Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial
Aula 4 - Busca não-informada

Prof. José Reinaldo Silva

reinaldo@usp.br

Correções no plano de aula

COURSE PLAN TEMPLATE			
COURSE REFERENCE:	PMR3510	SUBJECT:	Inteligência Artificial Clássica
COURSE TITLE:	Inteligência Artificial		
COURSE OBJECTIVE:	Posicionamento da IA como atributo lógico e formal para automação		
LEVEL:	graduação	EXAM?	

Lesson Plan Ref:	Lesson Title:	Topic:
Aula 1 (18/08)	Introdução - Evolução da IA	Chap. 1
Aula 2 (25/08)	Resolução de Problemas em IA, Agente inteligente	Chap. 2
Aula 3 (1/09)	Métodos de Busca e algoritmos	Chap. 2
Aula 4 (8/09)	Busca não informada	Chap. 3
Aula 5 (15/09)	Busca informada, heurísticas	Chap. 3
Aula 6 (22/09)	Busca informada, exercícios	Chap. 4
Aula 7 (29/09)	Regras de produção, sistemas especialistas	Chap. 4
Aula 8 (6/10)	Aplicações dos algoritmos de busca em automação	Chap. 4
Aula 9 (13/10)	Aplicações em jogos	Chap. 6
Aula 10 (20/10)	Agentes racionais (knowlege agents)	Chap. 7
Aula 11 (27/10)	Métodos de Inferência e Lógica	Chap. 9
Aula 12 (03/11)	Representação de conhecimento	Chap 10
Aula 13 (10/11)	Planejamento Automático	Chap. 11
Aula 14 (17/11)	O modelo STRIPS	Chap 11
Aula 15 (24/12)	Explorando o STRIPS: o mundo de blocos	Chap. 11 + Nilsson
Aula 16(1/12)	Developing planners e exercício programa	
Aula 17 (8/12)	Aplicações; dúvidas sobre o EP	

Problem-solving: methods or heuristics

George Pólya
1887-1985

PONTES (2019)

HOLOS
ISSN 1807 - 1600

MÉTODO DE POLYA PARA RESOLUÇÃO DE PROBLEMAS MATEMÁTICOS: UMA PROPOSTA METODOLÓGICA PARA O ENSINO E APRENDIZAGEM DE MATEMÁTICA NA EDUCAÇÃO BÁSICA

E.A.S.PONTES*
Instituto Federal de Alagoas
edelpontes@gmail.com*

Artigo submetido em 20/12/2017 e aceito em 24/06/2019

DOI: 10.15628/holos.2019.6703

RESUMO

No mundo contemporâneo diversas pesquisas são realizadas em busca de uma solução eficaz no processo ensino e aprendizagem de matemática, tendo como foco as suas novas técnicas da educação matemática. Este trabalho tem como objetivo apresentar uma proposta metodológica para o ensino e aprendizagem de matemática na educação básica, através da resolução de problemas utilizando o método de Polya. O método de Polya consiste em três etapas: Compreender o problema, Designar um plano, Executar o plano e Retrospecto do problema. Metodologicamente serão apresentados três problemas matemáticos cuja resolução seguirá o método de Polya. Está sugestão, Resolução de Problemas através do método de Polya, como prática educacional no processo de ensino e aprendizagem de matemática possibilita ao professor facilitador e ao aluno aprendiz desenvolver novas habilidades no intuito de fortalecer o pensamento crítico e o raciocínio lógico.

PALAVRAS-CHAVE: Ensino e aprendizagem de matemática, método de Polya, Resolução de problemas.

https://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/6703/pdf

IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 13, NO. 6, NOVEMBER/DECEMBER 2001 913

Structured Development of Problem Solving Methods

Dieter Fensel and Enrico Motta

Abstract—Problem solving methods (PSMs) describe the reasoning components of knowledge-based systems as patterns of behavior that can be reused across applications. While the availability of extensive problem solving method libraries and the emerging consensus on problem solving method specification languages indicate the maturity of the field, a number of important research issues are still open. In particular, very little progress has been achieved on foundational and methodological issues. Hence, despite the number of libraries which have been developed, it is still not clear what organization principles should be adopted to construct truly comprehensive libraries, covering large numbers of applications and encompassing both task-specific and task-independent problem solving methods. In this paper, we address these "fundamental" issues and present a comprehensive and detailed framework for characterizing problem solving methods and their development process. In particular, we suggest that PSM development consists of introducing assumptions and commitments along a three-dimensional space defined in terms of *problem-solving strategy, task commitments*, and *domain (knowledge) assumptions*. Individual moves through this space can be formally described by means of *adapters*. In the paper, we illustrate our approach and argue that our architecture provides answers to three fundamental problems related to research in problem solving methods: 1) what is the epistemological structure and what are the modeling primitives of PSMs? 2) how can we model the PSM development process? and 3) how can we develop and organize truly comprehensive and manageable libraries of problem solving methods?

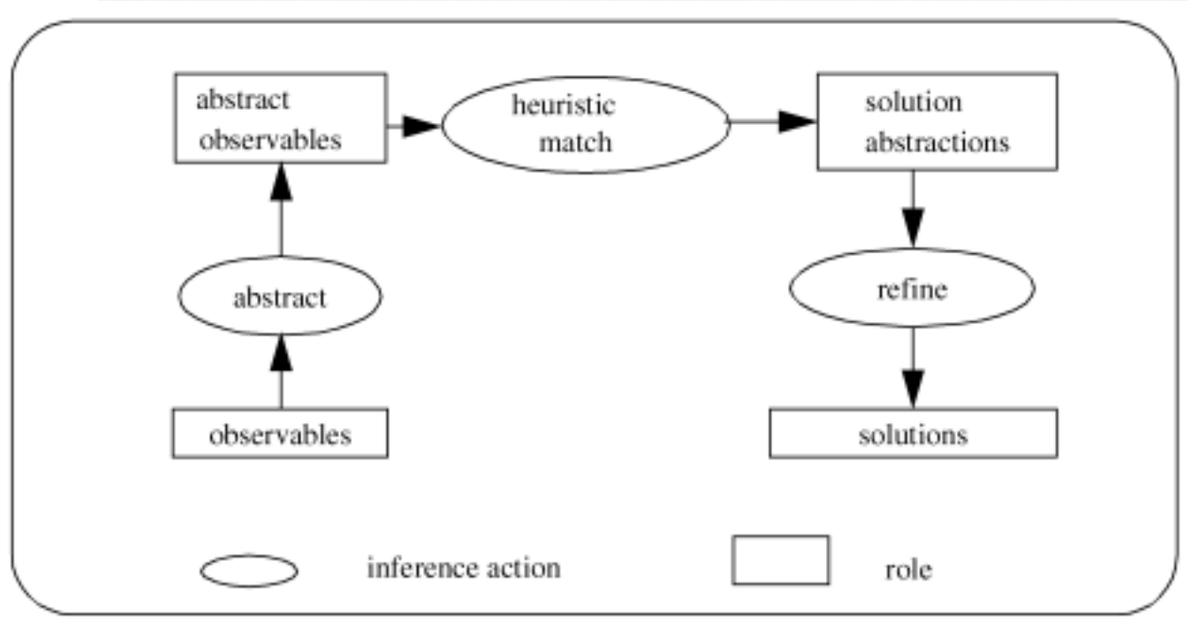
Index Terms—Knowledge modeling, problem-solving methods, ontologies, knowledge engineering, software engineering, formal languages.

1 Introduction

Problem solving methods (PSMs) describe the reasoning components of knowledge-based systems as patterns of behavior that can be reused across applications. For instance, the problem solving method *Propose & Revise* ([56], [92]) provides a generic reasoning pattern, characterized by iterative sequences of model "extension" and "revision," which can be reused when solving—for instance—scheduling [81] or design [56] problems. Problem solving methods define an important technology for supporting structured development approaches in knowledge engineering: they 1) provide strong model-based frameworks in which to carry out knowledge acquisition ([55], [87]) and 2) support the rapid development of robust

number of papers describing the state of the art in problem-solving method research can be found in [8].

So far, most of the research effort has focused on identifying and defining specific classes of problem solving methods. As a result, several problem solving method libraries are now available ([11], [55], [19], [71], [6], [12], [67], [64], [59], [76]) and a number of problem-solving method specification languages have been proposed, ranging from informal notations (e.g., CML [74]) to formal modeling languages—see [41] and [28] for comprehensive surveys. Some of these libraries provide executable reasoning components (for example, [59]), others (e.g., the CommonKADS library [12]) provide only conceptual models of such components similar to design

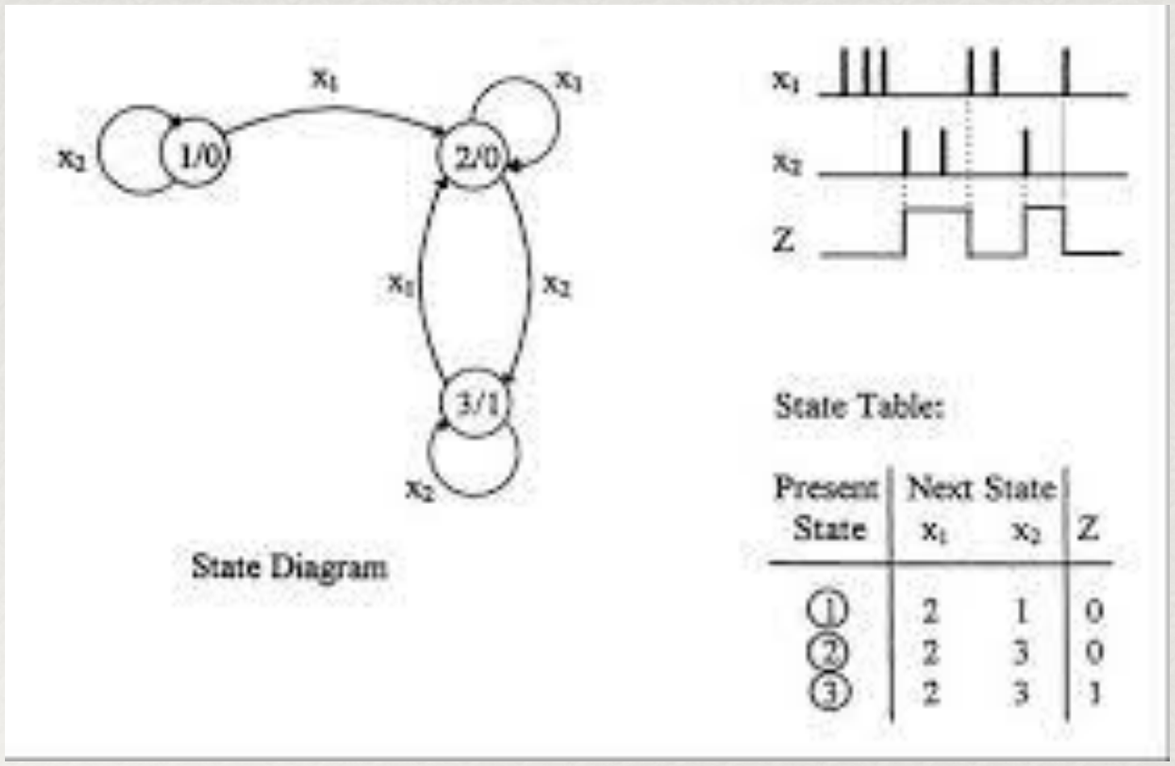


Problem-solving: a synthesis

- Let D be an application domain
- Let F be a dynamic system in this domain
- Let S be the state space for F

What would be a problem to be solved in (D, F)?

Um sistema dinâmico muda de estado quando acontece algum evento ou ação, seja interna ou originada no domínio D.



Assim, possíveis problemas em (D, F) seriam:

- Saber se o sistema atinge um estado s.
- Programar o sistema para sair de um estado inicial e atingir um estado s.
- Evitar que um sistema atinja um estado s.
- Monitorar a evolução do sistema.

Planning ahead...

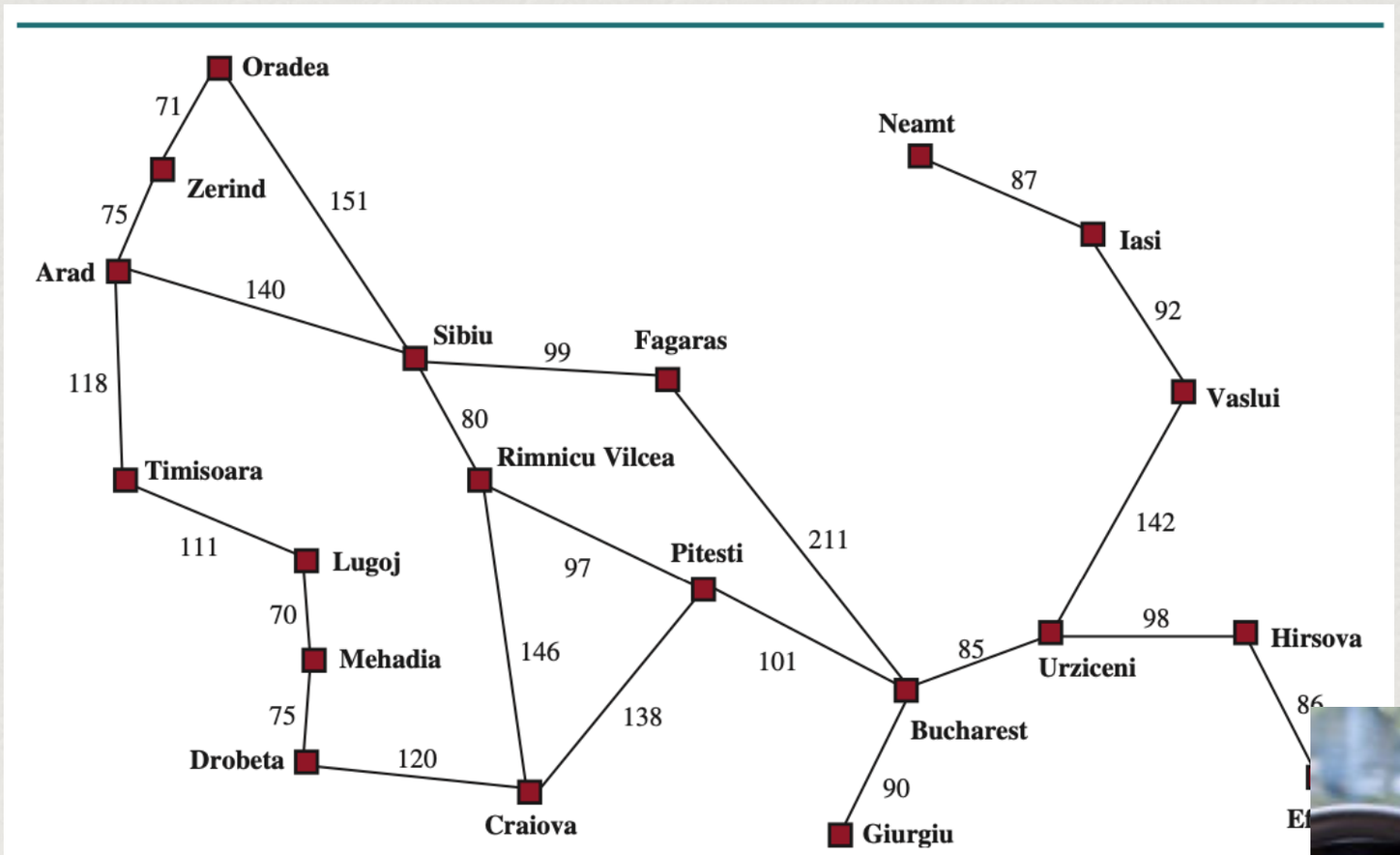


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Agentes baseados em objetivos (Goal-based agents)

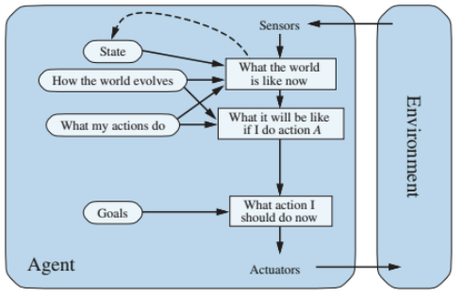
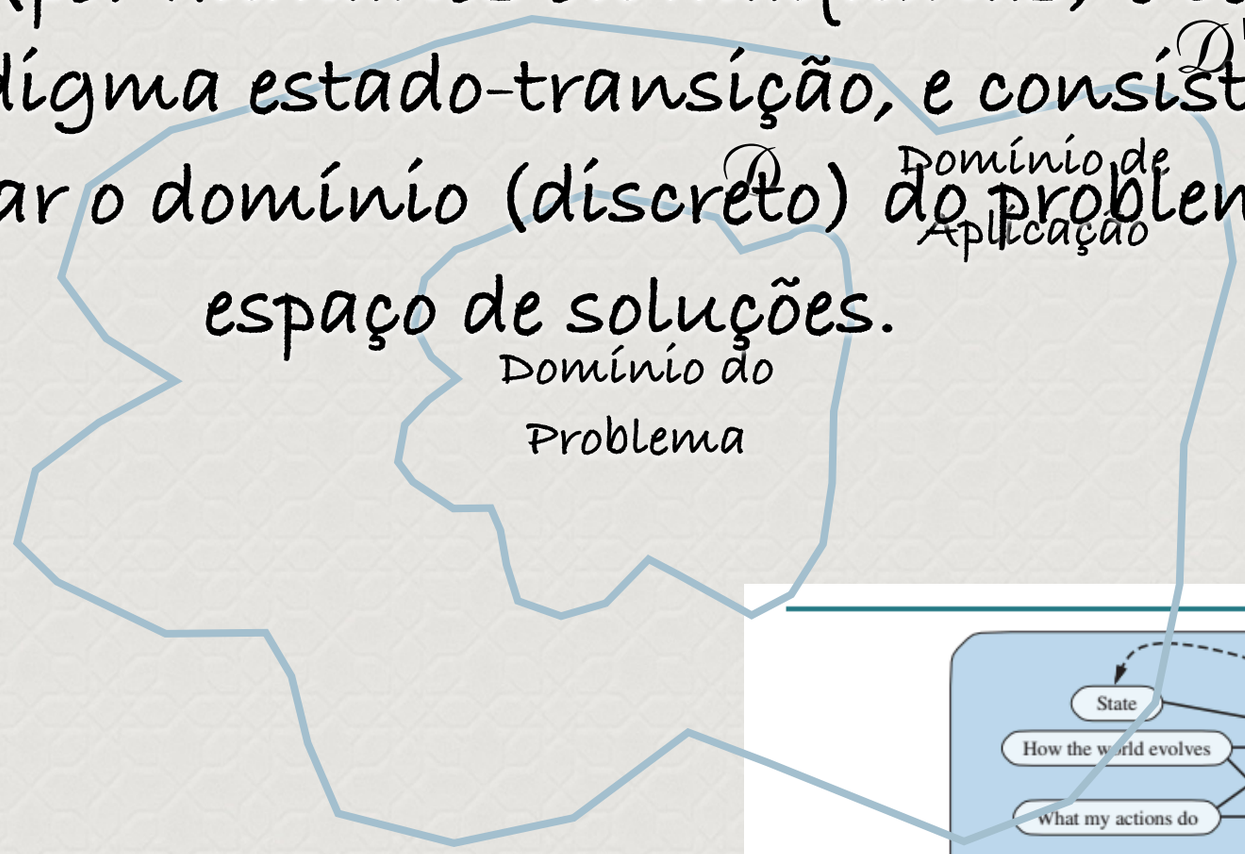


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Inicialmente, a forma da abordagem para problemas (por humanos ou máquinas) é centrada no paradigma estado-transição, e consiste em identificar o domínio (discreto) do problema e o espaço de soluções.



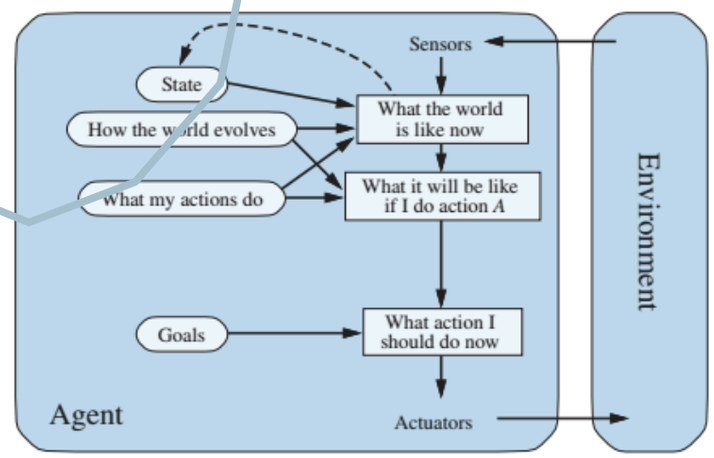


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Genericamente vamos assumir a hipótese de que "resolver um problema" significa: levando em conta o domínio de aplicação e o domínio do problema achar uma sequência de ações que leve da condição (estado) inicial ao objetivo (estado final), a situação (estado) em que "o problema está resolvido".

Um procedimento para resolução de problemas deve começar pela identificação do espaço de estados do problema. O estado inicial é identificado neste espaço. Por exemplo, queremos uma rota saindo de Arad.

No caso de agentes baseados em objetivos, temos que identificar qual é este objetivo. Por exemplo, o objetivo pode ser "ir para Bucharest".

Agentes baseados em objetivos (Goal-based agents)

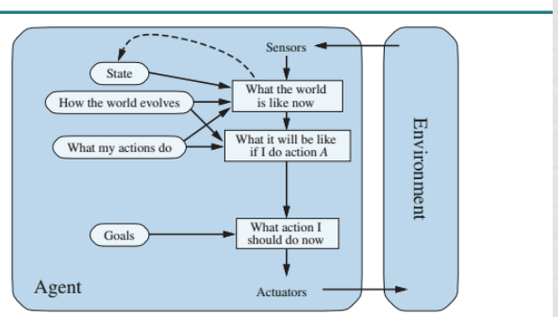
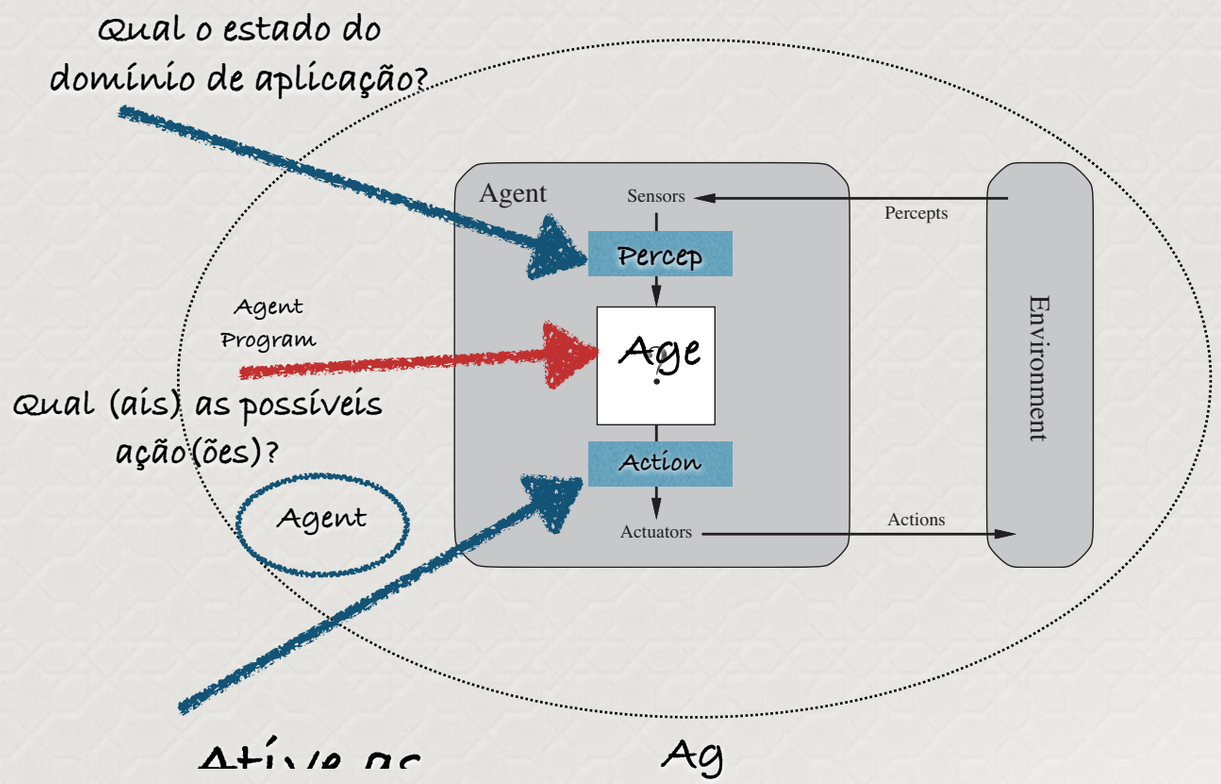


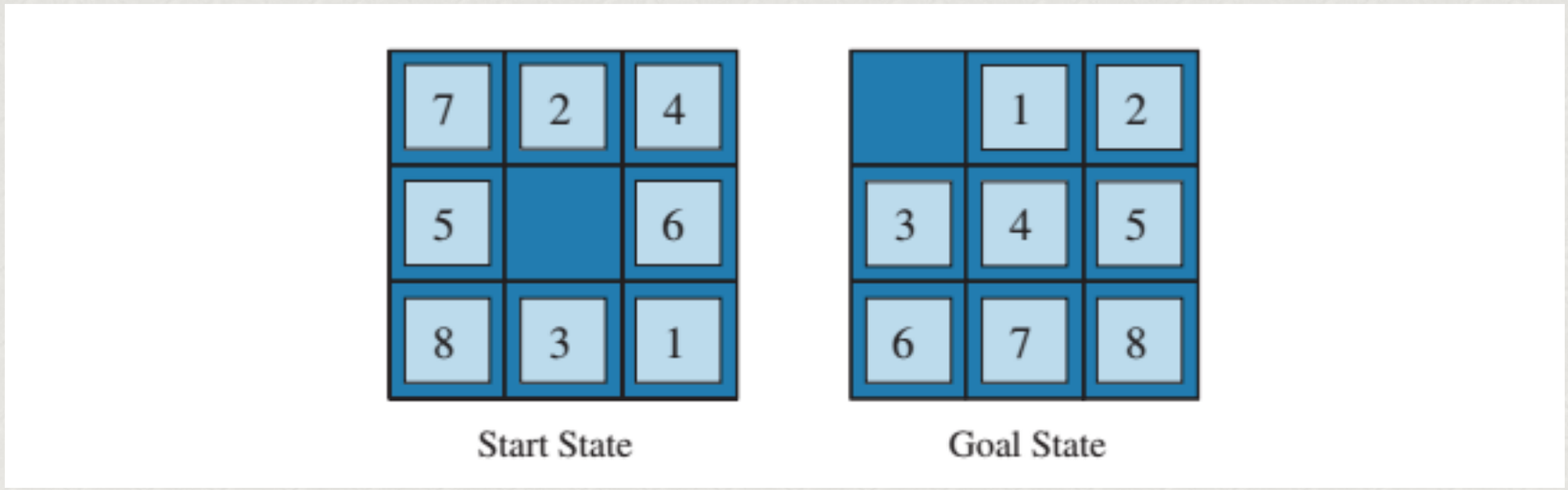
Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Em uma primeira aproximação, resolver um problema, pode ser apenas achar um caminho entre o estado inicial e o estado final, o que pode ser implementado por um algoritmo de busca.

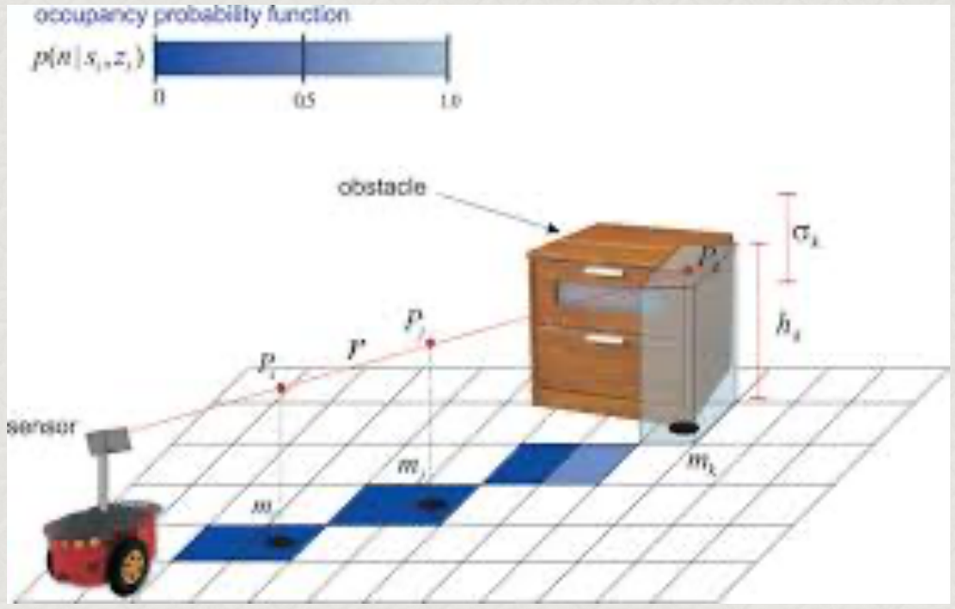


Modelo do "Grid problem" em games

Uma adaptação simples do "grid problem" é o jogo de tiles, mostrado abaixo, que combina a estratégia orientada a objetivos com os movimentos em "grid".



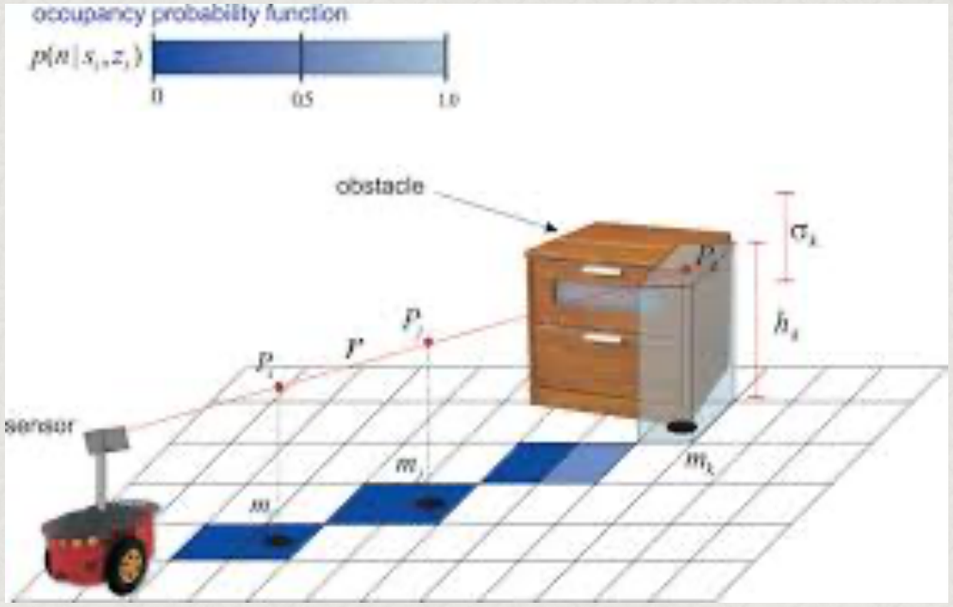
Um algoritmo conhecido como "Occupancy Grid" foi proposto em 1985 para orientar o movimento de robôs móveis. Este algoritmo foi proposto em Carnegie Mellon por Hans Peter Moravec e Alberto Elfes.



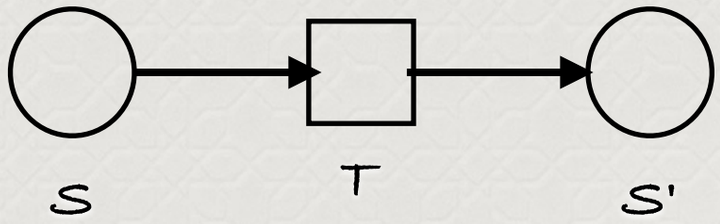
Hans Peter Moravec

Alberto Elfes

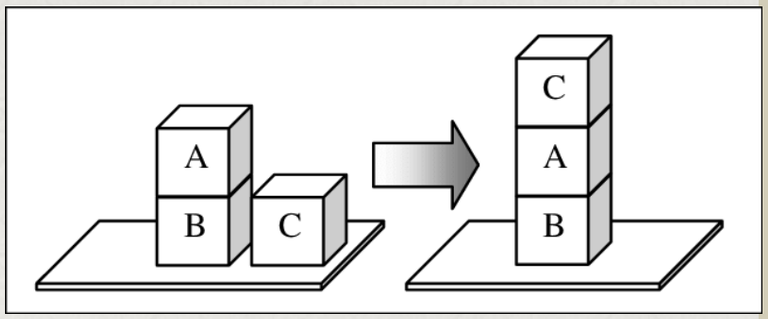
Porque o "grid problem" é tão importante (além do fato de ter inspirado grandes aplicações)? Por que nesse caso a "transição" é uma decorrência imediata da configuração (vizinhança entre os elementos do sistema).

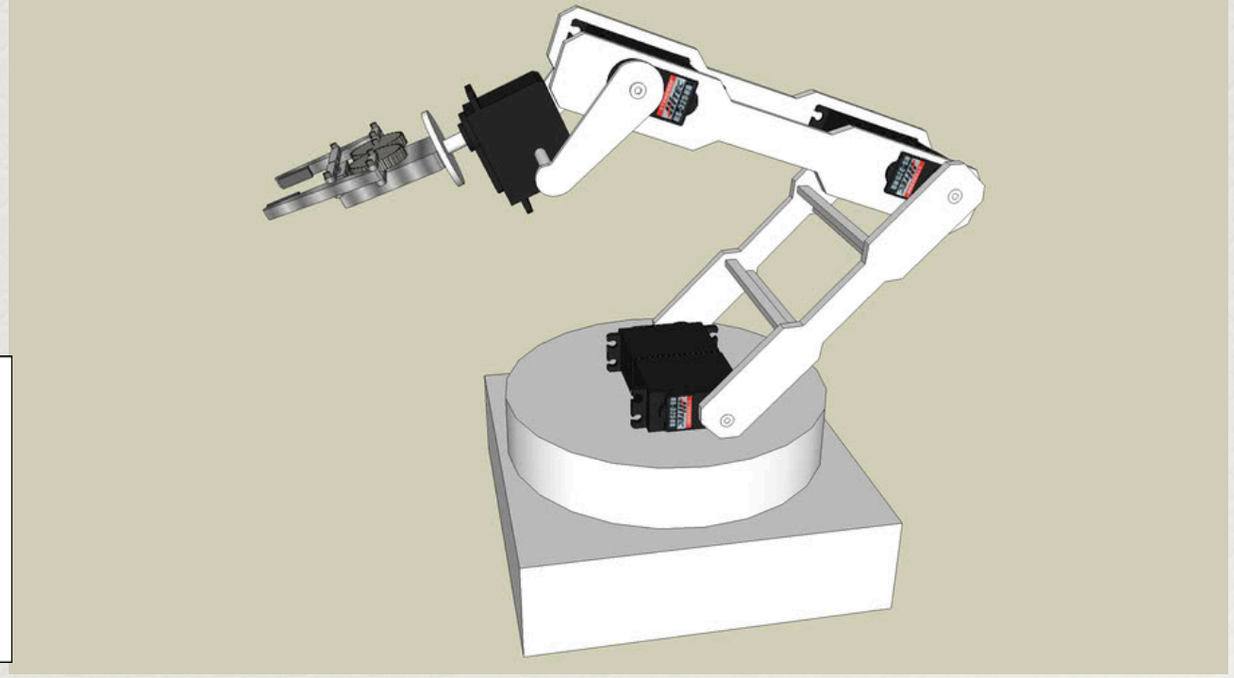


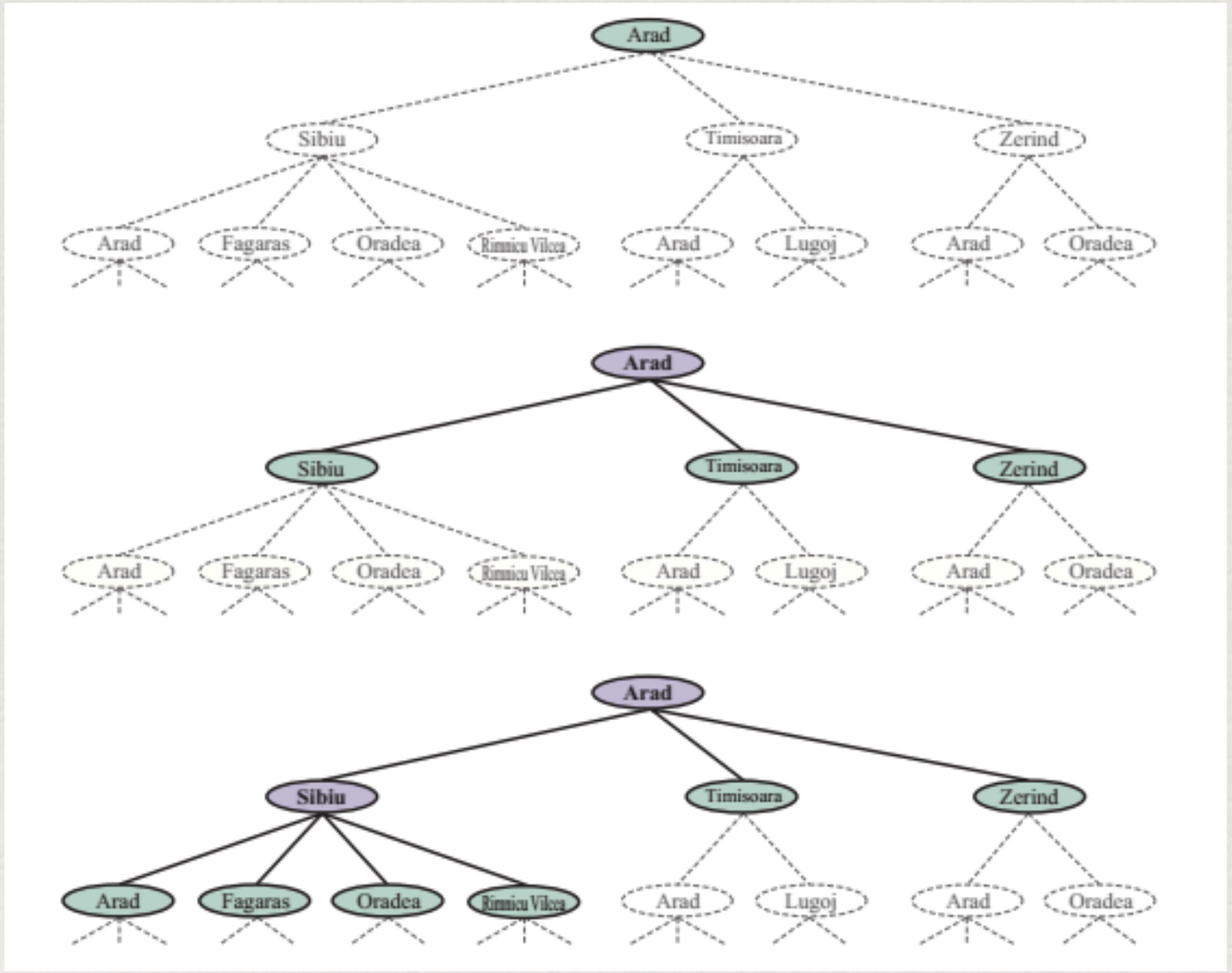
Mars Pathfinder



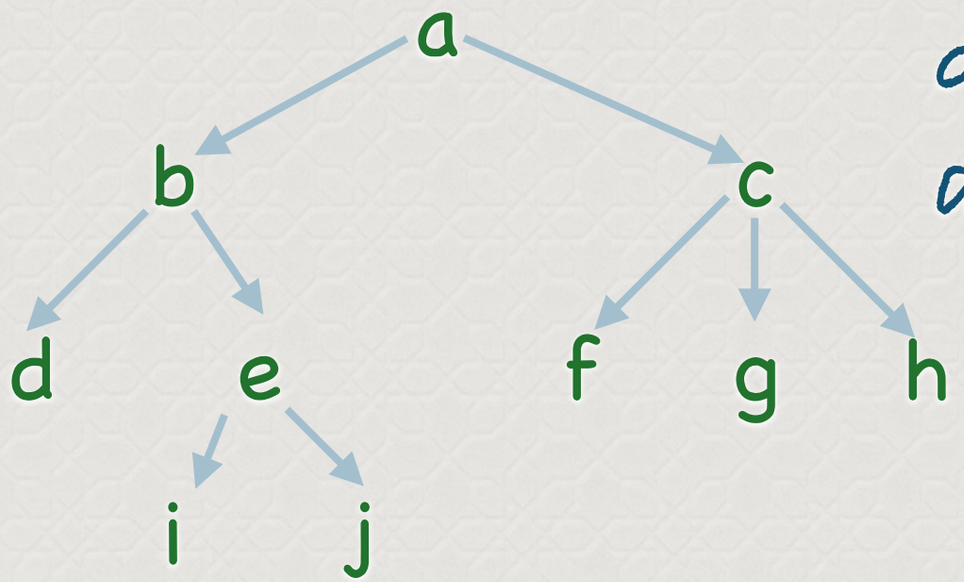
Em um ambiente com vários elementos, com diferentes possibilidades de arranjos, e restrições de movimento, o estado seria descrito de forma mais elaborada e a transição não seria tão imediata.







Expandindo a árvore de busca

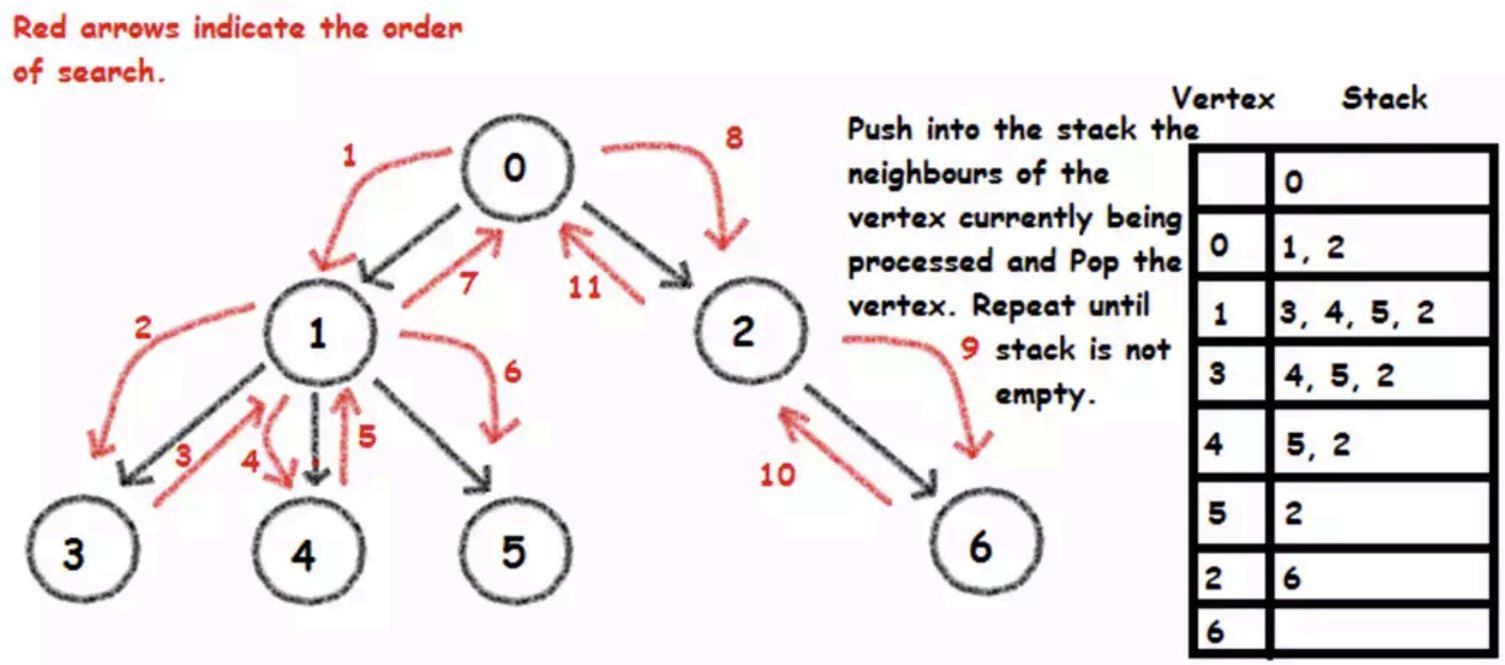


A estrutura não-linear mais simples é a árvore, sobre a qual se montam os algoritmos mais eficientes.

[a | [b | [d, [e | [i, j]]]], [c | [f, g, h]]]

A busca "cega" consiste em varrer toda a árvore de busca até encontrar uma solução, ou esgotar as possibilidades de geração de novos estados (para um espaço de busca finito). A busca cega também é chamada de "busca não-informada", isto é, não existe na base de conhecimento do agente nenhuma informação que permita diferenciar as ações admissíveis a serem escolhidas. (Ter uma função custo associada ao resultado das ações seria uma opção).

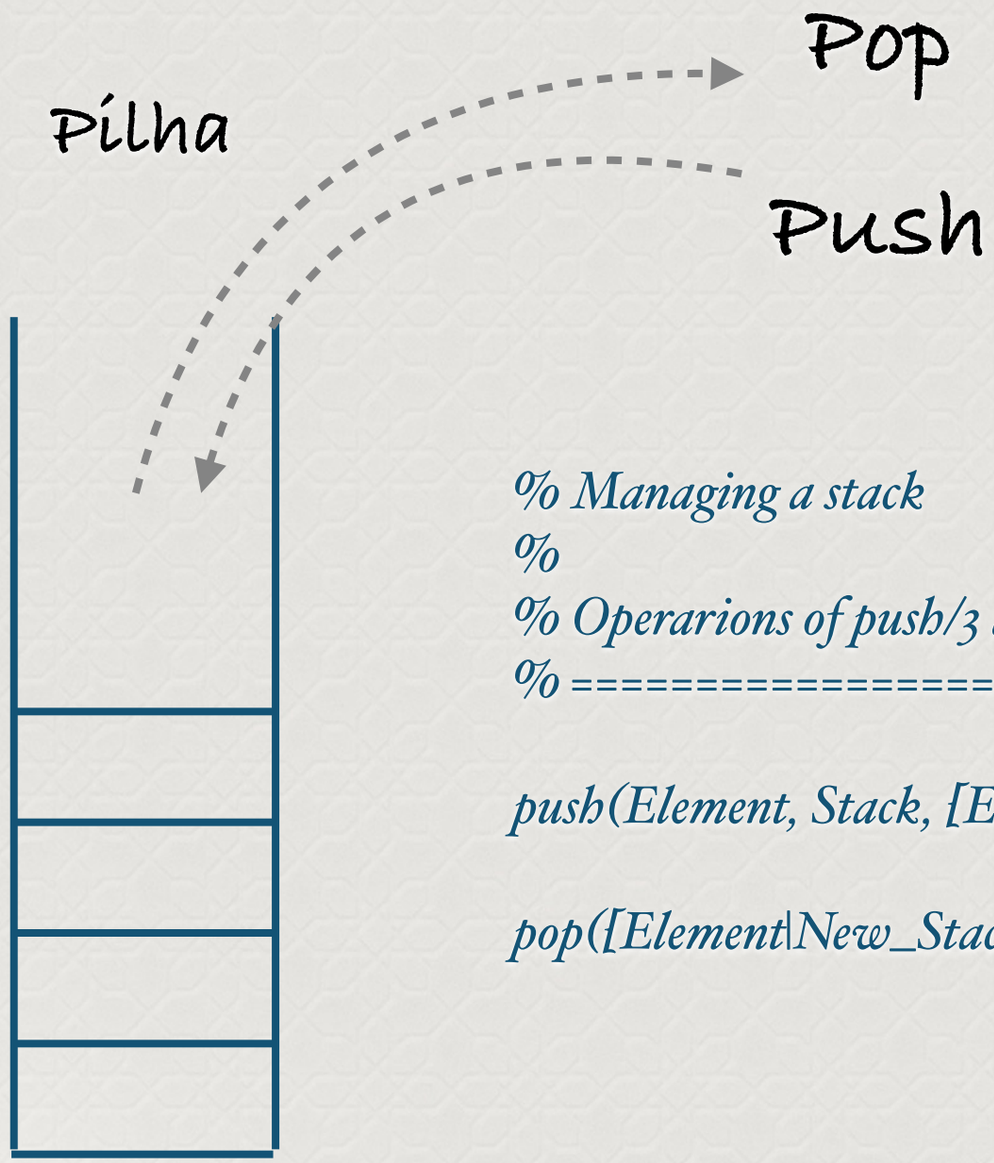
Busca em profundidade



Busca em profundidade (Depth-first Search, DFS)

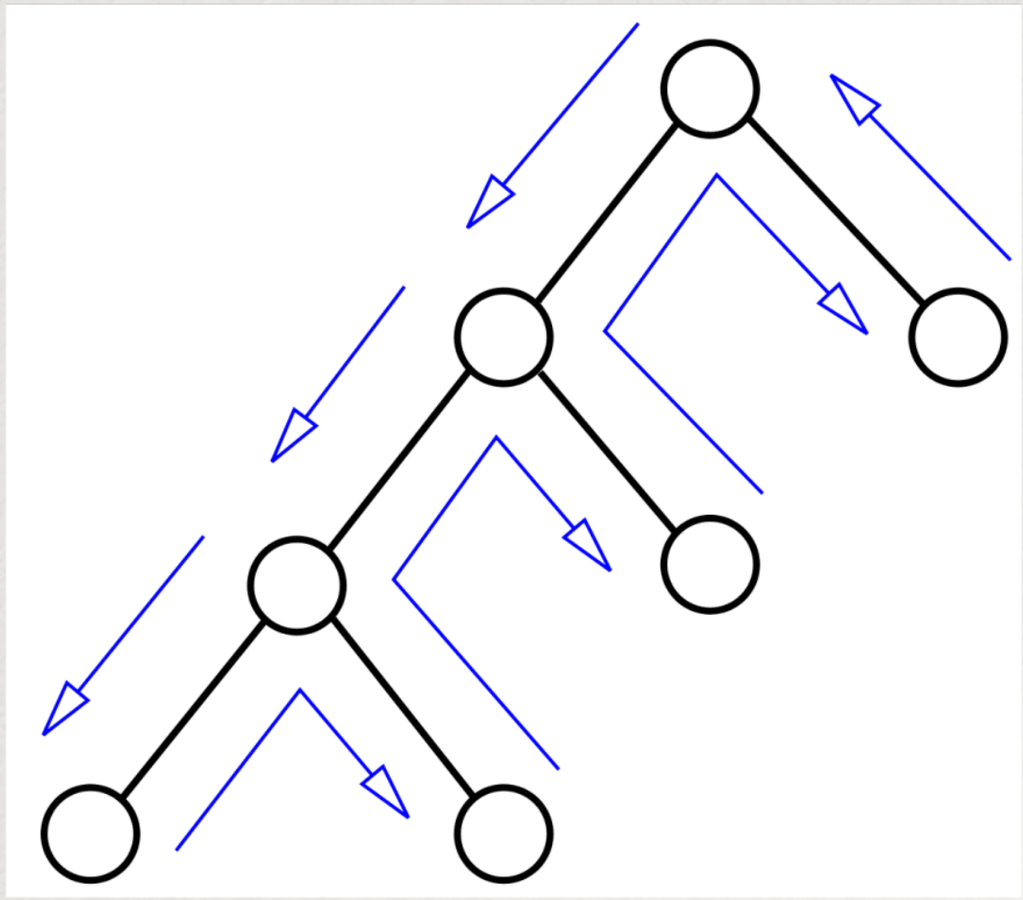
Pilha

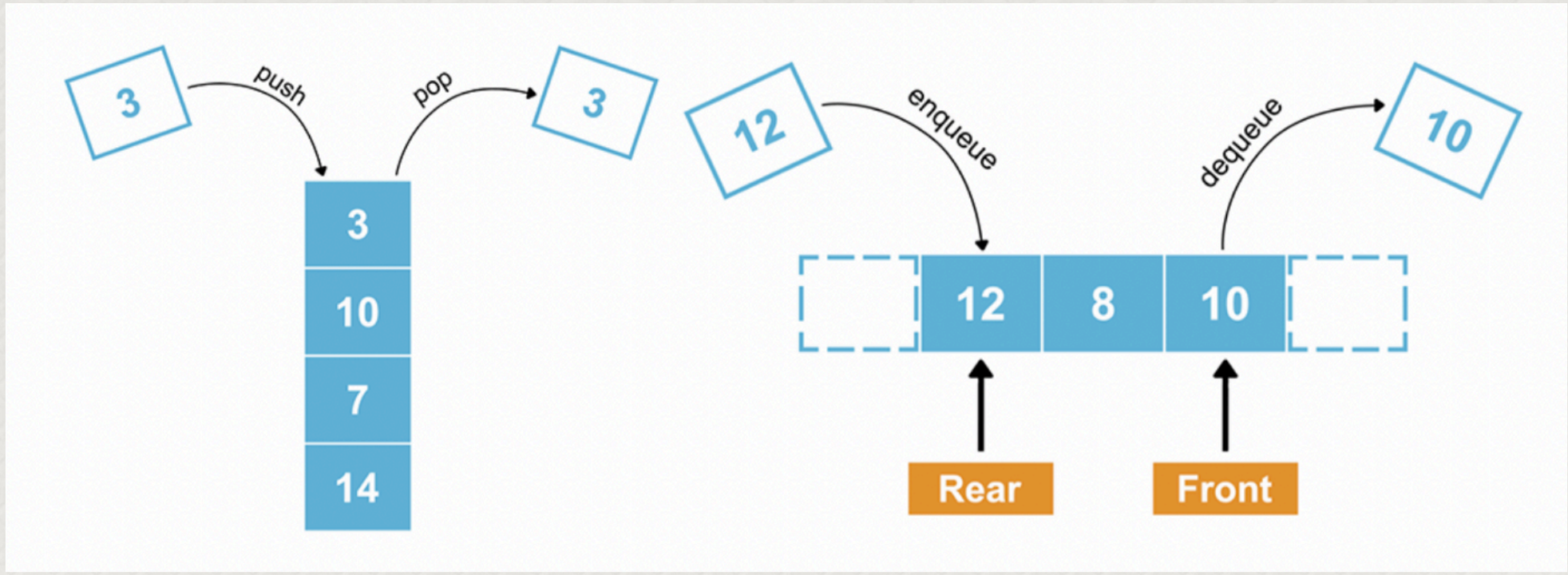
```
go(Start, Goal) :-
   empty_stack(Empty_been_list),
   stack(Start, Empty_been_list, Been_list),
   path(Start, Goal, Been_list).

% path implements a depth first search in PROLOG

% Current state = goal, print out been list
path(Goal, Goal, Been_list) :-
    reverse_print_stack(Been_list).

path(State, Goal, Been_list) :-
    mov(State, Next),
    % not(unsafe(Next)),
    not(member_stack(Next, Been_list)),
    stack(Next, Been_list, New_been_list),
    path(Next, Goal, New_been_list), !.

reverse_print_stack(S) :-
    empty_stack(S).
reverse_print_stack(S) :-
    stack(E, Rest, S),
    reverse_print_stack(Rest),
    write(E), nl.
```




```
% Managing a stack
%
% Operarions of push/3 and pop/3 over a stack implemented in a list
% ====================================

push(Element, Stack, [Element|Stack]).

pop([Element|New_Stack], Element, New_Stack).
```

O uso do “stack" é importante para o backtracking.





Stack

Queue

SWISH File Edit Examples Help

1 users online

Search

Program

```
% Managing a stack
%
% Operarions of push/3 and pop/3 over a stack implemented in a list
% ==================================

push(Element, Stack, [Element|Stack]).

pop([Element|New_Stack], Element, New_Stack).
```


pop([a,b,c,d],X,Z).

X = a,
Z = [b, c, d]

push(m,[a,b,c,d],Z).

Z = [m, a, b, c, d]

?-

```
push(m,[a,b,c,d],z).
```

Examples History Solutions table results Run!

Listas

[head | tail]

Elemento — Lista

[ ] - Lista vazia

```
push(Element, Stack, [Element | Stack]).

pop([Element | New_Stack], Element, New_Stack).
```

SWISH File Edit Examples Help

72 users online

Search

minimax22

Game Trees

Robert Laing

This tutorial has now been appended to https://swish.swi-prolog.org/p/Graphs1.swinb, partly because I've managed to get the same code base to handle and games and puzzles, and because for some reason I can't keep the same URL when I save with this file for some reason.

Please post any suggestions and comments to a discussion I started at https://swi-prolog.discourse.group/t/game-tree-tutorial-on-swish/921.

As with puzzles, I'm using conventions from Stanford University's General Game Playing course, translating rules written in Game Description Language into Prolog.

This is an initial stab at advancing from depth first search to iterative deepening.

I've kept the historical names -- minimax and alpha-beta -- though they assume two player games, when the same ideas can be applied from one player games (commonly called puzzles) to N player games.

I've tried to keep the code similar to the graph traversal tutorial, but one key change I've made is instead of storing the game tree as a list of `move(Parent, Label, Child)` clauses, I've expanded this to `move(RewardsList, Parent, Label, Child)`.

As far as I understand the jargon term dynamic programming, this is an example of it in that it involves lots of updating the values of nodes, something Prolog's *assign-once variables* make fairly laborious, I hope to gradually polish the spaghetti code I've written below do this into something more elegant.

```
6  % Returns a list of does(Player, Move) clauses in sequence to
7  % solve the puzzle using depth first search
8  solve_dfs(Moves) :-
9      findinit(Start),
10     heuristic(Start, Rewards), % Assuming start not terminal
11     depthfirst([move(Rewards, start, noop, Start)], [], Tree),
12     start_to_end(Tree, Moves), !.
13
14 depthfirst([], Tree, Tree).
15
16 depthfirst([move(Rewards, Parent, Label, Child)|Frontier], Acc, Tree) :-
17     (    visited(Acc, move(Rewards, Parent, Label, Child))
18     ->   (    member(move(Rewards, Parent, Label, Child), Acc) % Can different path to existing
19          ->   AccOut = Acc
20          ;    backward_induction([move(Rewards, Parent, Label, Child)], [move(Rewards, Parent, L
```

?- Your query goes here ...

Examples History Solutions table results Run!

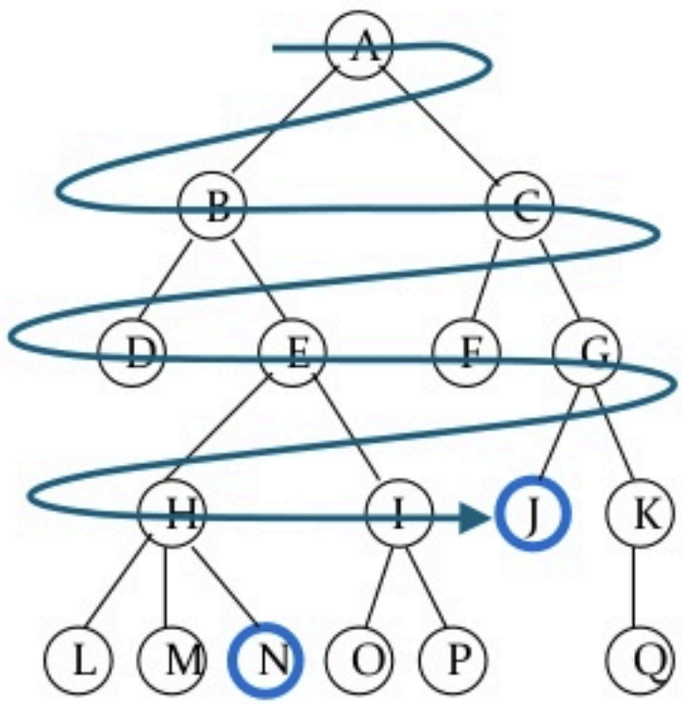
```
% Returns a list of does(Player, Move) clauses in sequence to
% solve the puzzle using depth first search
solve_dfs(Moves) :-
    findinit(Start),
    heuristic(Start, Rewards), % Assuming start not terminal
    depthfirst([move(Rewards, start, noop, Start)], [], Tree),
    start_to_end(Tree, Moves), !.

depthfirst([], Tree, Tree).

depthfirst([move(Rewards, Parent, Label, Child)|Frontier], Acc, Tree) :-
    (    visited(Acc, move(Rewards, Parent, Label, Child))
    ->   (    member(move(Rewards, Parent, Label, Child), Acc) % Can different path to existi
         ->   AccOut = Acc
         ;    backward_induction([move(Rewards, Parent, Label, Child)], [move(Rewards, Parent,
         ),
         NewFrontier = Frontier
    ;    backward_induction([move(Rewards, Parent, Label, Child)], [move(Rewards, Parent, Labe
         (    alpha_beta_test(AccOut, move(Rewards, Parent, Label, Child))
         ->   generate_children(Child, Unsorted),
              sort_generated_children(Unsorted, Children),
              append(Children, Frontier, NewFrontier)
         ;    NewFrontier = Frontier
         )
    ),
    depthfirst(NewFrontier, AccOut, Tree).
```

Pseudocódigo Depth-first

- Recebe uma lista (grafo) de entrada;
- Chama o procedimento depth-firs(Lista, Alvo, Caminho)
  - "Push" o primeiro elemento da lista e compara com o Alvo;
  - "Pop" o primeiro elemento da lista e chama recursivamente resto da lista;
- ...

Busca em largura

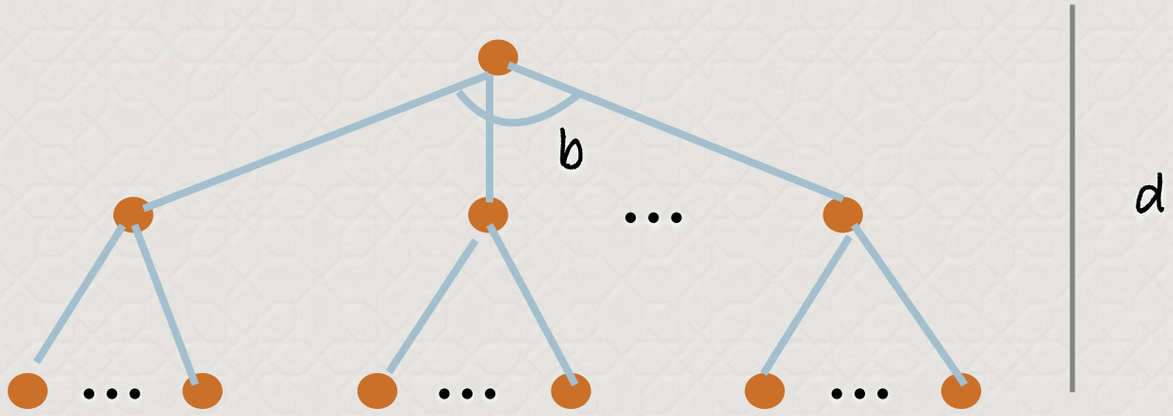
Breadth-first searching[1]



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

```
34 %% solve_bfs(-Moves) is nondet
35 % Returns a list of does(Player, Move) clauses in sequence to
36 % solve the puzzle using breadth first search
37 solve_bfs(Moves) :-
38     findinit(Start),
39     heuristic(Start, Rewards), % Assuming start not terminal
40     breadthfirst([move(Rewards, start, noop, Start)], [], Tree),
41     start_to_end(Tree, Moves), !.
42
43 breadthfirst([], Tree, Tree).
44
45 breadthfirst([move(Rewards, Parent, Label, Child)|Frontier], Acc, Tree) :-
46     (    visited(Acc, move(Rewards, Parent, Label, Child))
47     ->   (    member(move(Rewards, Parent, Label, Child), Acc) % Can different path to existin
48          ->   AccOut = Acc
49          ;    backward_induction([move(Rewards, Parent, Label, Child)], [move(Rewards, Parent,
50          ),
51          NewFrontier = Frontier
52     ;    backward_induction([move(Rewards, Parent, Label, Child)], [move(Rewards, Parent, Labe
53          (    alpha_beta_test(AccOut, move(Rewards, Parent, Label, Child))
54          ->   generate_children(Child, Unsorted),
55               sort_generated_children(Unsorted, Children),
56               append(Frontier, Children, NewFrontier)
57          ;    NewFrontier = Frontier
58          )
59     ),
60     breadthfirst(NewFrontier, AccOut, Tree).
```

Seja qual for o algoritmo (em profundidade, largura, ou outro) a busca sobre árvores tem a seguinte característica:



$$S = (b^d - 1)/(b - 1) \sim O(b^d)$$

Outras variantes de busca não-informada: depth-limited search

Define-se como o diâmetro de um espaço de busca ao inteiro ℓ, tal que, o estado atingível de "maior distância" pode ser alcançado em pelo menos ℓ disparos de transições. Uma análise do grafo de atingibilidade pode usado para determinar o valor de ℓ.

Uma análise do grafo ao lado mostra que Bucharest pode ser "atingida" com no máximo nove deslocamentos. Portanto se após nove deslocamentos o "alvo" de chegar na capital não for "resolvido" podemos abandonar a busca.

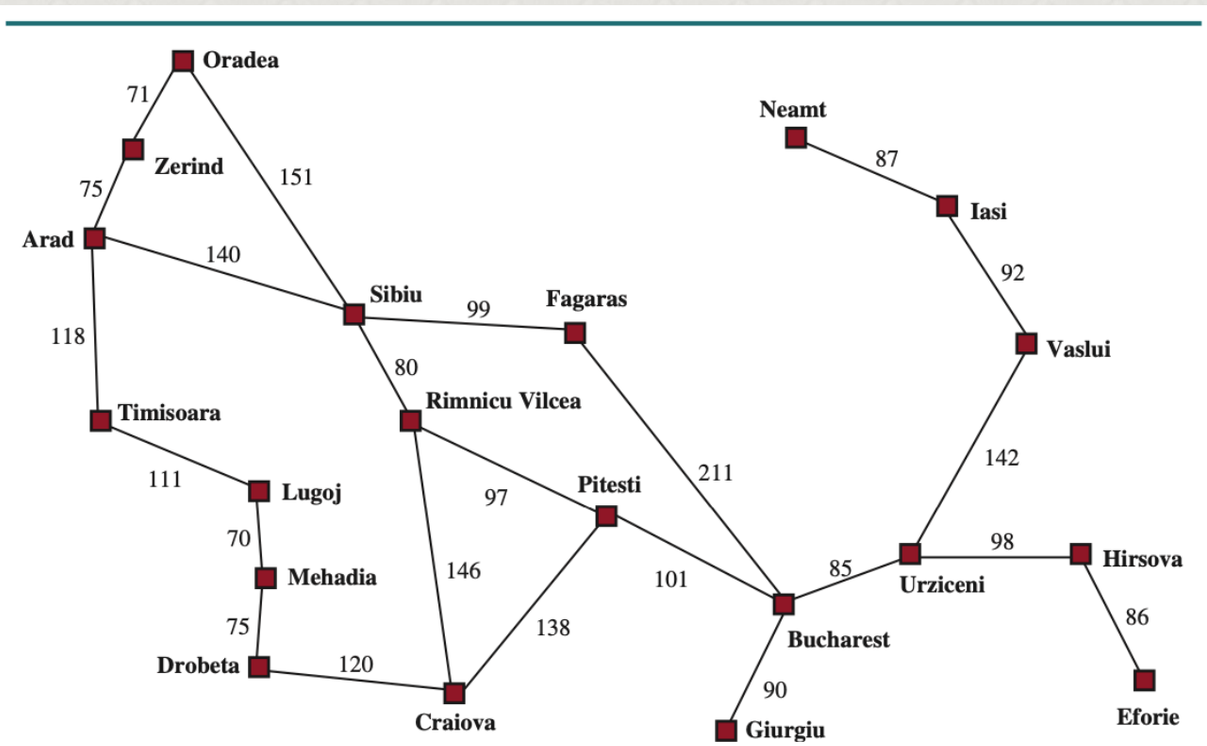
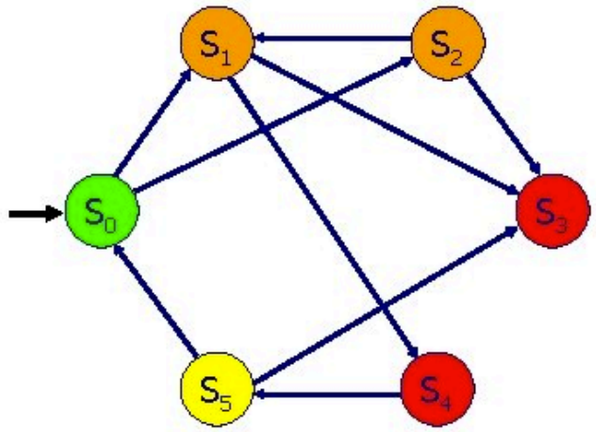


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Esta é a base do algoritmo depth-limited search. Note que apesar de ser ainda um algoritmos de busca não informada a identificação do diâmetro do espaço de busca é baseada em uma análise do problema ou em "Knowledge Engineering".

The State Space Diameter Problem



initial state: S_0

step 1: S_1, S_2

step 2: S_3, S_4

step 3: S_5

diameter = 3

Start from the initial states, the minimum number of steps needed to visit every reachable state

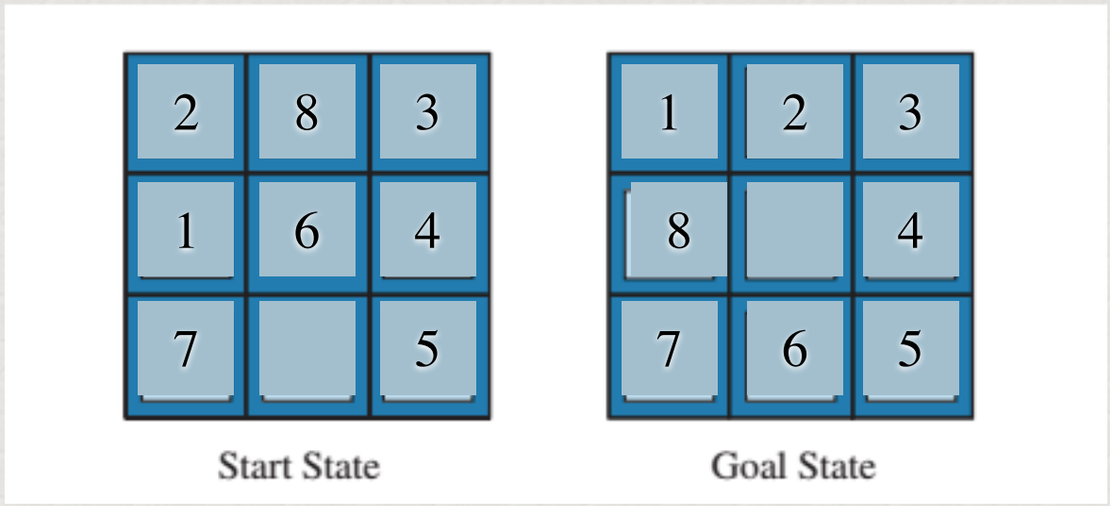
Decision Procedures - An algorithmic point of view 22

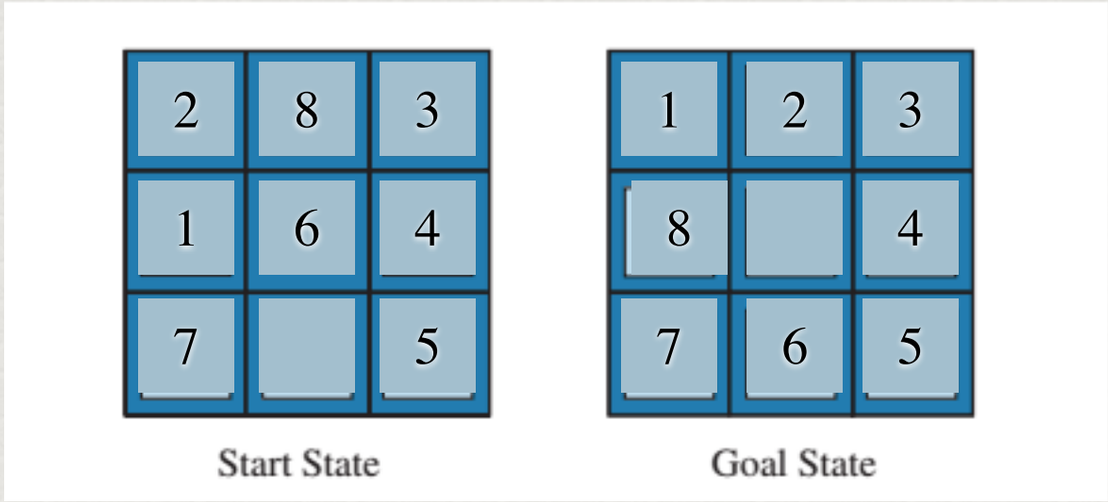
Outras variantes de busca não-informada: bidirectional search

Consiste em "varrer" o espaço de estados em duas direções: evoluindo do estado inicial e retornando à partir do estado final.

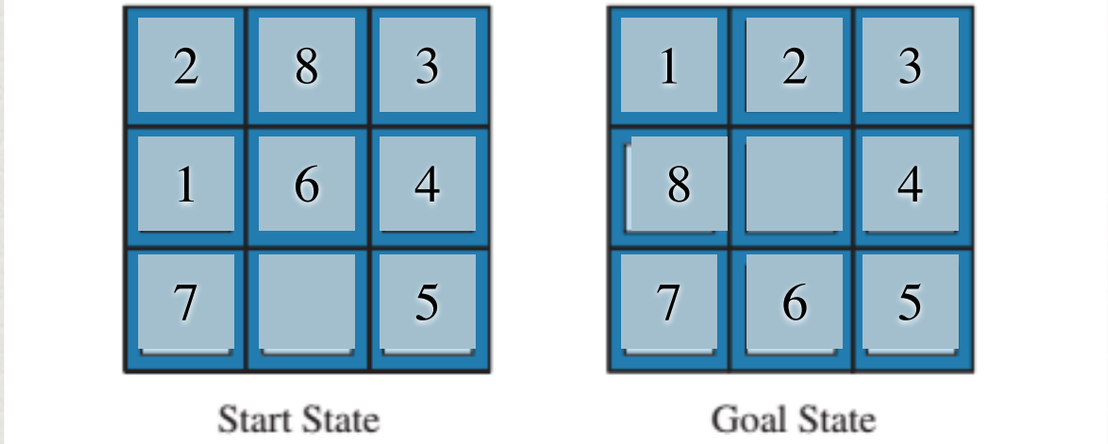
Excetuando as duas primeiras abordagens, depth and breadth-first todos os demais algoritmos dependem de uma análise do "domínio" (do espaço de estados).

Vamos tomar como exemplo o jogo de tiles com os estados inicial e final mostrados abaixo.



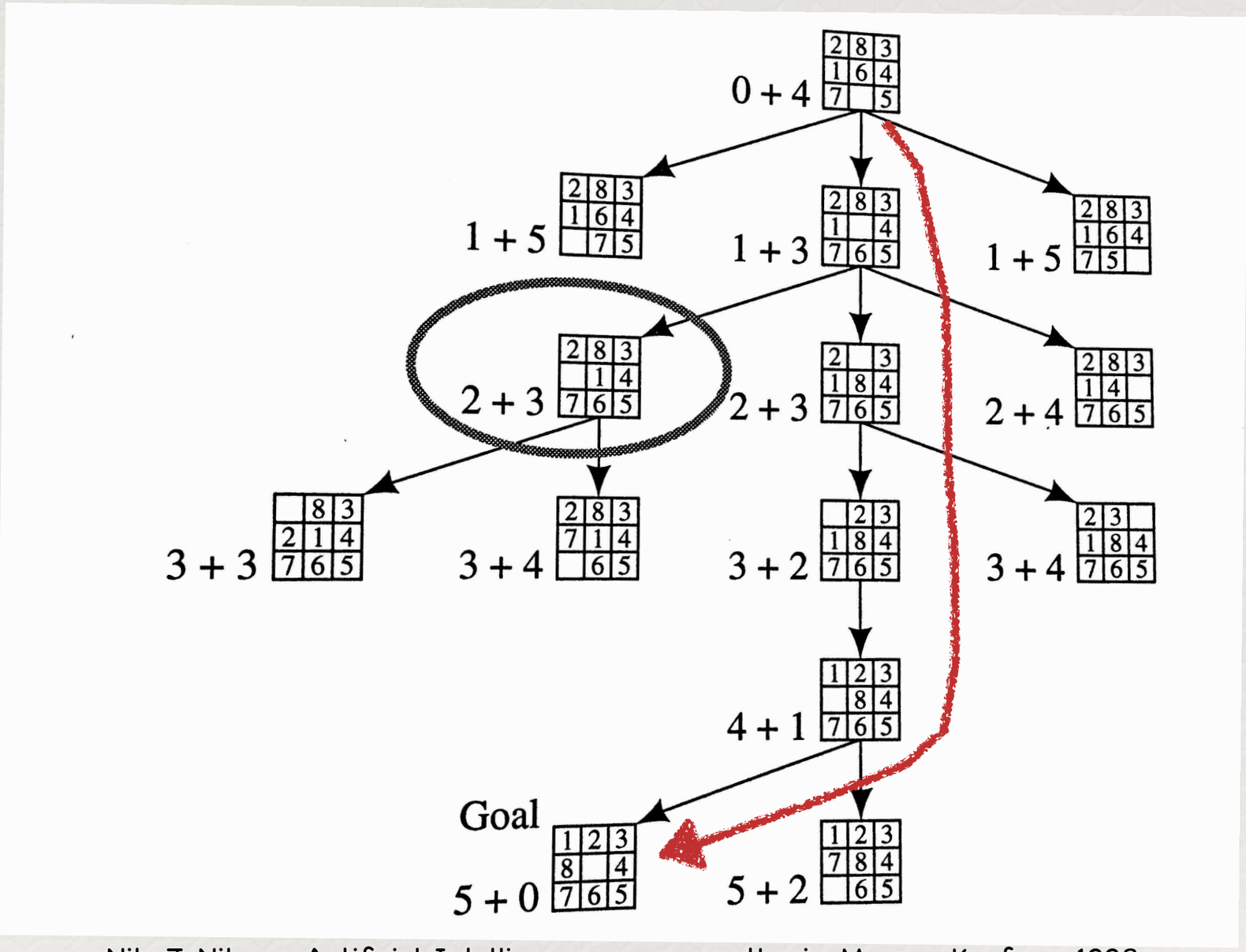


Comparando o estado inicial e o estado final temos 4 tiles fora do lugar: 1, 2, 8, e 6. Os demais estão no lugar mostrado no estado final.



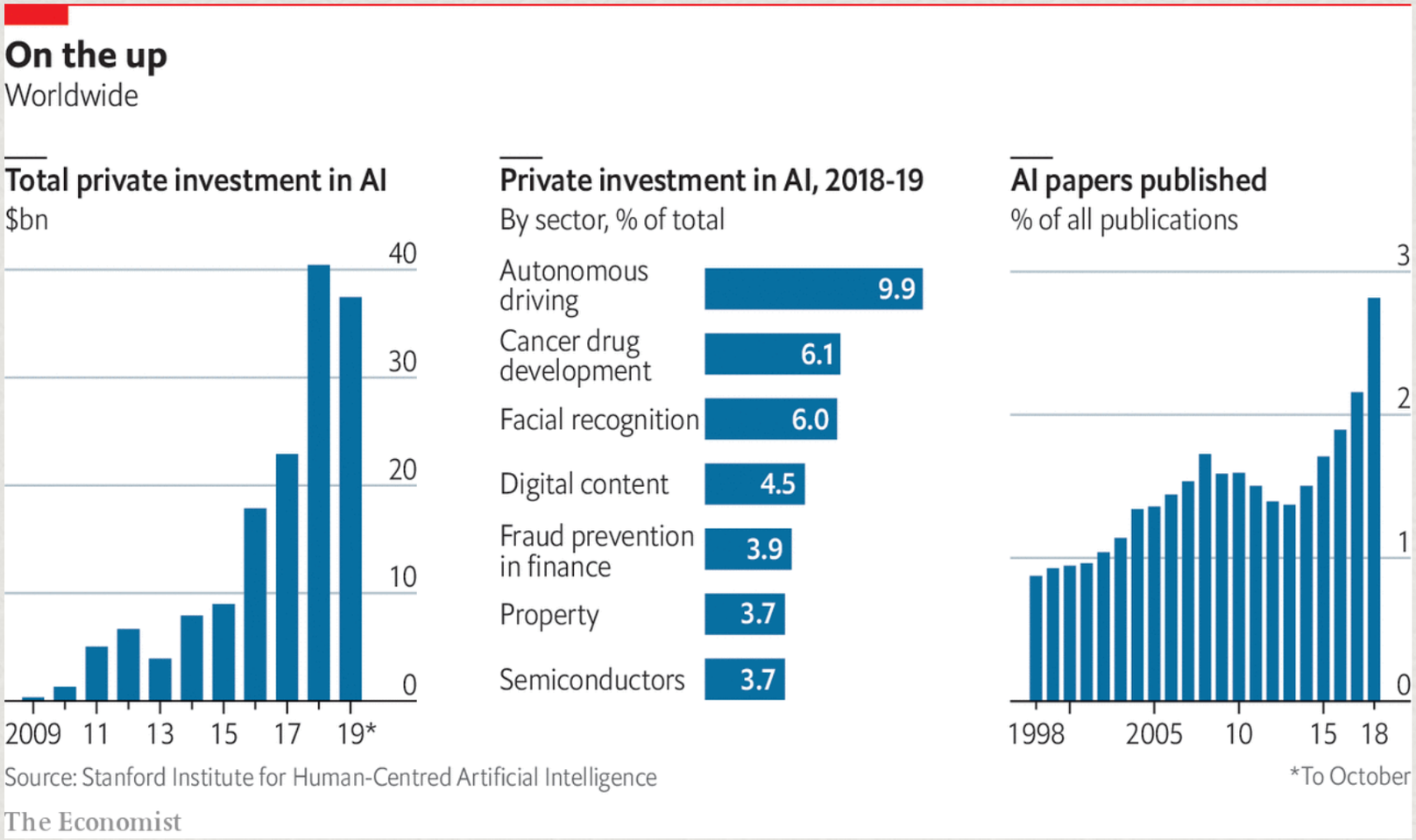
"Informar" o processo de busca significa achar algum critério de avaliação sobre o "custo" ou esforço que se tem que fazer para colocar os tiles no lugar. Uma possibilidade é considerar inteiros ligados à profundidade da árvore de busca e ao número de tiles fora do lugar.

Árvore de busca e a busca informada



Nils J. Nilsson, Artificial Intelligence: a new synthesis, Morgan Kaufann, 1998

Neste exemplo tentamos explorar o "melhor caminho" para prosseguir a busca. Voltaremos a este ponto na aula que vem para encerrar a discussão sobre os algoritmos de busca.



É importante ter em mente também que alguns problemas podem ter soluções "inteligentes" usando algoritmos simples usando busca não-informada.

Perguntas?