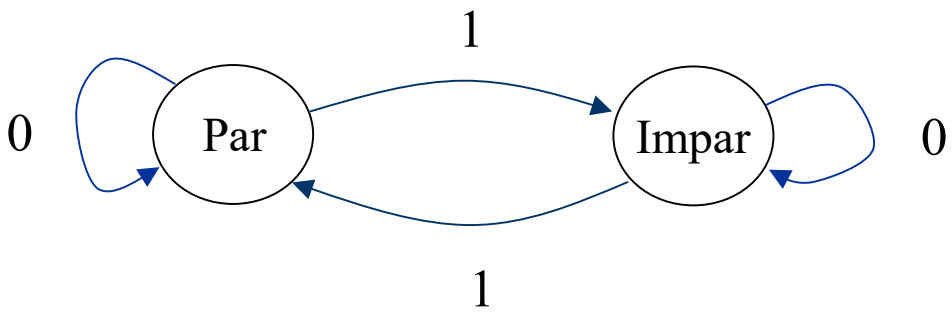
- Rede semântica
  - léxico: nós, arcos, rótulos (nomes)
  - estrutural: arcos conectam um nó cauda a um nó cabeça
  - semântica: nós e arcos denotam itens específicos de uma aplicação
  - procedimentos:
    - construtores (nós, arcos e nomes)
    - leitores (lista de arcos que entram e saem de um nó)

- Árvores de decisão
  - cada nó (estado) tem um ou mais estados sucessores
  - caminho de um estado inicial até estado(s) desejado(s)



perde quem pegar último elemento

- Grafo de transição de estados
  - composto nós e arcos rotulados
  - transição através de um arco requer condições de arco satisfeitas



- Regra de produção
  - sentença na forma condição-ação/conclusão

se < *condição*> **então** <*ação/conclusão*>

- Frame
  - representa conhecimento sobre objetos ou eventos comuns a uma situação

```
( ?
  a dog
     with [breed = ?]
          [feet   = 4]  default
          [ears   = 2] default
          [name  = ?]
          [size    = ?]
          [color   = ?] )
```

```
( dog1
  a dog
     with [breed =  dalmata]
          [feet   = 3]
          [ears   = 2] default
          [name  = Pirata]
          [size    = ?]
          [color   = preto e branco] )
```

- Lógica matemática
  - cálculo proposicional
  - cálculo de predicados

$$(\forall\ x)\ [\ dog\ (x) \Rightarrow animal\ (x)\ ]$$

- Matemática
  - série de potências, de Fourier, matrizes, coordenadas espaciais

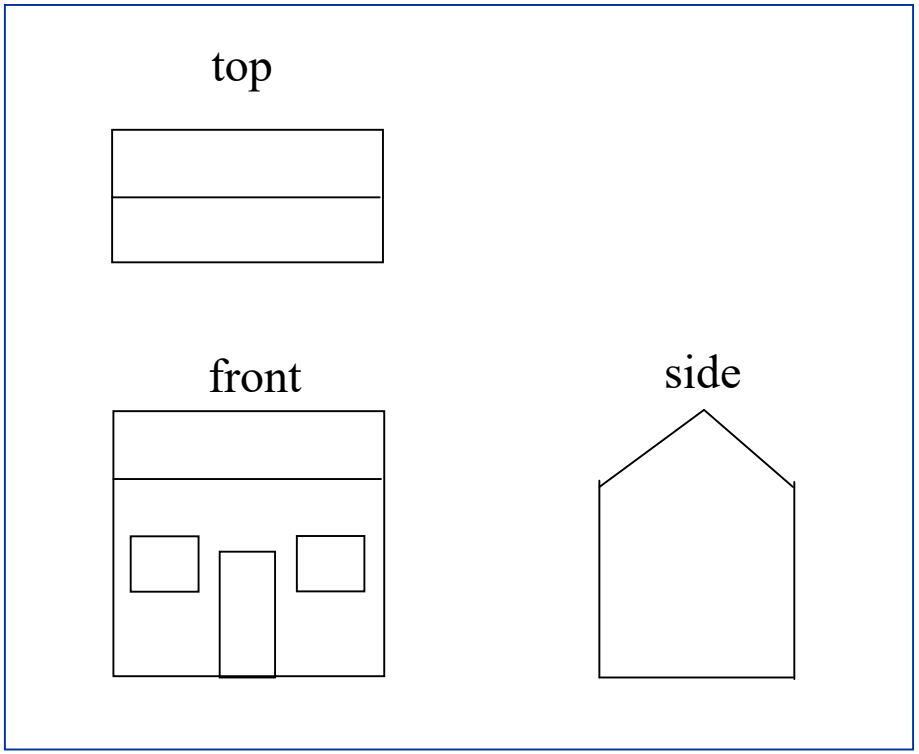
$$R(x,y) = a\ \cos x + b\ \sin y$$

- Procedimentos

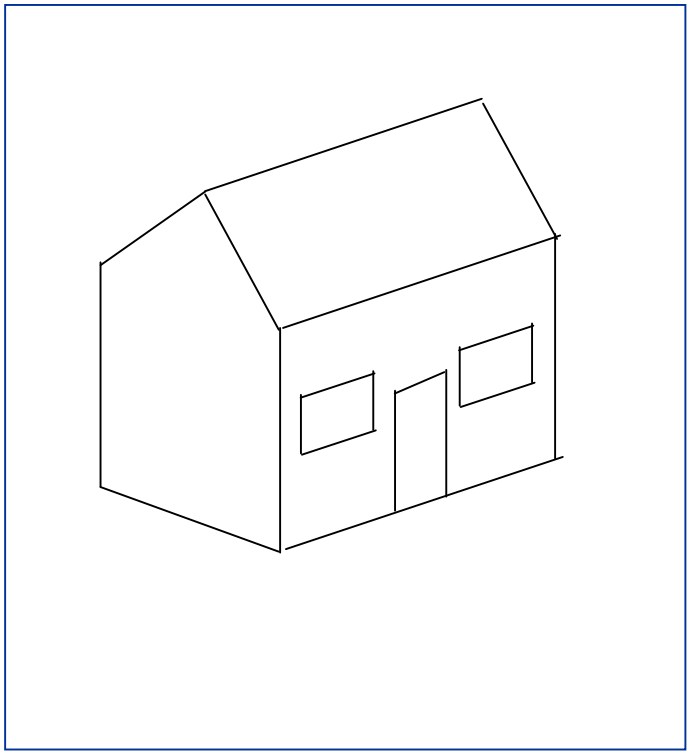
  – conhecimento sobre um domínio formulado através de procedimentos

```
procedure boil water
 begin
     obtain a pot and put water in it
      put the pot over range burner
      turn burner on
      turn off burner when steam rises
end
```

- Isomorfismos (ícones, analogias)
  - representações com relações estruturais diretas com os objetos

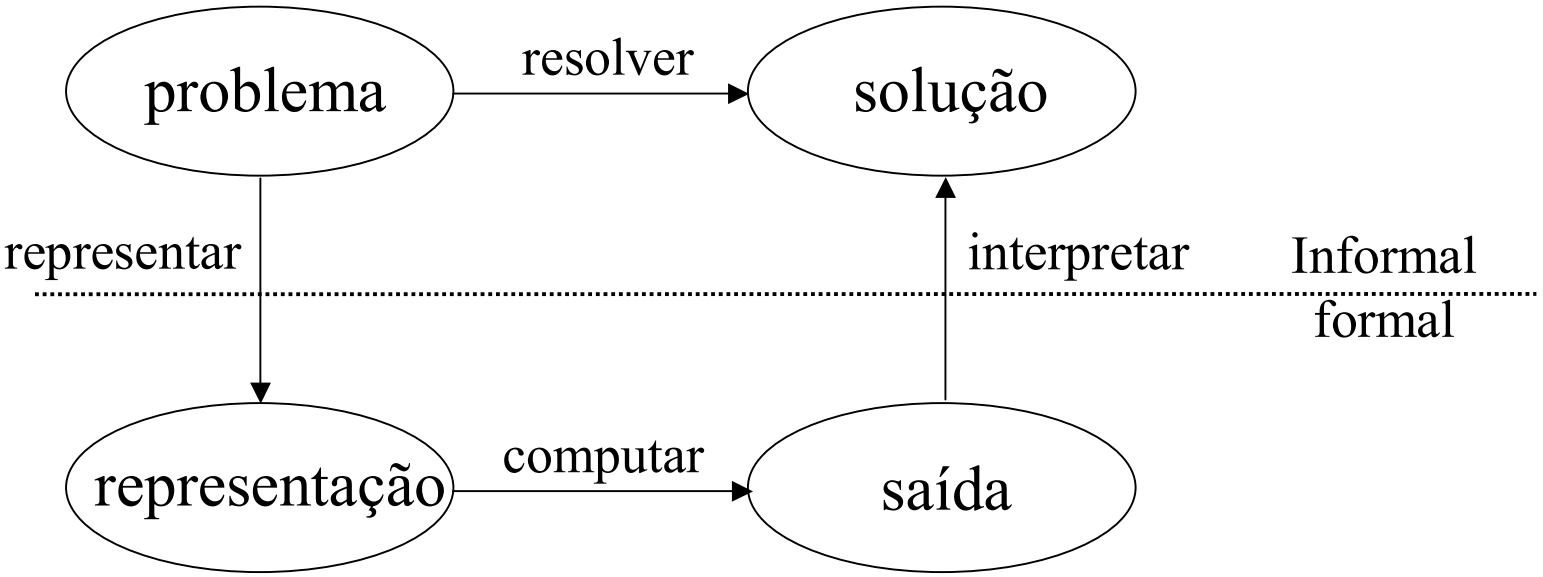


Representação



Realidade

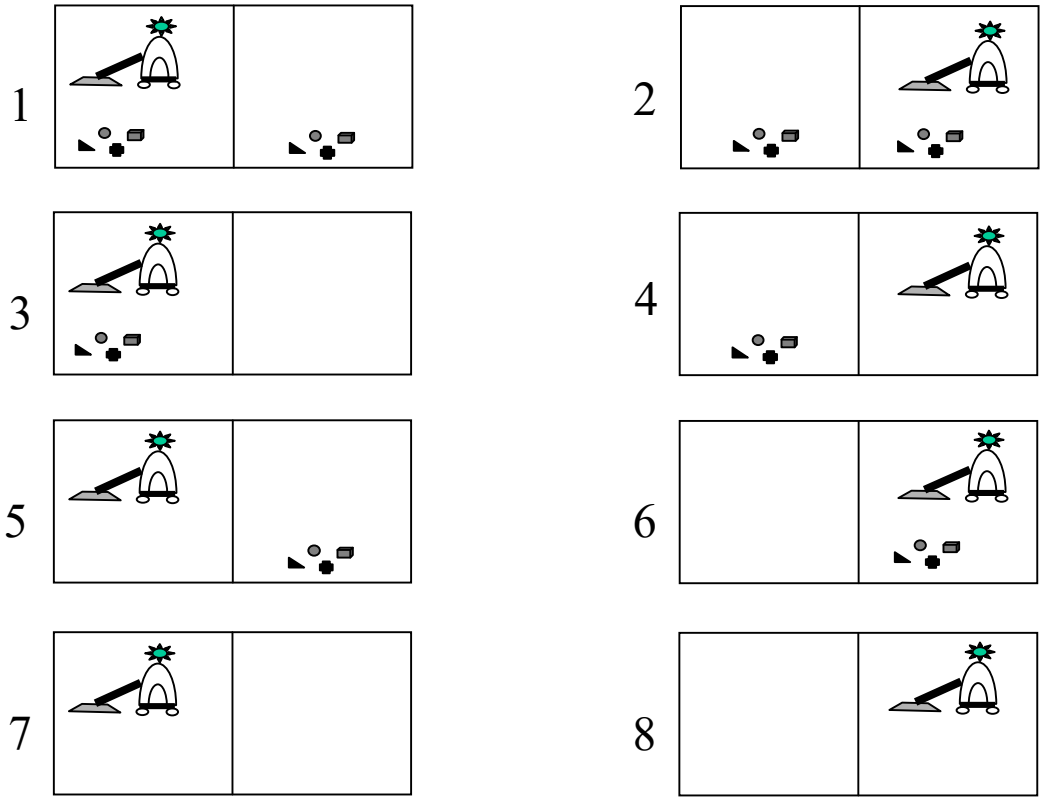
Papel das representações na solução de problemas



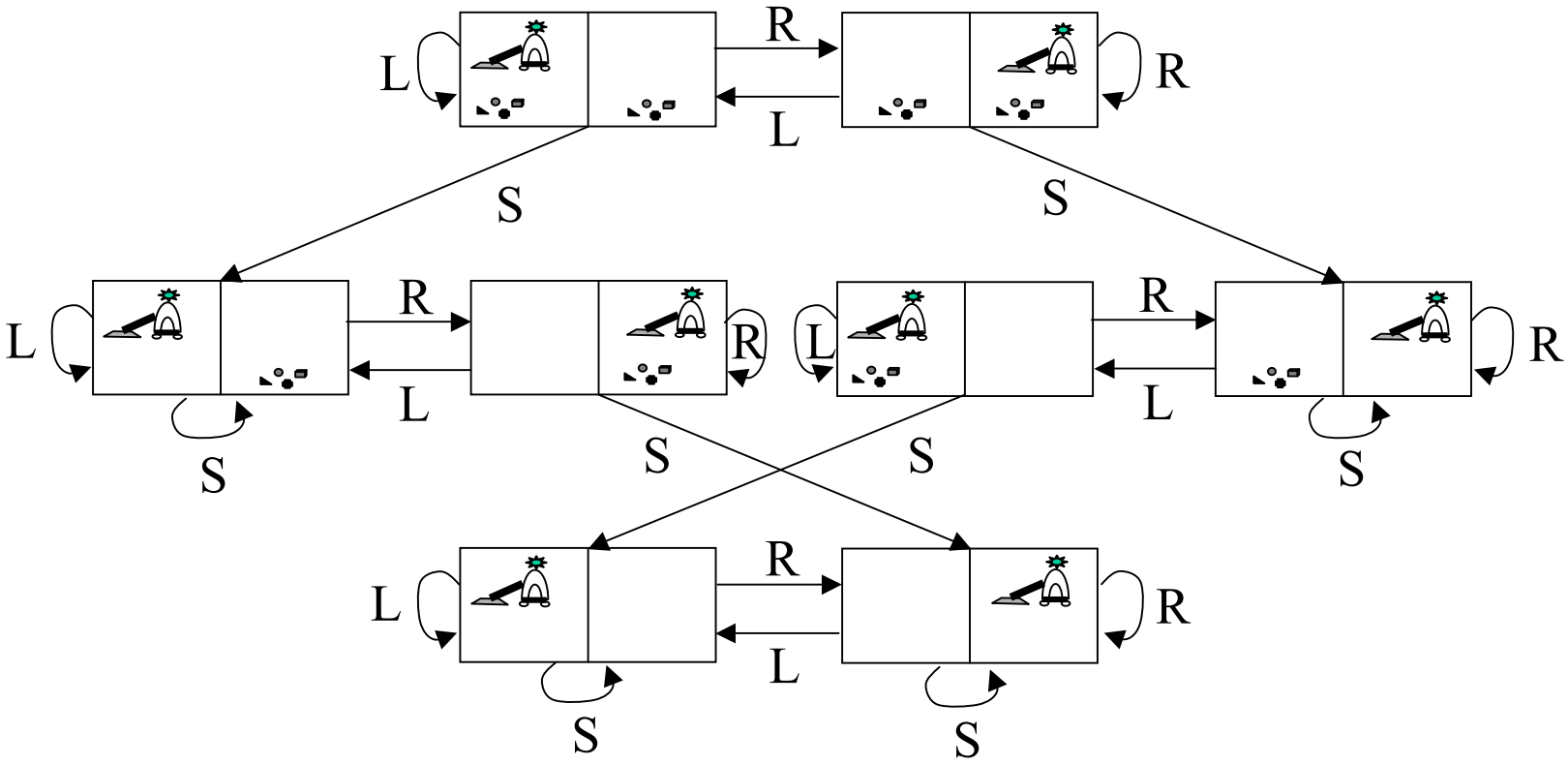
Estado e espaço de estado

- Espaço de estado
  - representação vista como grafo onde
    - nós denotam estados
    - arcos denotam transições entre estados
  - espaço de estado: conjunto de todos os estados possíveis
- Estados atingíveis
  - estados alcançados a partir do estado inicial e uma sequência de ações
- Single state × multi state

Exemplo: vacuum world



Espaço de estado vacuum world



Solução de problemas em IA

- Abordagem simbólica
  - baseada na physical symbol hypothesis
  - operações lógicas aplicadas à bases de conhecimento declarativas
  - top-down, knowledge-based approach

- Abordagem sub-simbólica
  - processa sinais ao invés de símbolos
  - redes neurais, algoritmos genéticos, clustering
  - bottom-up, animat (animal-materials) approach

Exemplo: problem solving agent

- Componentes de um problema
  - estado inicial
  - ações disponíveis ao agente
  - modelo de transição (efeito ações)
  - teste de meta
  - função de avaliação de um caminho (custo)

- Agente orientado por metas
  - meta (*goal*): organiza comportamento (objetivos e ações)
  - meta: conjunto de estados que satisfazem objetivos
  - tarefa: determinar sequência de ações para atingir a meta

- Solução de problemas
  - definir a meta (dada a situação corrente e o critério desempenho)
  - formular o problema: ações e estados a considerar (p/ dada meta)
  - resolver: busca para determinar uma sequência de ações (solução)

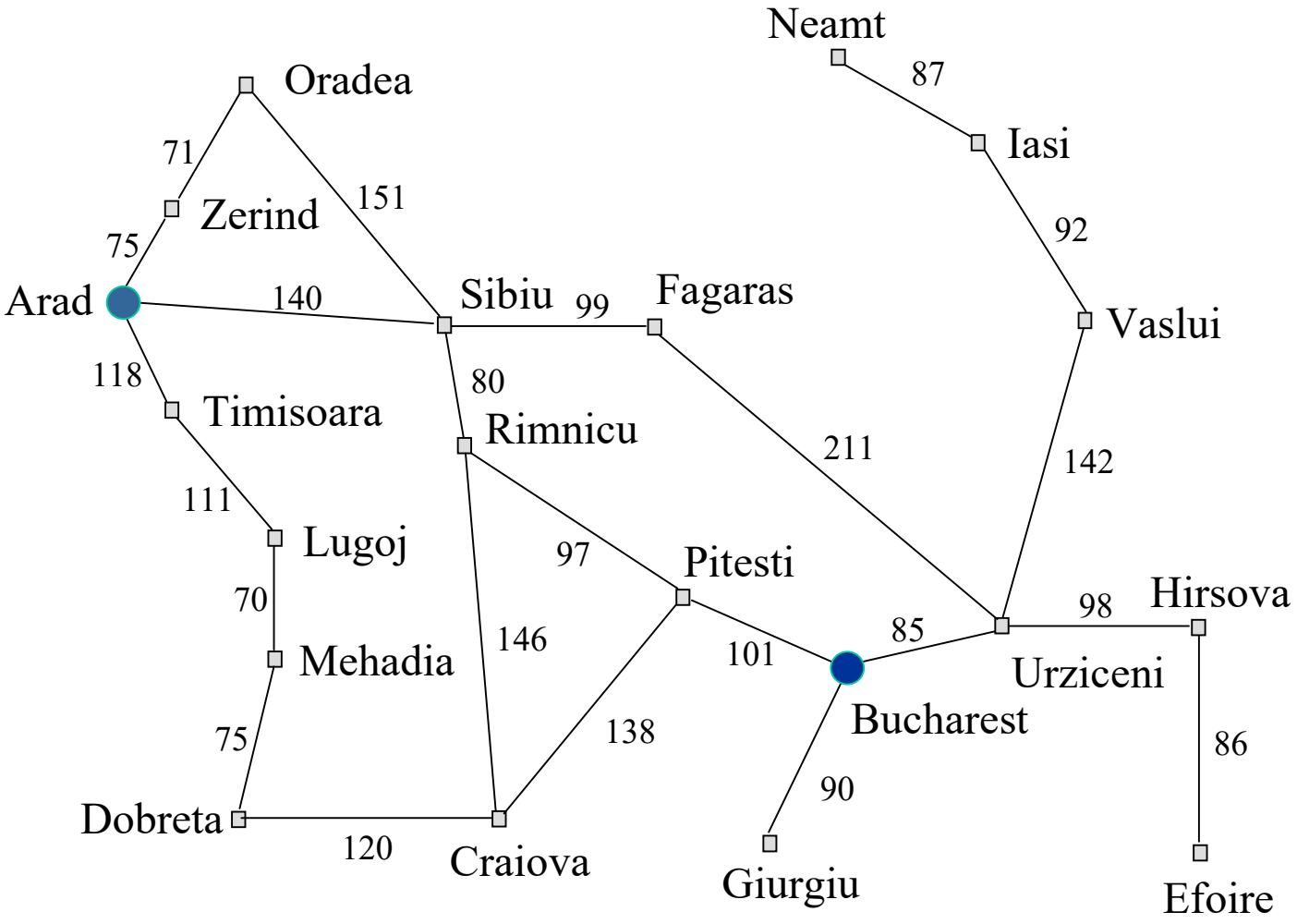
- Projeto do agente
  - representação (estruturada)
  - algoritmo de busca
  - nível de abstração
  - ambiente determinístico

function SIMPLE_PROBLEM_SOLVING_AGENT (*percept*) **returns** an action

persistent: *seq*: an action sequence, initially empty
state: some description of the current 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*)
if *seq* = failure **then return** a null action
action ← FIRST(*seq*)
seq ← REST (seq)
return *action*

Caminho de Arad para Bucareste



- Estado inicial
  - *In* (*Arad*) condição inicial do agente
- Ações disponíveis
  - ACTION(*s*): retorna conjunto ações aplicáveis (factíveis) em *s*
  - ACTION(*In* (*Arad*)) = {*go* (*Sibiu*), *go* (*Timisoara*), *go* (*Zerind*)}
- Modelo de transição
  - função RESULT(*s*, *a*)
  - retorna estado que resulta quando se aplica *a* em *s*
  - RESULT (*In* (*Arad*), *Go* (*Zerind*)) = *In* (*Zerind*)

estado inicial + modelo transição + ações definem espaço de estado

espaço estados forma um grafo (nós: estados, arcos: ações)

caminho no espaço estado: sequência de estados conectados por ações

- Teste de meta
  - determina se estado atual é a meta e.g. {(*In* (*Bucharest*)}

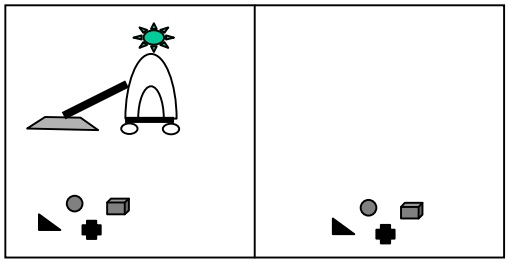
- Avaliação (custo) de um caminho
  - atribui um valor numérico a cada caminho
  - reflete critério (medida) de desempenho
  - exemplo: soma distâncias de uma origem a um destino

    $\sum c\ (x, a, y)$, $c\ (x, a, y)$, custo de um passo, de x para y

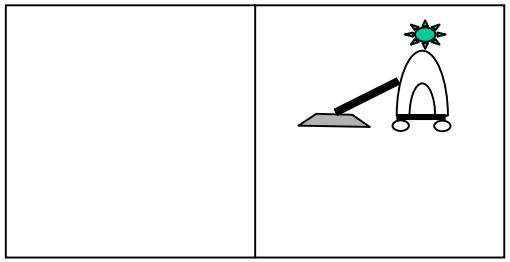
- Solução
  - caminho do estado inicial à meta (objetivo)
  - qualidade da solução: custo do caminho
  - solução ótima: menor caminho

Vacuum world

1. estados: posição do agente e presença sujeira em n locais ($n2^n$ estados)
2. estado inicial: um dos estados do ambiente
3. ações: *right* (R), *left* (L), *suck* (S)
4. modelo de transição: gera estados aplicando ações L, R e S
5. teste de meta: verifica se locais estão limpos
6. avaliação (custo) de um caminho: número total passos

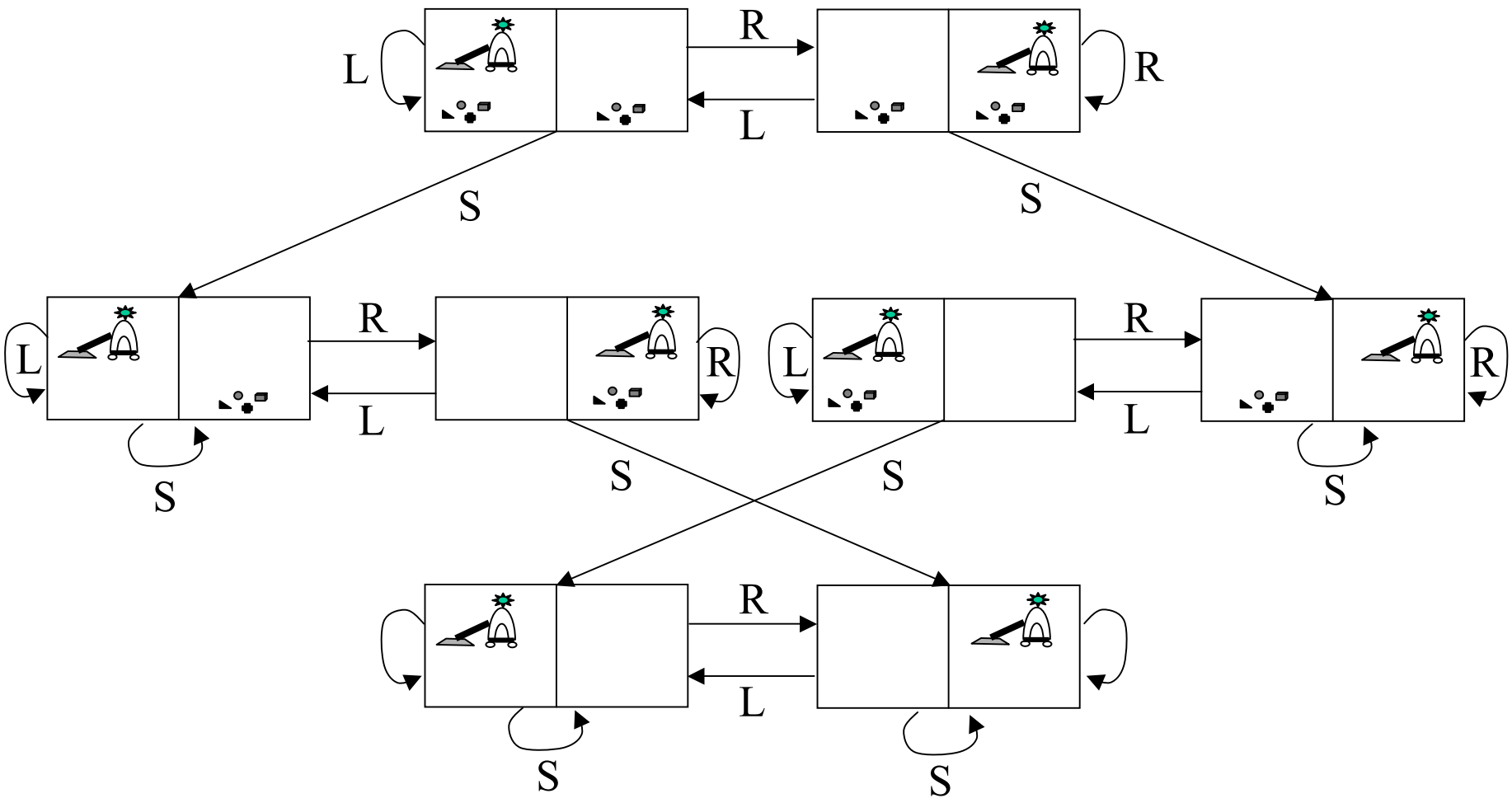


Estado inicial



Meta

Espaço de estado



Algoritmos de busca

- busca em um espaço de estado (e.g. grafo)
- árvore de busca
  - estado inicial
  - função sucessor
- raiz da árvore: estado inicial
- expandir nó: aplicar ações factíveis e gerar novos nós

- nós folha: nós sem filhos
- fronteira: conjunto de nós folhas para expansão (*open list*)
- estado a expandir: estratégia de busca
- árvore de busca ≠ espaço de estado
- conjunto explorado: conjunto de nós expandidos (*closed list*)

Descrição informal de algoritmos de busca em árvore

function TREE_SEARCH (*problem*) **returns** a solution or failure

initialize the frontier using the initial state of *problem*

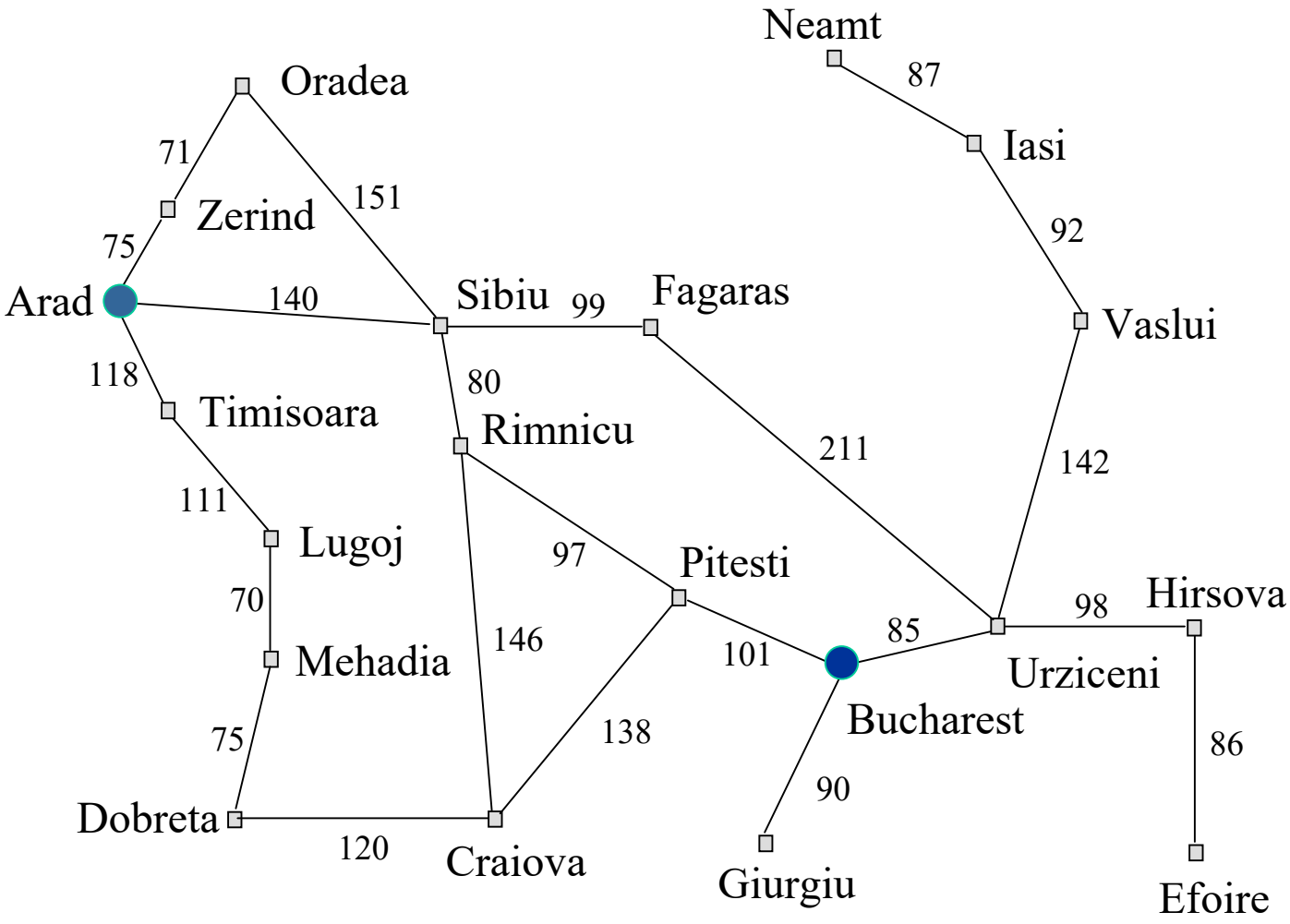
loop do
if the frontier is empty **then return** failure
choose a leaf node and remove it from frontier
if node contains a goal state **then return** the corresponding solution
expand the chosen node and add the resulting nodes to the frontier
end

Descrição informal de algoritmos de busca em grafo

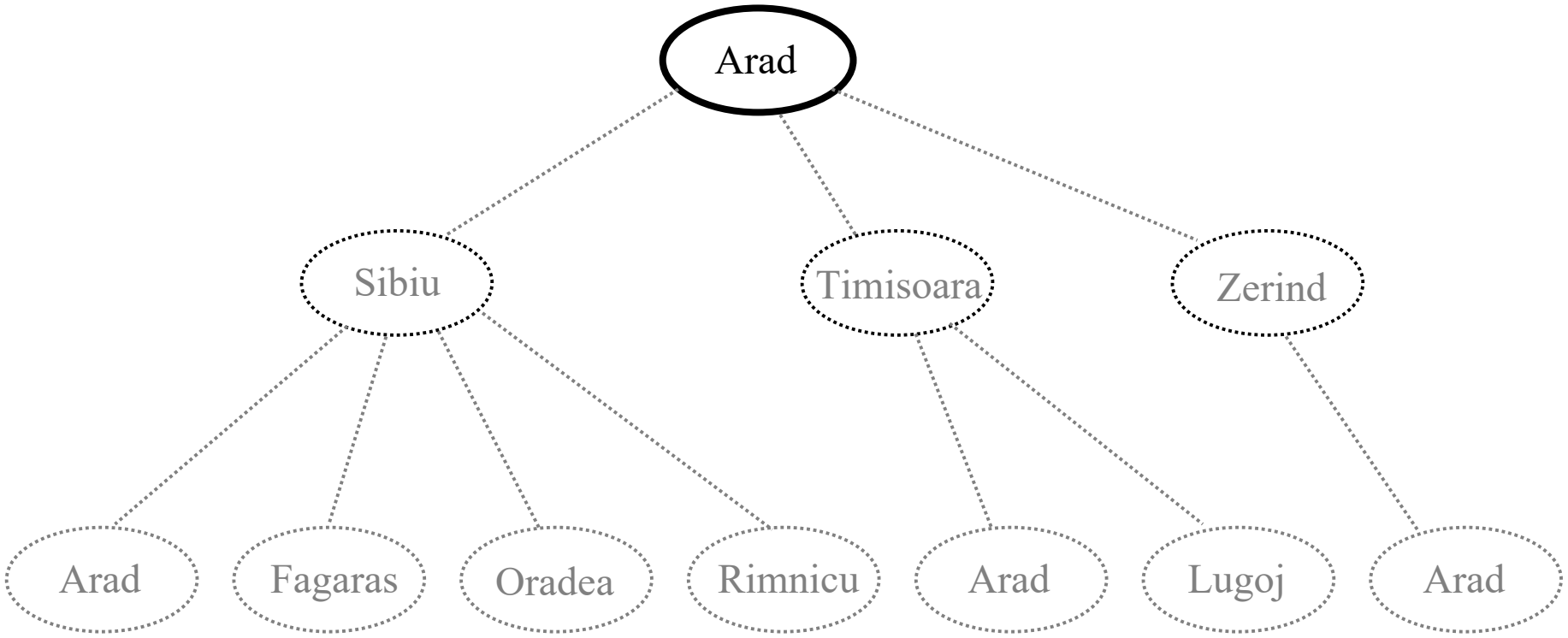
function GRAPH_SEARCH (*problem*) **returns** a solution or failure

initialize the frontier using the initial state of *problem*
initialize the *explored* set to be empty

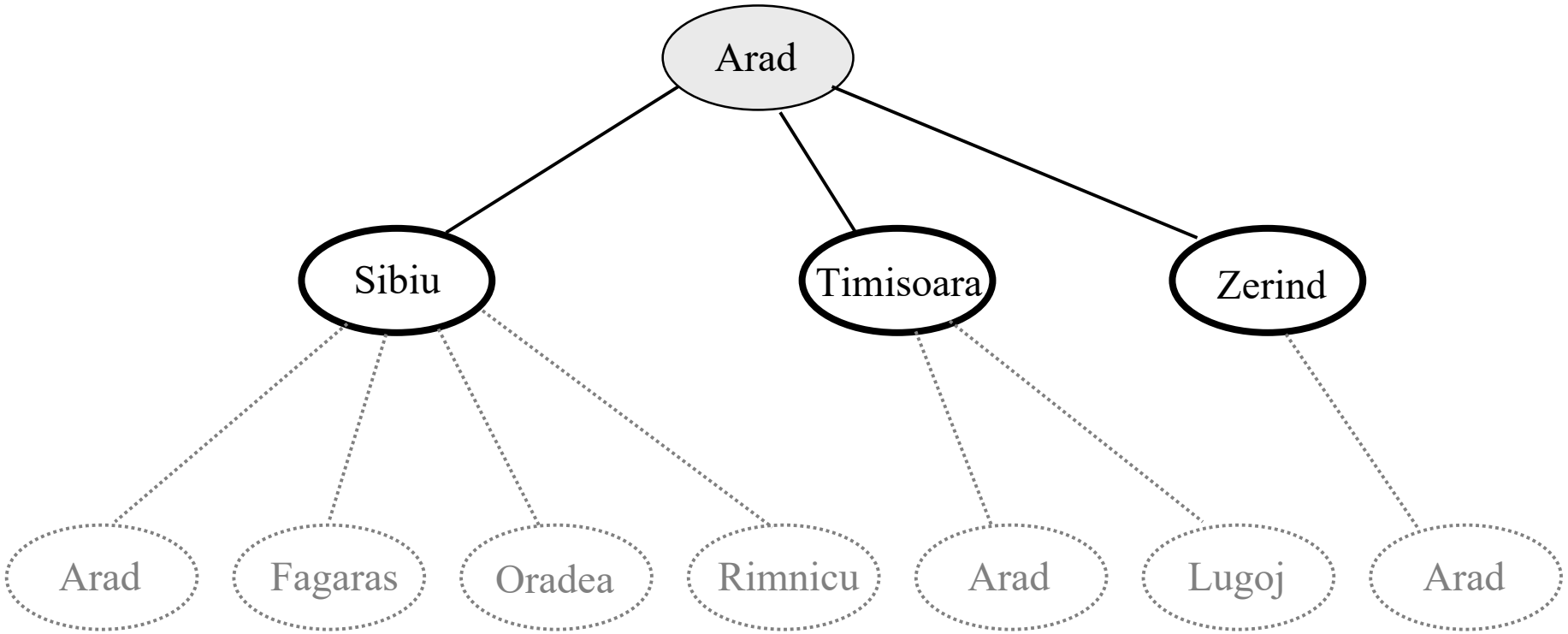
loop do
if the frontier is empty **then return** failure
choose a leaf node and remove it from frontier
if node contains a goal state **then return** the corresponding solution
add the node to the *explored* set
expand the node and add the resulting nodes to the frontier
only if not in the *frontier* or *explored* set
end



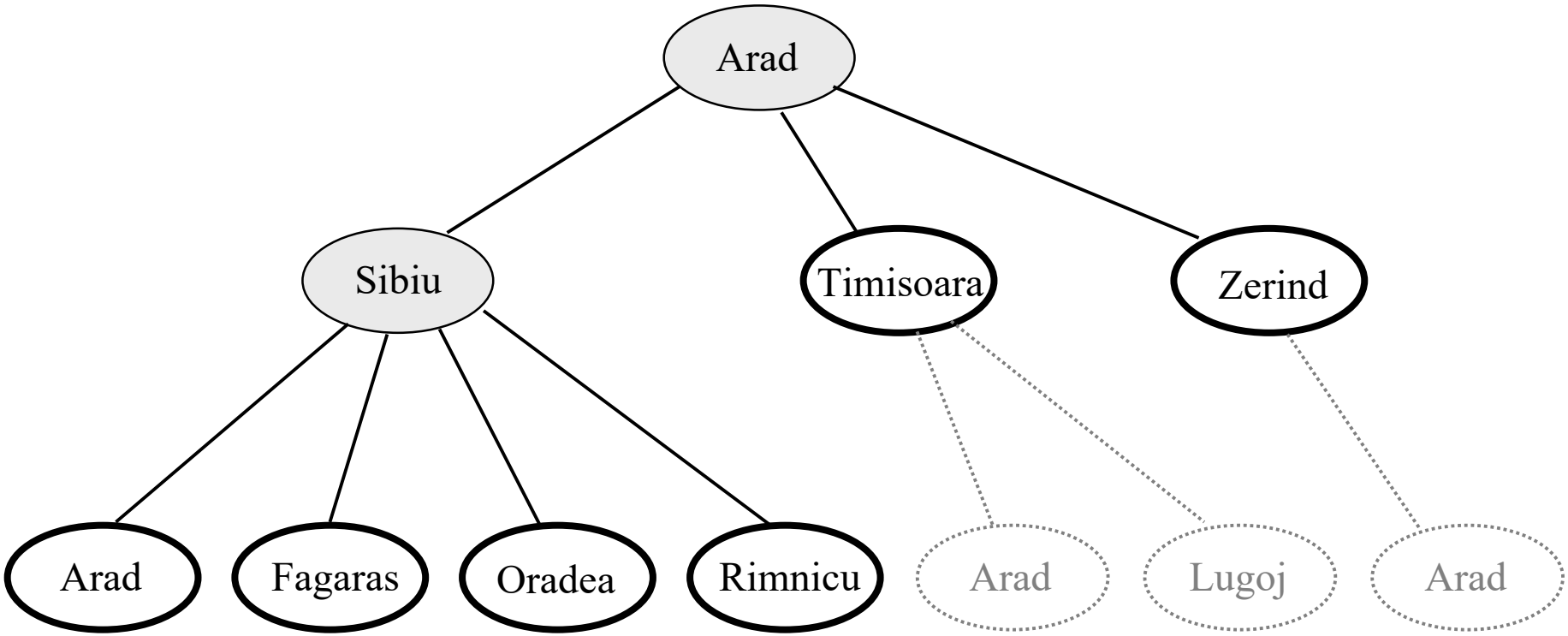
Estado inicial *In* (*Arad*)



Expande *In* (*Arad*)



Expande *In* (*Sibiu*)



Estrutura dados de um nó

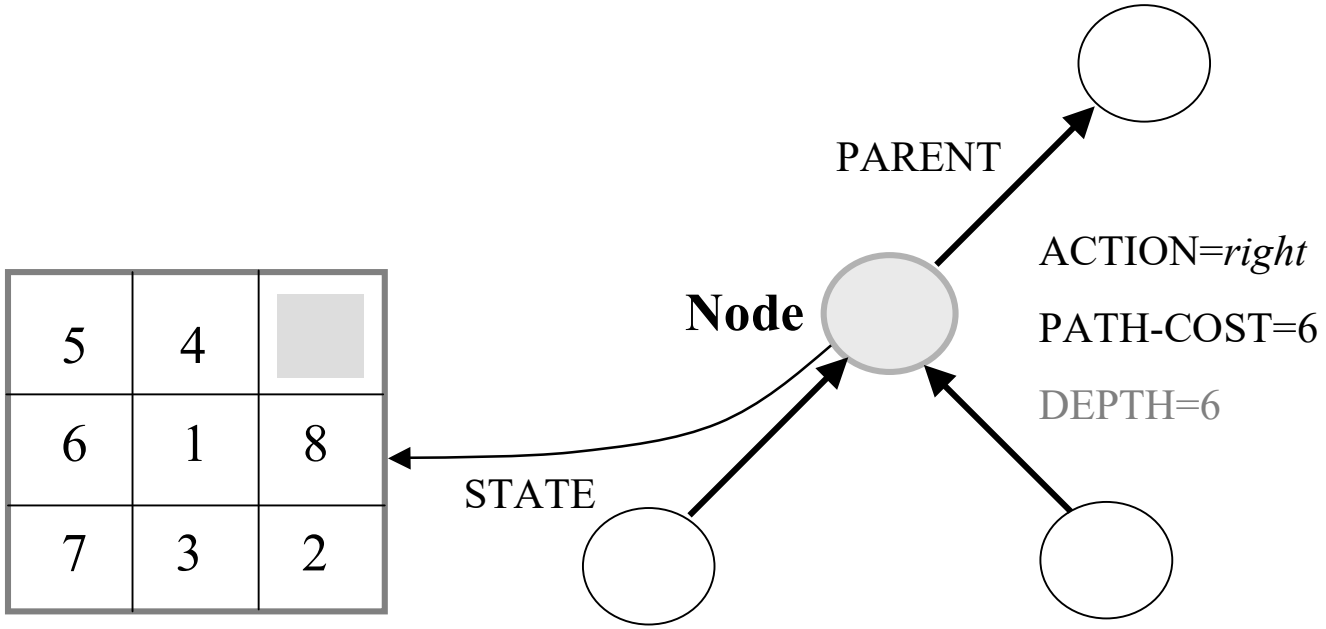
n.STATE	estado do espaço de estado correspondente nó *n*
n.PARENT	nó da árvore de busca que gerou o nó *n*
n.ACTION	ação aplicada ao pai do nó *n*
n.PATH-COST	custo, denotado por $g(n)$, do caminho da raiz até *n*

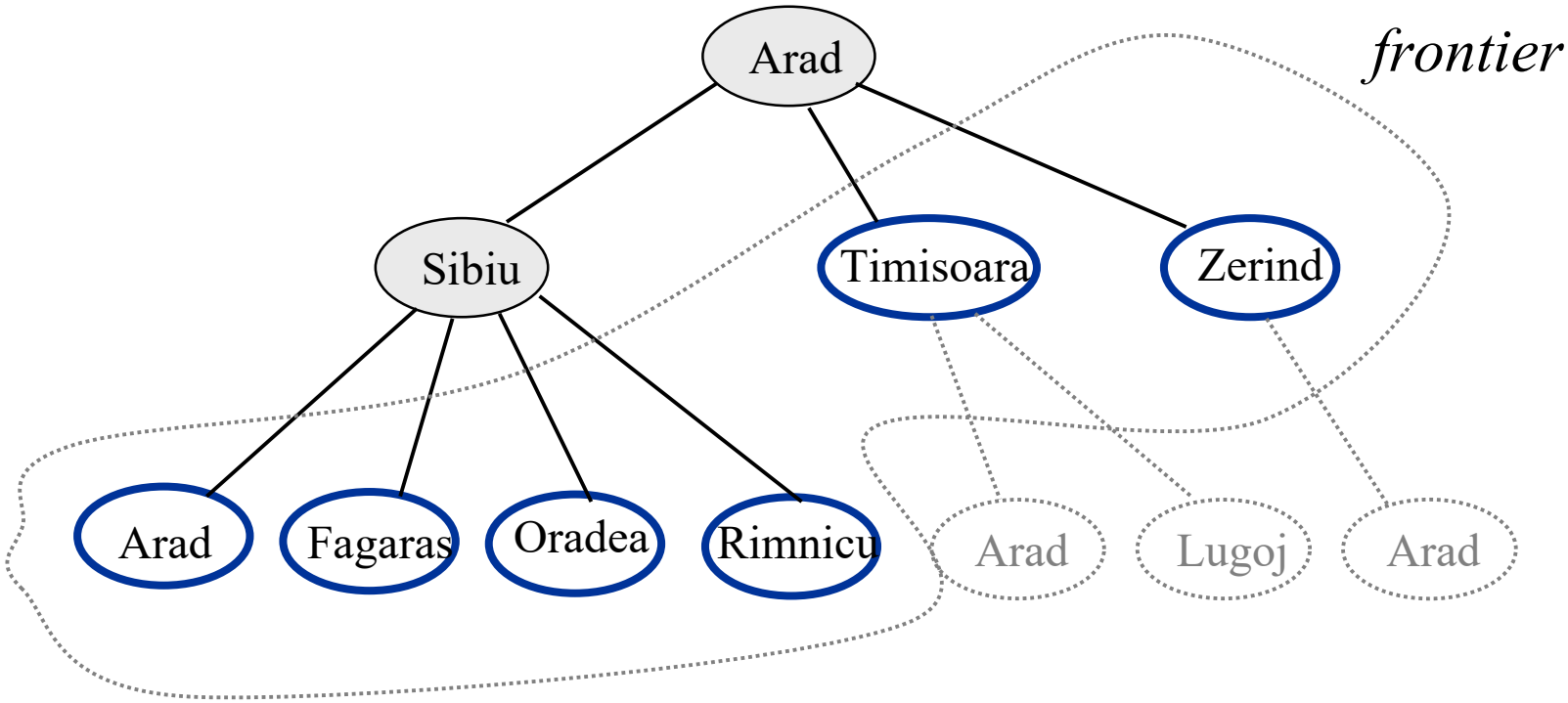
Geração dos componentes de um nó filho

function CHILD_NODE(*problem*, *parent*, *action*) **returns** a node

  return a node with
    STATE = *problem*.RESULT(*parent*.STATE, *action*)
    PARENT = *parent*
    ACTION = *action*
    PATH-COST = *parent*.PATH-COST + *problem*.STEP-COST(*parent*.STATE, *action*)

Nós e estruturas de dados





frontier: nós não expandidos (folhas)

representação fronteira: conjunto/lista de nós

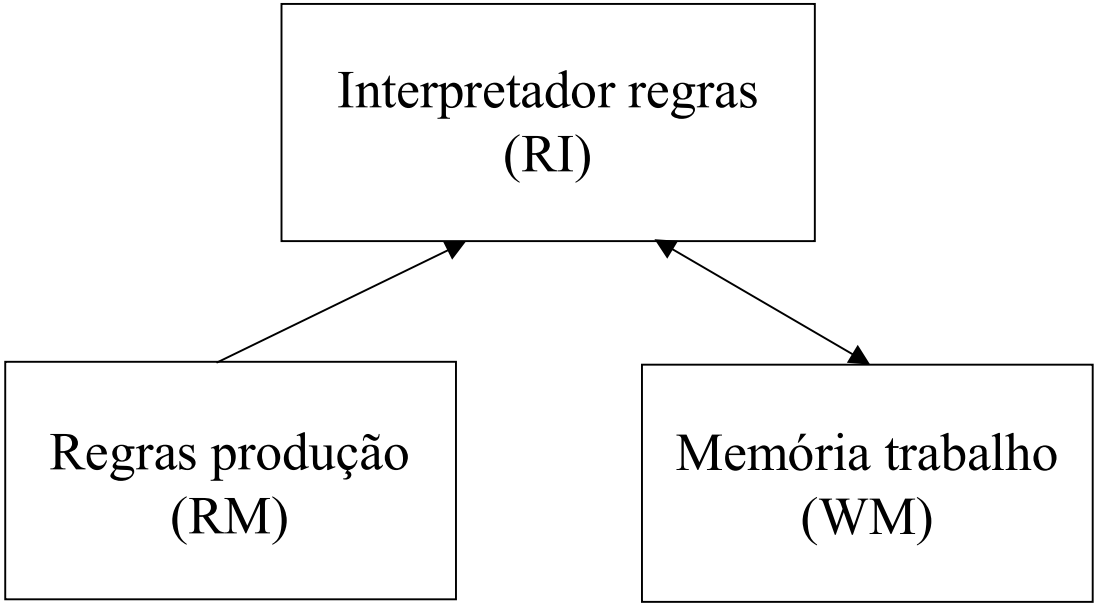
Algoritmo de busca em grafos

function GRAPH_SEARCH (*problem*) **returns** a solution or failure

frontier ← a node with STATE = *problem*.INITIAL-STATE
explored ← an empty set

loop do
  if EMPTY?(*frontier*) **then return** failure
  node ← POP (*frontier*) /* chooses a node in *frontier* */
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  for each *action* in *problem*.ACTIONS(*node*.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
    frontier ← INSERT (*child*, *frontier*)

Exemplo: sistemas de produção



Memória de trabalho (WM): dados representam estado (*state*) corrente

Regras de produção (RM): conjunto de regras na forma **se** A **então** B

se < *condição na* WM> **então** <*ação*>

Interpretador de regras (RI): módulo de controle que executa operações de comparação (*rule matching*) para determinar a regra a ser ativada em um ciclo de processamento.

RULE_MATCH (*state*, *rules*)

function PRODUCTION_SYSTEM (*state*) **returns** a solution or failure
persistent: *state*: some description of the current state
rules: a set of condition-action rules
action: most recent action, initially none

if WM satisfies a termination condition **then return** solution
WM ← *state*
rule ← RULE_MATCH(*state*, *rules*)
if *rule* = fail **then return** failure
action ← *rule*.ACTION
state ← UPDATE_STATE(*state*, *action*)

- Interpretador de regras (RI)
  - executa ciclo *match and act*
    - seleção de regras (*select*): determina as regras e itens da WM que serão considerados no ciclo
    - comparação (*matching*): itens da WM são comparados com regras da RM para determinar regras ativas (*pattern matching*)
    - resolução de conflitos (*scheduling*, *filter*): escolhe qual regra entre as ativas que será disparada
    - ação (*execution*, *action*) regra escolhida é *disparada* (executada)

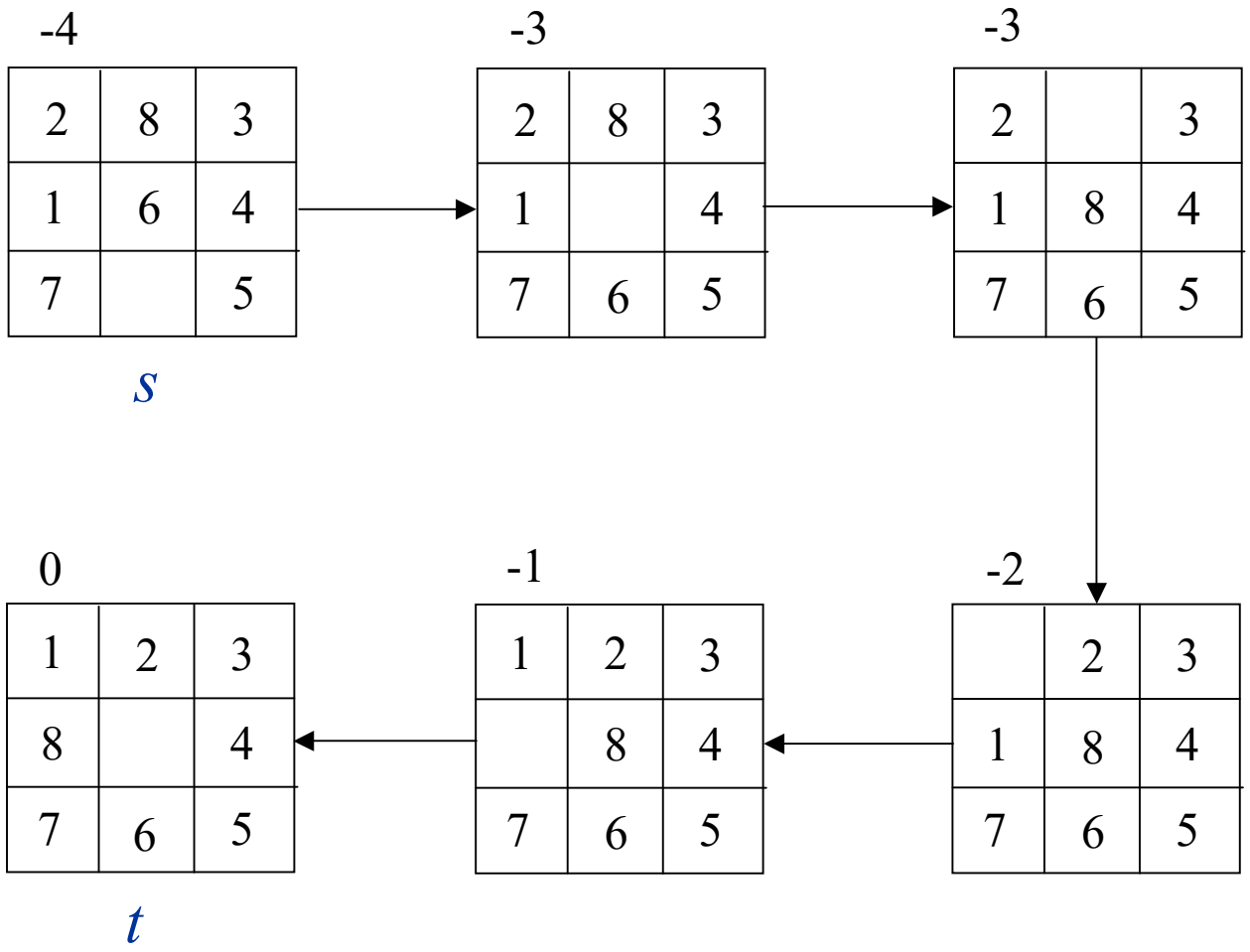
Estratégias de seleção de regras

- estratégias de controle
  - irrevogável: regra escolhida é aplicada, sem reconsiderações posteriores; exemplo: *hill-climbing*
  - tentativa: regra escolhida é aplicada, mas é possível voltar a este ponto do ciclo posteriormente para aplicar uma outra regra; exemplo busca em grafos com *backtracking*.

Sistema de produção

- funciona como um processo (implícito) de busca
- estratégia seleção de regras insuficiente para escolher a regra *ótima* em cada ciclo

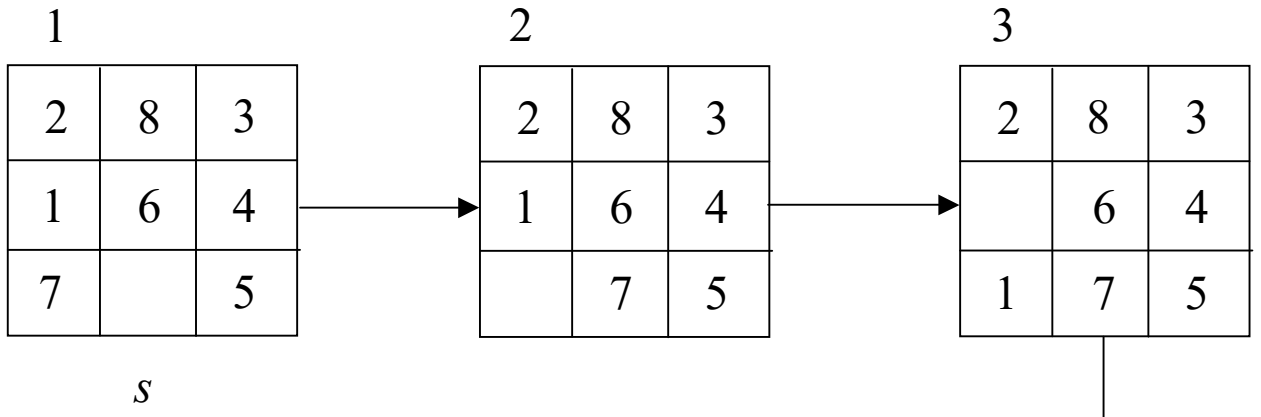
Hill-climbing



$f(x)$ = número de peças fora do lugar correto

x : estado s : estado inicial t : meta

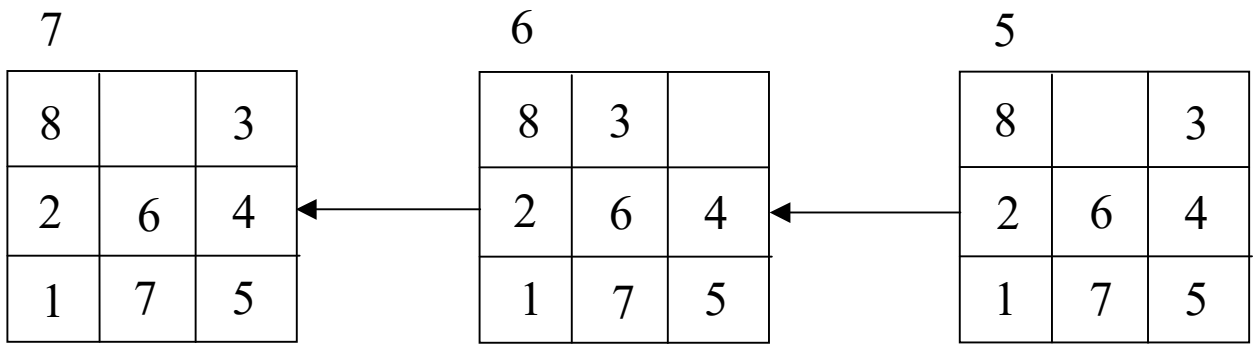
Backtracking



×

Este estado já ocorreu antes. Voltar para 5

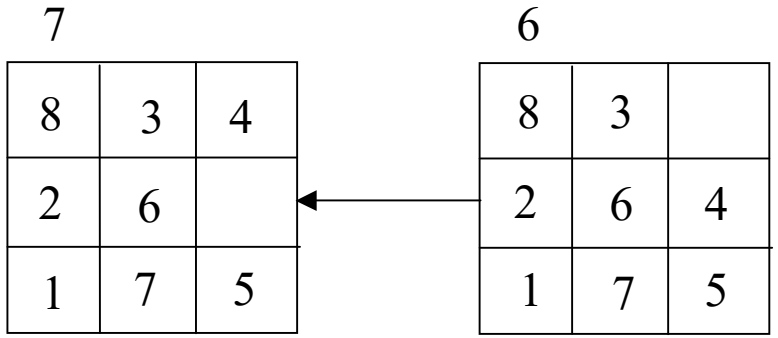
×



II

×

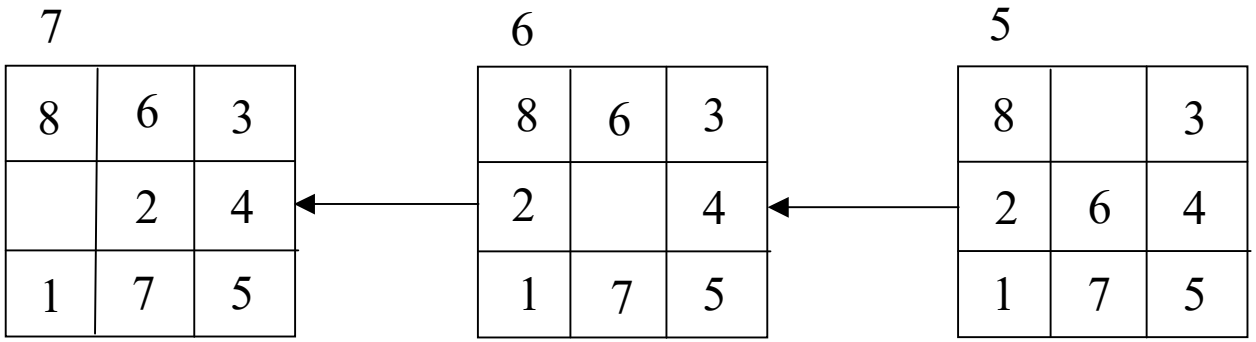
Este estado já ocorreu antes. Voltar para 6



III

×

Aplicamos todas regras a 6 e não atingimos meta. Voltar para 5.



IV

×

Aplicamos todas regras e a meta não foi atingida, etc….

Estratégias de seleção de regras

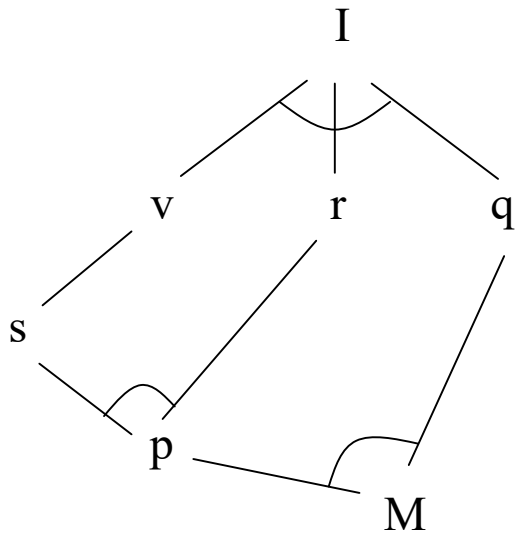
- data-driven: em cada ciclo, os antecedentes das regras na RM são examinados para determinar quais regras são satisfeitas pelo conteúdo da WM (*forward chaining*, encadeamento direto)
- goal-driven: em cada ciclo, os consequentes das regras em RM são examinados para verificar se algum deles contém a meta. A seguir o sistema busca os fatos necessários pelo o antecedente. (*backward chaining* encadeamento reverso)

Exemplo de encadeamento direto

RM

1. $p \wedge q \rightarrow M$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow q$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $I \rightarrow v \wedge r \wedge q$

Ciclo	WM	Regras Ativas	Regra Disparada
0	I	6	6
1	I, v, r, q	6,5	5
2	I, v, r, q, s	6,5,2	2
3	I, v, r, q, s, p	6,5,2,1	1
4	I, v, r, q, s, p, M	6,5,2,1	parar

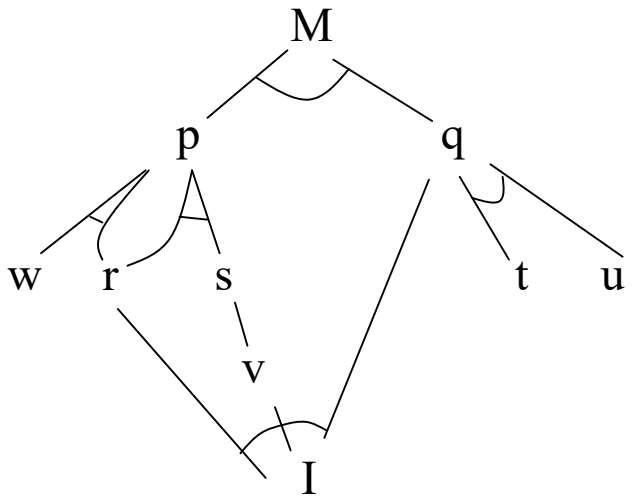


Encadeamento reverso

RM

1. $p \wedge q \rightarrow M$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow p$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $I \rightarrow v \wedge r \wedge q$

Ciclo	WM	Regras Ativas	Regra Disparada
0	M	1	1
1	M, p, q	1,2,3,4	2
2	M, p, q, r, s	1,2,3,4,5	3
3	M, p, q, r, s, w	1,2,3,4,5	4
4	M, p, q, r, s, w, t, u	1,2,3,4,5	5
5	M, p, q, r, s, w, t, u, v	1,2,3,4,5, 6	6
6	M, p, q, r, s, w, t, u, v, I	1,2,3,4,5, 6	parar



Observação

Este material refere-se às notas de aula do curso EA 072 Inteligência Artificial em Aplicações Industriais da Faculdade de Engenharia Elétrica e de Computação da Unicamp. Não substitui o livro texto, as referências recomendadas e nem as aulas expositivas. Este material não pode ser reproduzido sem autorização prévia dos autores. Quando autorizado, seu uso é exclusivo para atividades de ensino e pesquisa em instituições sem fins lucrativos.