Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 3 - Resolução de problemas em IA

Prof. José Reinaldo Silva

reinaldo@usp.br

Resolução de problemas simples com regras

Editar

Reunião no Dartmouth College para discutir a "inteligência Artificial".

Nesta aula vamos começar a tratar da resolução de problemas de forma inteligente. A idéia é capacitar máquinas (computadores) para resolver problemas, uma capacidade normalmente associada a sistemas (humanos) inteligentes. Já vimos as limitações que esta inteligência de máquina pode ter, mas ainda assim, seria uma evolução tecnológica considerável ter sistemas capazes de fazer planos, aprender analizar padrões ou dar apoio à decisão. Ao invés de introduzir este tópico de maneira formal vamos abordá-lo (inicialmente) de forma simples com alguns problemas que podem ser tratados com "regras", agindo sobre uma base de fatos.

Podemos sofisticar o nível de abstração destas regras (a distância para os "fatos") e também o número de regras tornando os programas mais complicados e mais "inteligentes". Mas é ainda possível ver as limitações e também as possibilidades que surgem para aplicações, especialmente em automação. Aproveitamos para introduzir a linguagem Prolog e exercitar a sua aplicação no Swish SWI Prolog.

Terminaremos propondo um avanço na forma de resolver problemas que nos aproxima ainda mais da modelagem estado-transição (ou dos "transition systems"). Novamente vamos entrar neste assunto de forma prática para colocar mais tarde a abordagem formal. A aplicação no momento seria um sistema automático para jogos.

Cerca de 30% da emissão de passagens em companhias aéreas é feita usando Prolog.

Uma matéria divulgada pela SICStus Prolog, que faz um ambiente de programação Prolog onde se pode produzir programas de interesse de mercado. Veja o link a abaixo...

Materia da SICStus Prolog — Editar

Aula2 — Editar

Exercício1 — Editar

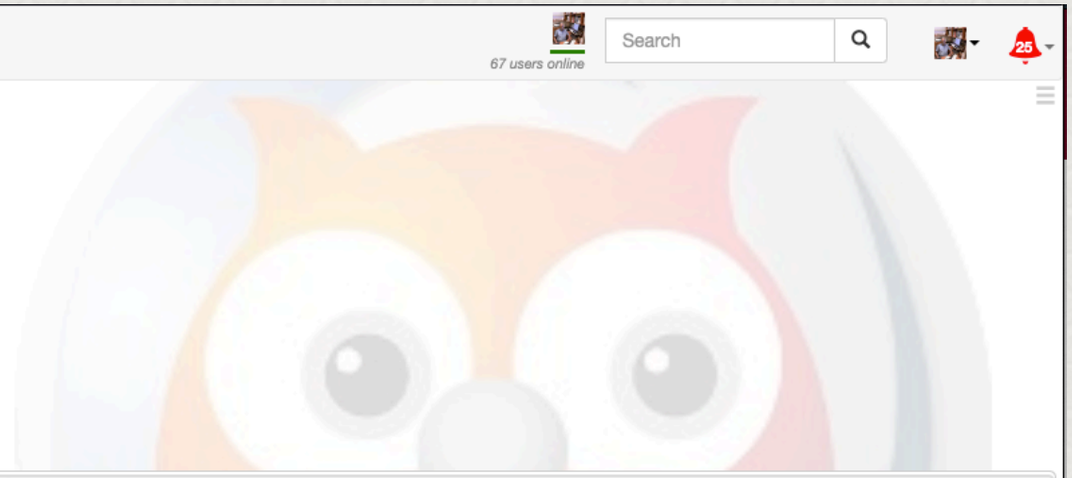
Faça o upload do exercício 1. Este exercício consiste em se familiarizar com a programação Prolog implementando um

...usando o comando “or" ou “;” nos queries.

SWISH File Edit Examples Help — 67 users online — Search

Program

```
1  mae(maria,paulo).
2  mae(maria,carla).
3  mae(susaza,jose).
4  mae(vania,mara).
5  mae(carla,antonio).
6
7  pai(flavio,jose).
8  pai(flavio,beatriz).
9
10 corintiana(maria).
11
12 irmao(X,Y):-mae(Z,X),mae(Z,Y),dif(X,Y).
13 irmao(X,Y):-pai(Z,X),pai(Z,Y),dif(X,Y).
14
```



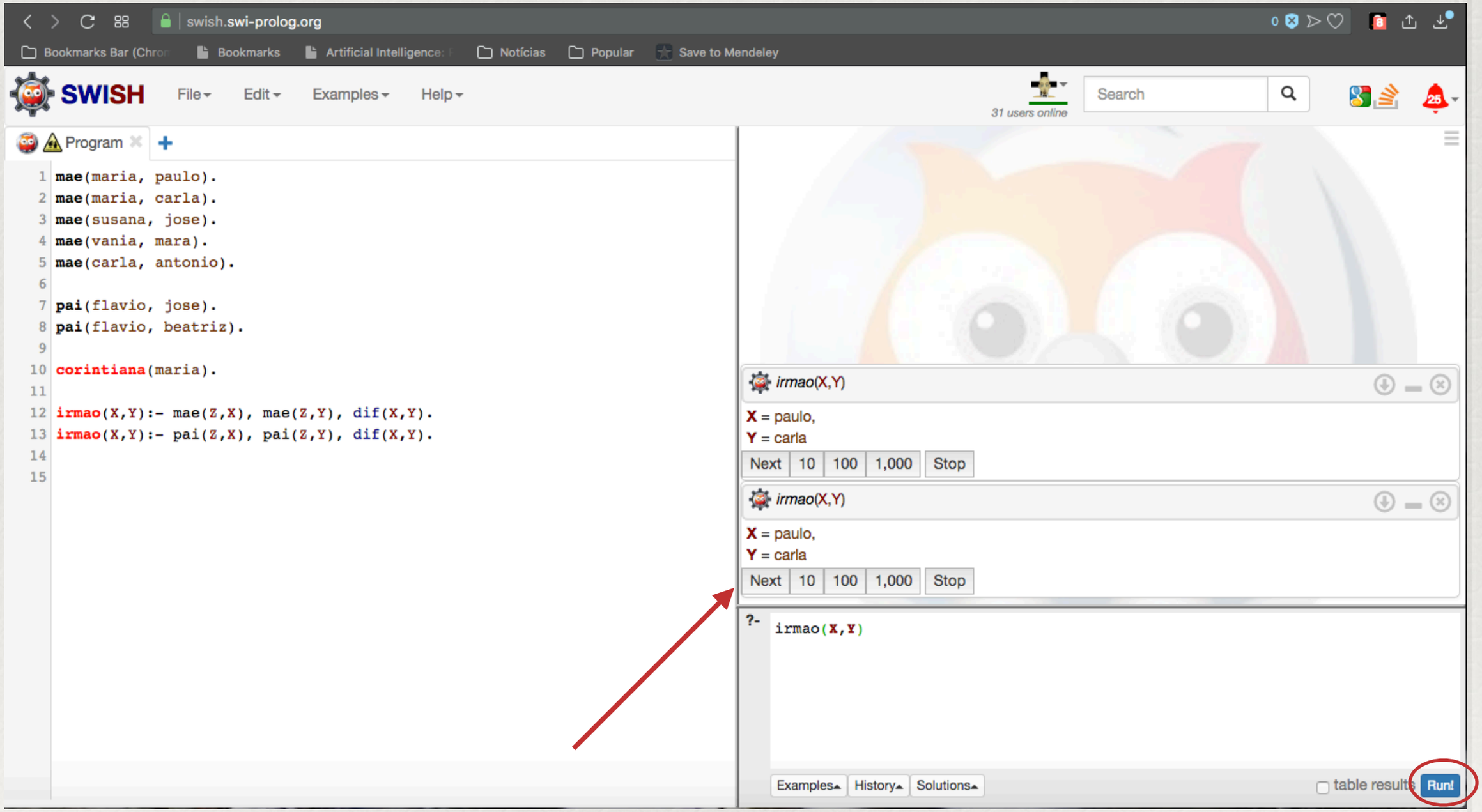
irmao(X,Y).

X = paulo,
Y = carla
X = carla,
Y = paulo
X = jose,
Y = beatriz
X = beatriz,
Y = jose
false

?- irmao(X,Y).

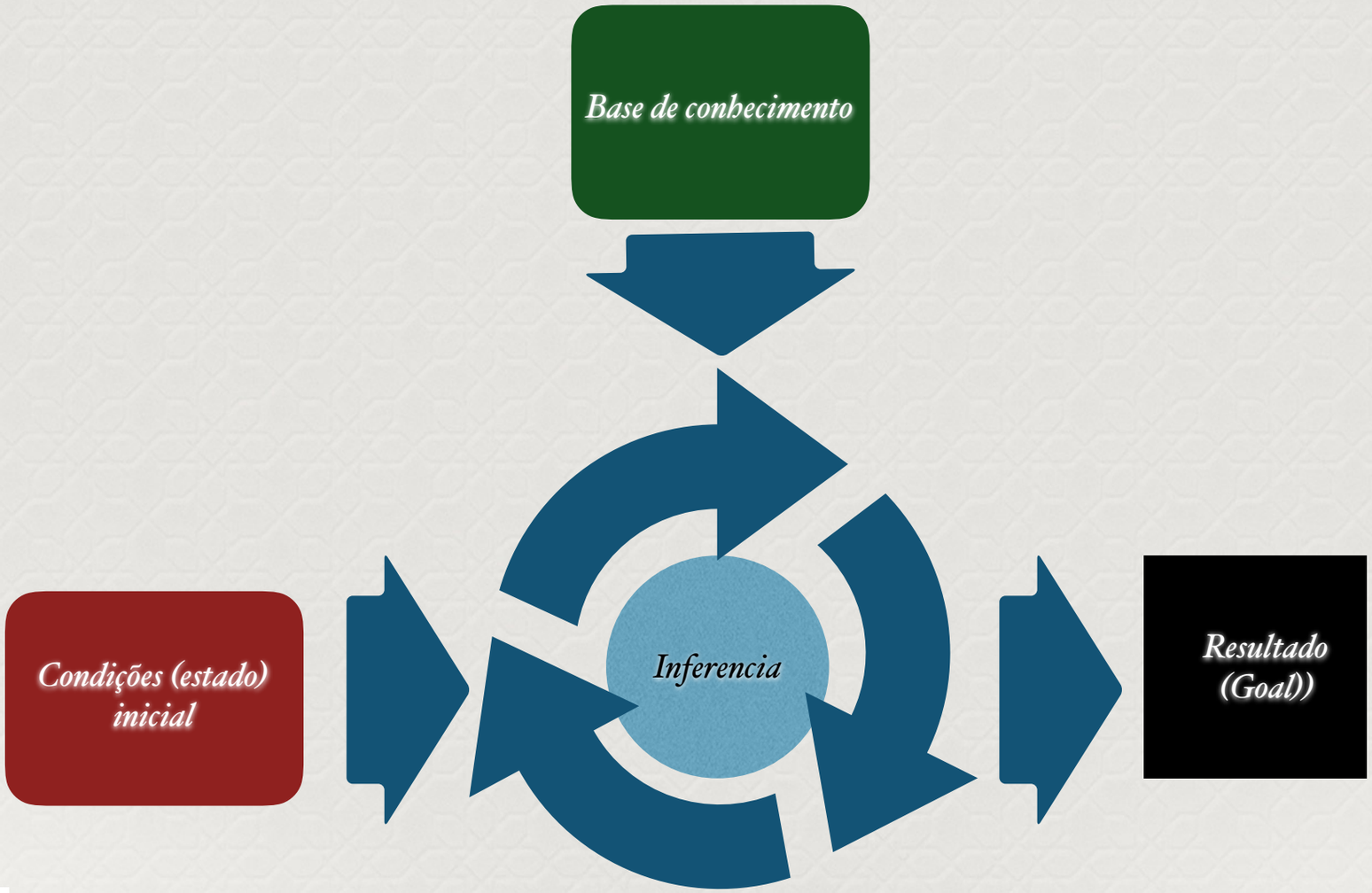
Examples History Solutions — table results — Run!

swish.swi-prolog.org



Como se usa IA clássica (deep knowledge) para resolver problemas?

Em uma primeira abordagem, gostaríamos de ter "agentes inteligentes" capazes de "resolver problemas". O que significa isso?



Como "resolver problemas automaticamente"

Podemos utilizar duas grande abordagens para resolver problemas automaticamente:

Problema

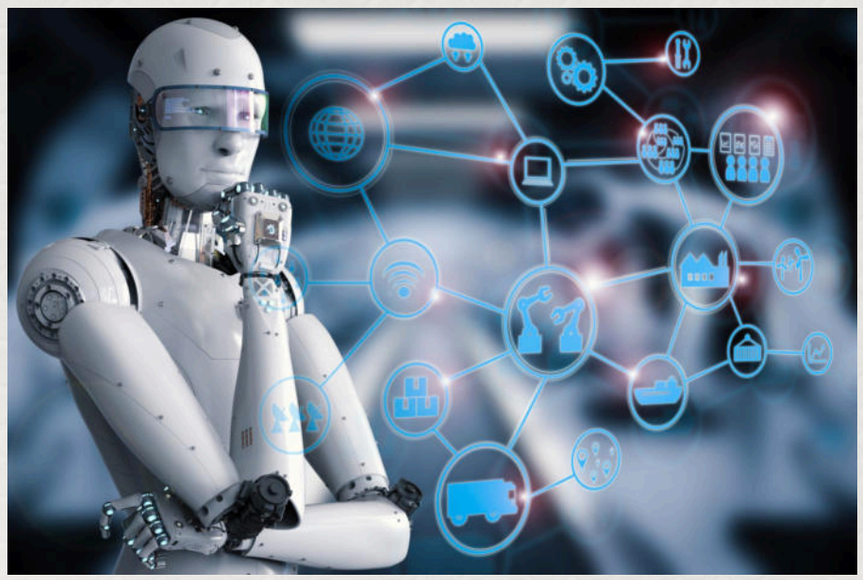
1. achar uma abordagem geral que leva do estado inicial ao estado final;
2. testar esta abordagem em alguns casos (sem levar em conta o tempo para chegar à solução);
3. checar se a abordagem é completa, isto é, resolve todos os casos ou há casos especiais onde o problema "não converge";
4. preparar a implementação do revolvedor (estrutura de dados e base de conhecimento, regras de dedução);

Solução

Achar um método geral de resolução de problemas

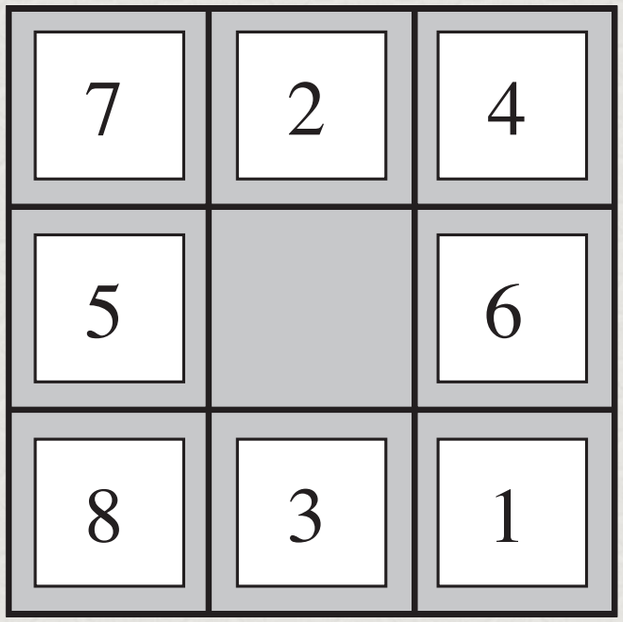
Estrutura de um "resolvedor automático de problemas"

Para dotar uma máquina da capacidade de resolver problemas (ou uma classe de problemas) é preciso ter uma estrutura com os seguintes atributos:

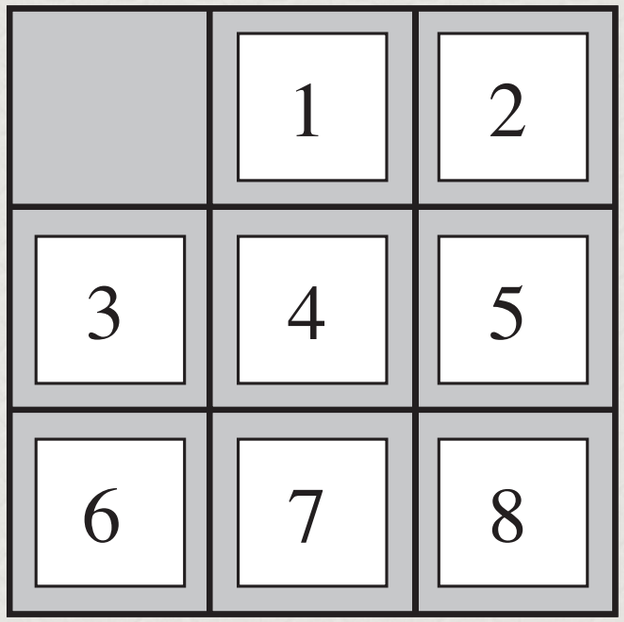


1. uma descrição clara do "estado inicial" ou seja das condições iniciais do problema a ser resolvido;
2. uma descrição clara do objetivo ou "estado final", de modo que seja possível saber quando (e se) o problema foi resolvido;
3. em cada estágio do processo de solução saber quais os próximos estados que podem ser atingidos;
4. poder escolher um (ou o melhor) caminho entre os estados acima;
5. saber que operadores (ou passos) aplicar para fazer a "transição" para um próximo estado;
6. discernir se estamos convergindo para a solução.

Uma solução específica para este problema... ?

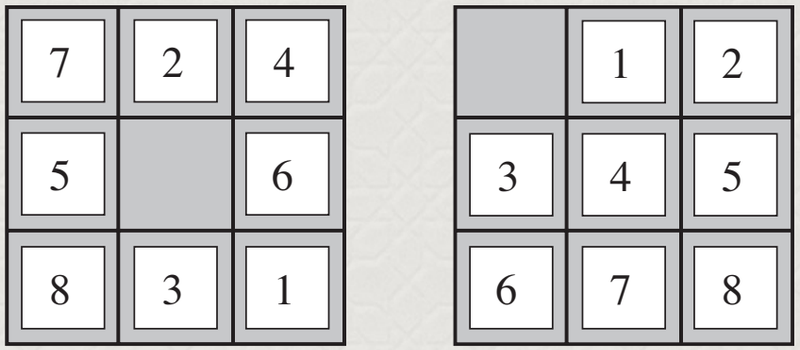


Start State



Goal State

Uma possível solução...



1. o problema consiste em receber uma configuração genérica de "tiles" (pastilha) - o estado inicial - e colocar os tiles numerados em ordem crescente (o estado final);
2. portanto uma possível solução específica é posicionar o tile maior e assim sucessivamente até ordenar todos;

1. achar uma abordagem geral que leva do estado inicial ao estado final; ✓
2. testar esta abordagem em alguns casos (sem levar em conta o tempo para chegar à solução); ✓
3. checar se a abordagem é completa, isto é, resolve todos os casos ou há casos especiais onde o problema "não converge";
4. preparar a implementação do revolvedor (estrutura de dados e base de conhecimento, regras de dedução);

Nesta aula vamos discutir a primeira abordagem...

... antes porém vamos deixar claro a hierarquia de "métodos" para resolução de problemas que vamos abordar nesta e nas próximas aulas:

Paradigma de resolução: estado/transição

método geral de resolução (STRIPS)

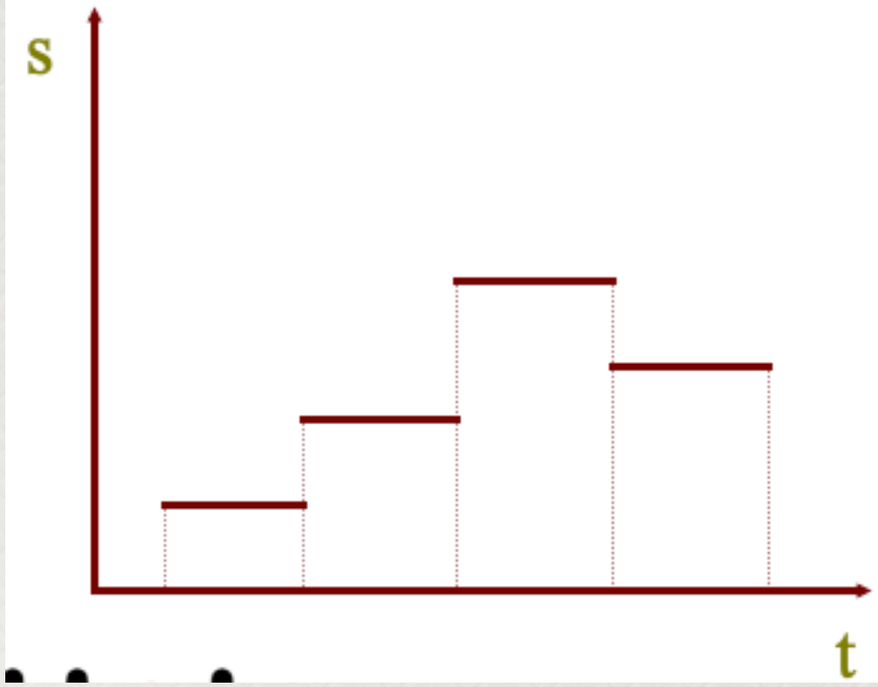
Solução específica para os problemas

abstração

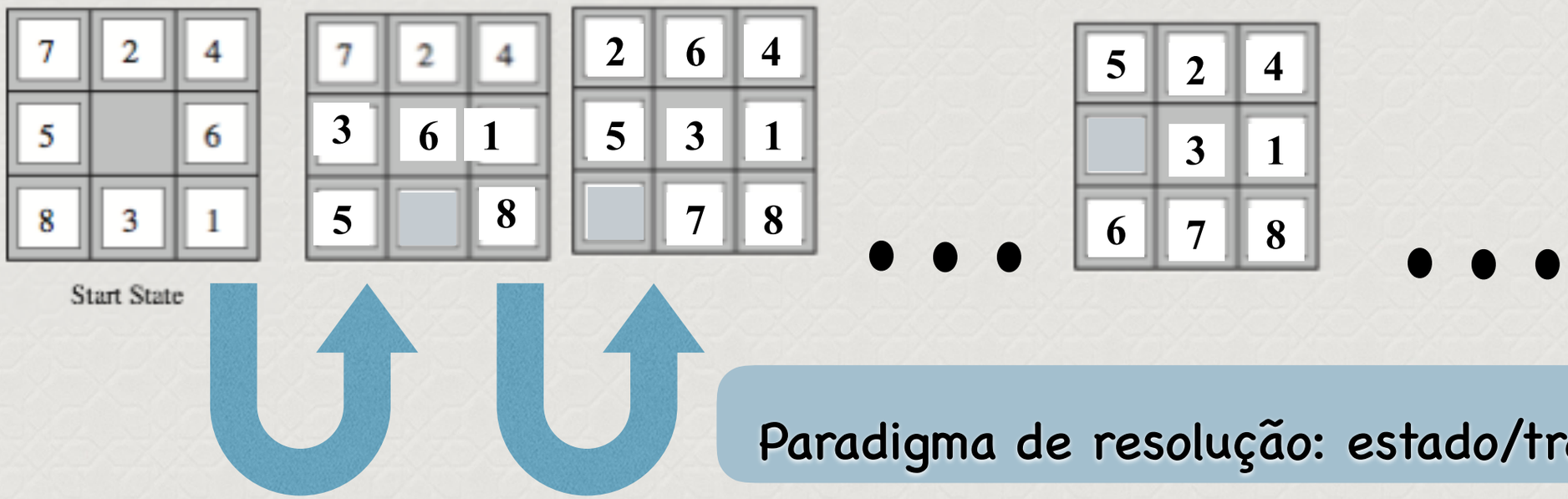
Representação em grafos e redes

Sistemas a Eventos Discretos

Eventos causam uma mudança instantânea no estado



Uma solução específica para este problema... ?



Paradigma de resolução: estado/transição

método geral de resolução (STRIPS)

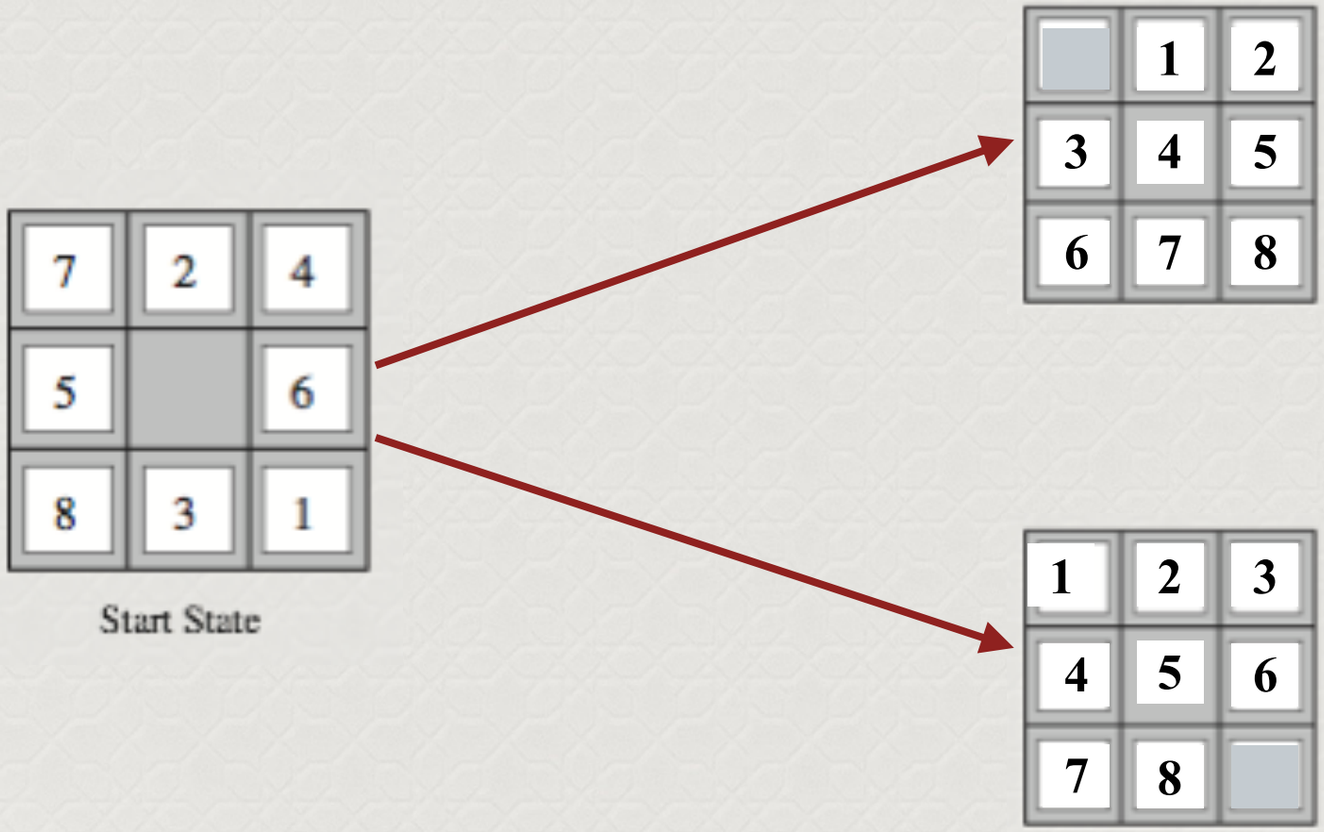
Solução específica para os problemas

Como se define "estado" para este problema?

Portanto a estratégia de solução passa por definir claramente o estado do sistema.

Start State

- ao definir o estado define-se também a transição (mudança de estado) já que a idéia inicial é ter um "transition system";
- voltamos então à questão do "objetivo" (goal) que é o estado final, portanto é preciso definir um critério de convergência para o estado final;
- define-se então uma "heurística" ou seja, forma de escolher "a melhor opção" entre as possíveis transições;


7 2 4
5 6
8 3 1
Start State
1 2
3 4 5
6 7 8
1 2 3
4 5 6
7 8

Tree search algorithms

Basic idea:

offline, simulated exploration of state space

by generating successors of already-explored states

(a.k.a. expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

AIMA - UC Berkeley

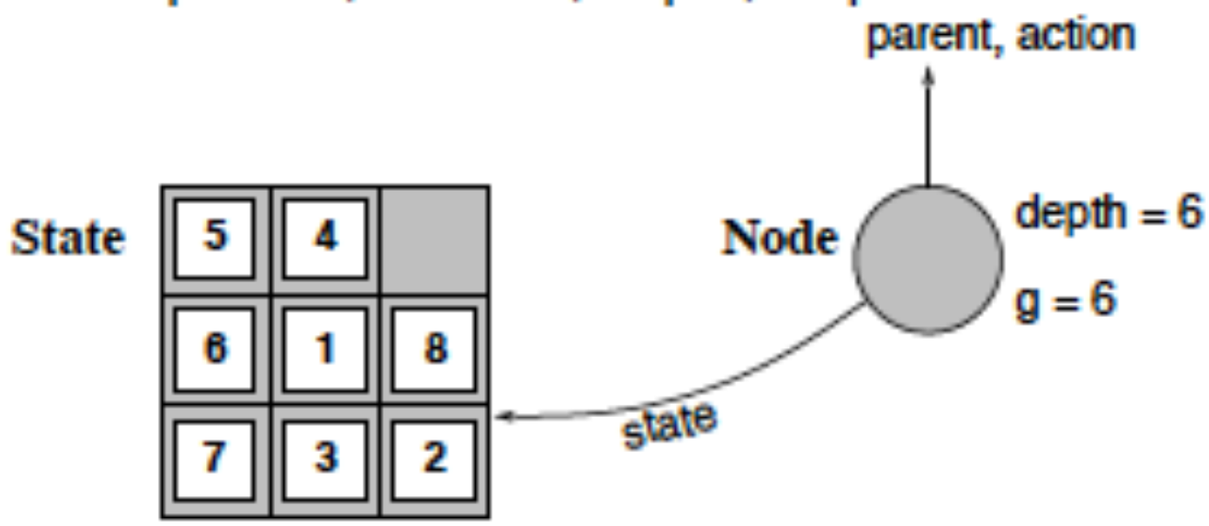
Implementation: states vs. nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree

includes parent, children, depth, path cost $g(x)$

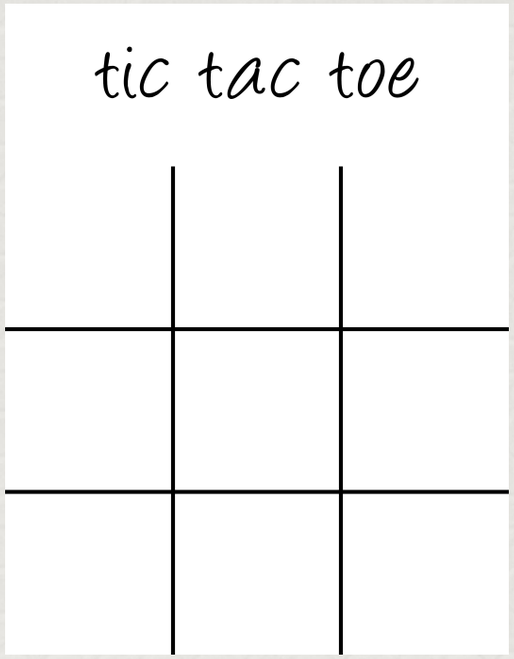
States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the SUCCESSORFN of the problem to create the corresponding states.

AIMA - UC Berkeley

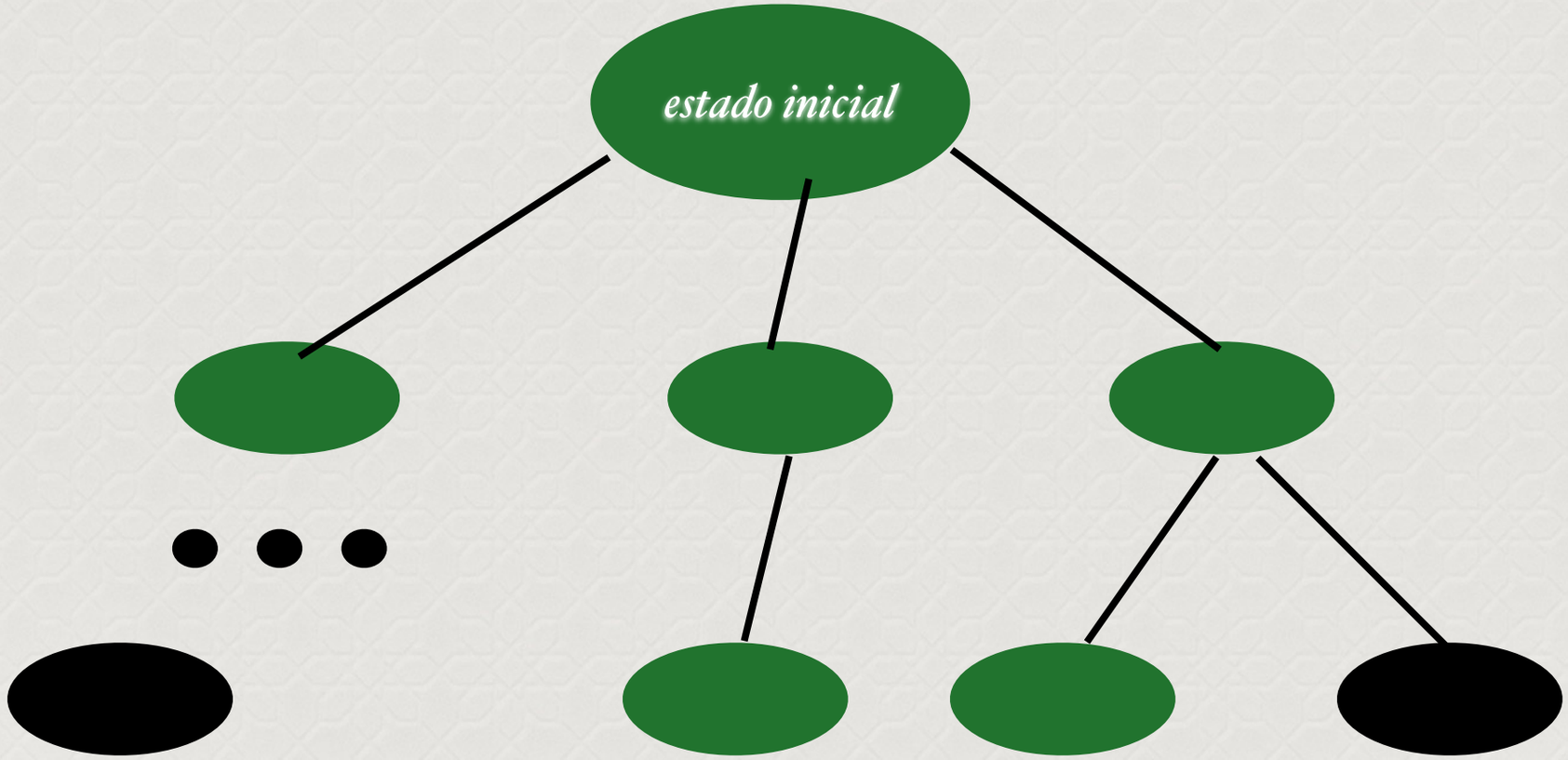
Como discutimos na aula passada existe ainda outro tipo de "problema"... também discreto.

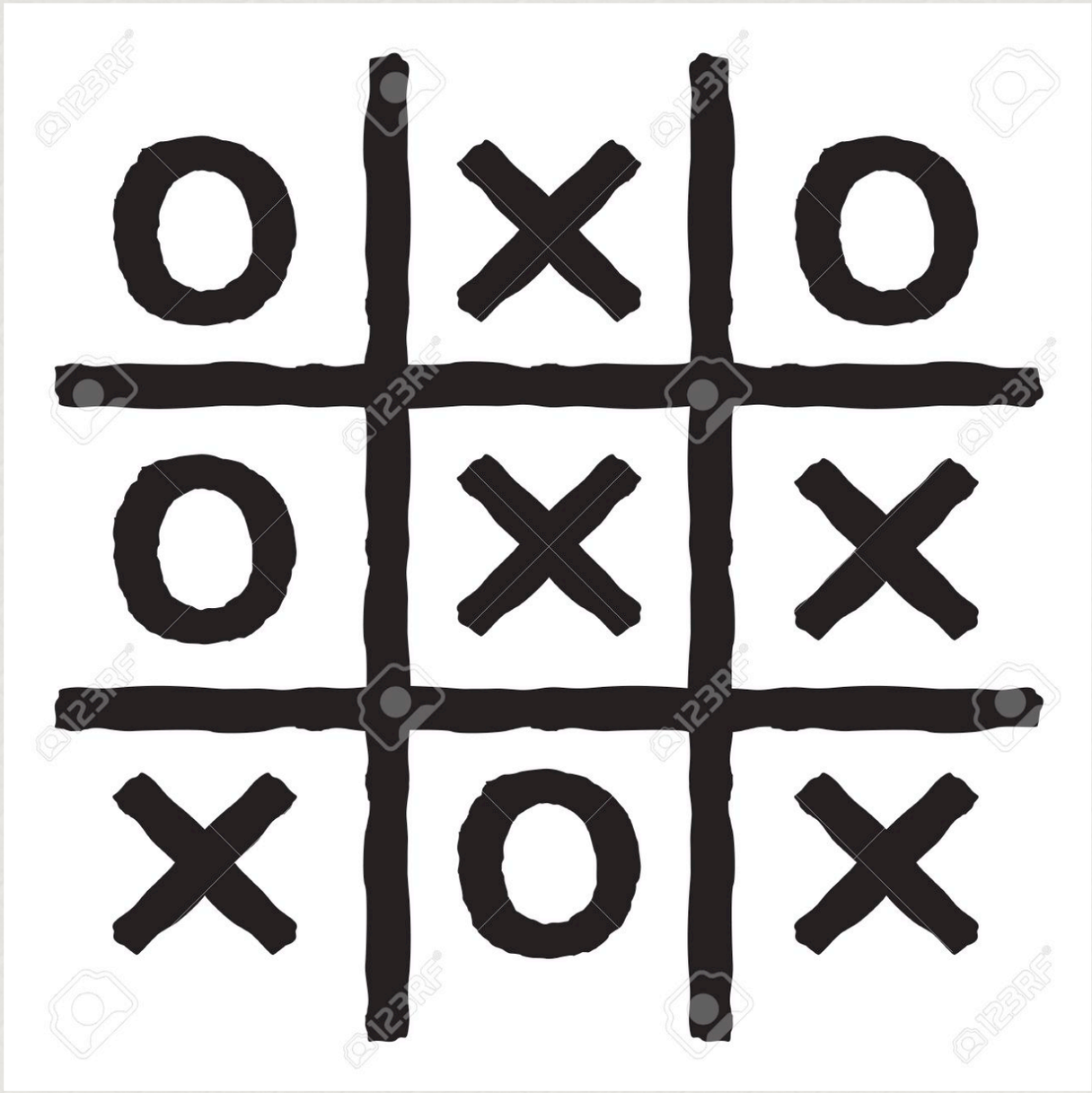


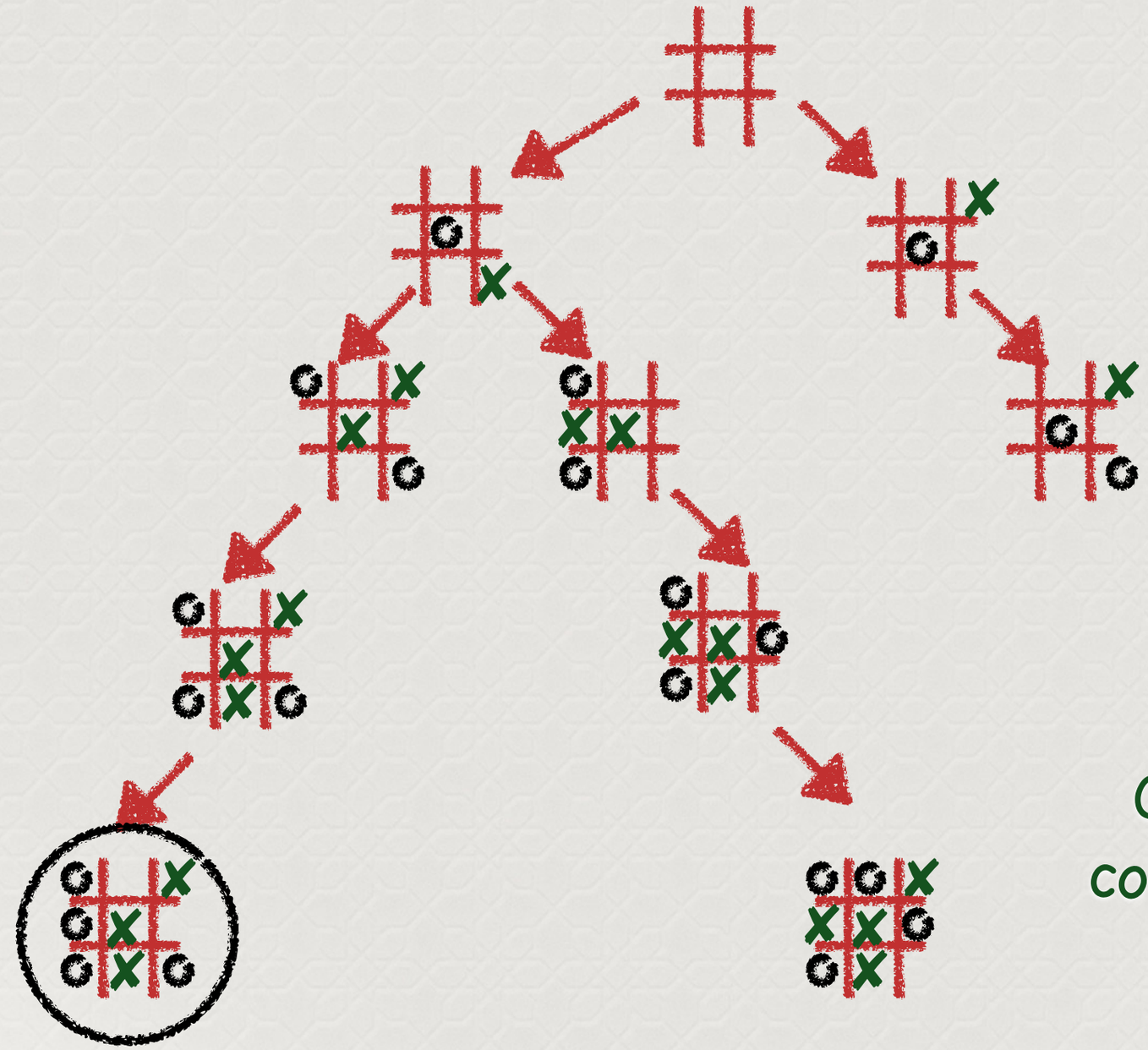
estado inicial

estado final

A estrutura do revolvedor de problemas é uma árvore, onde a raiz é o estado inicial.







O programa termina
com no máximo 5 níveis
à partir da raiz

O algoritmo para jogar...

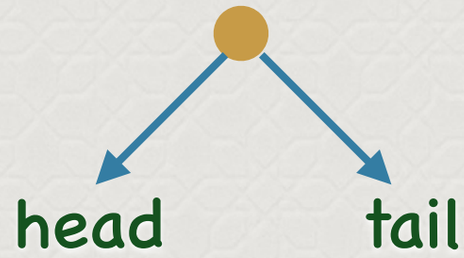
- Interface e interação com o jogador humano
- Inicialização
- Algoritmo
- Pontos de parada (como o jogo termina)

- Interface e interação com o jogador humano

```
display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
                                write([G,H,I]),nl,nl.
```

Estrutura de lista: [head | tail]

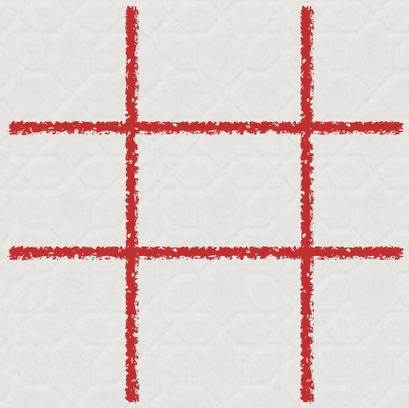
Estrutura de lista: [head | tail]



List	Head	Tail
[a, b, c]	a	[b, c]
[]	(none)	(none)
[[the, cat], sat]	[the, cat]	[sat]
[the, [cat, sat]]	the	[[cat, sat]]
[the, [cat, sat], down]	the	[[cat, sat], down]
[X+Y, x+y]	X+Y	[x+y]

- Inicialização

playfrom([b,b,b,b,b,b,b,b,b])



- Algoritmo

Será o nosso próximo exercício: veja o programa Prolog que implementa o jogo tic-tac-toe (jogo da velha) proposto por S. Tanimoto e que pertence à biblioteca do SWISH. Identifique e procure entender a parte do programa que resolve a jogada do computador. (https://swish.swi-prolog.org/p/Tic-Tac-Toe.swinb)

- Pontos de parada (como o jogo termina)

```
win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].

diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].
```

SWISH File Edit Examples Help

25 users online

Search

Tic-Tac-Toe

```
% A Tic-Tac-Toe program in Prolog.   S. Tanimoto, May 11, 2003.
% To play a game with the computer, type
% playo.
% To watch the computer play a game with itself, type
% selfgame.
%
% original at https://courses.cs.washington.edu/courses/cse341/03sp/slides/Prolog

% Predicates that define the winning conditions:

win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].

diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].

% Helping predicate for alternating play in a "self" game:

other(x,o).
```

?- playo

?- Your query goes here ...

Examples History Solutions table results Run!

SWISH File Edit Examples Help

25 users online

Search

Tic-Tac-Toe

```
31 game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).
32 game(Board, Player) :-
33   other(Player,Otherplayer),
34   move(Board,Player,Newboard),
35   !,
36   display(Newboard),
37   game(Newboard,Otherplayer).
38
39 move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).
40 move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).
41 move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).
42 move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).
```

```
?- playo
Redefined static procedure display/1
You play X by entering integer positions followed by a period.
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

1.
[x, b, b]
[b, b, b]
[b, b, b]

[x, o, b]
[b, b, b]
[b, b, b]
```

Please enter a Prolog term | Send | Abort

?- Your query goes here ...

Examples History Solutions table results Run!

Problem types

Deterministic, fully observable $\Longrightarrow$ single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable $\Longrightarrow$ conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable $\Longrightarrow$ contingency problem

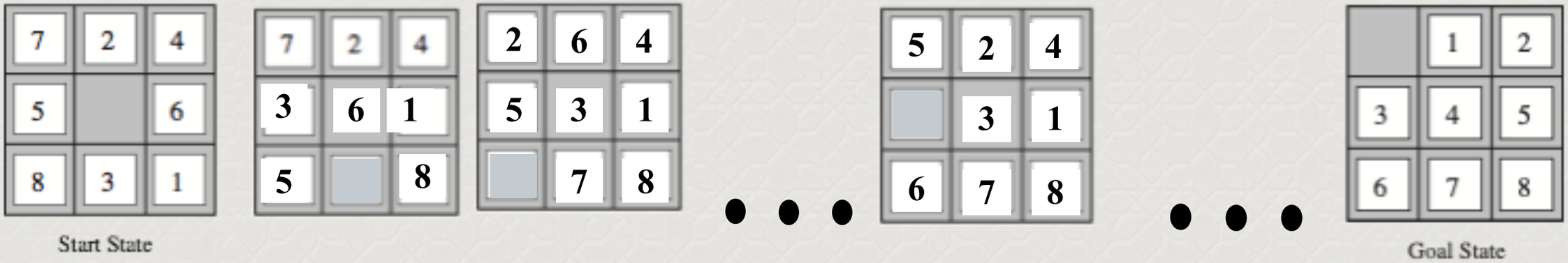
percepts provide **new** information about current state

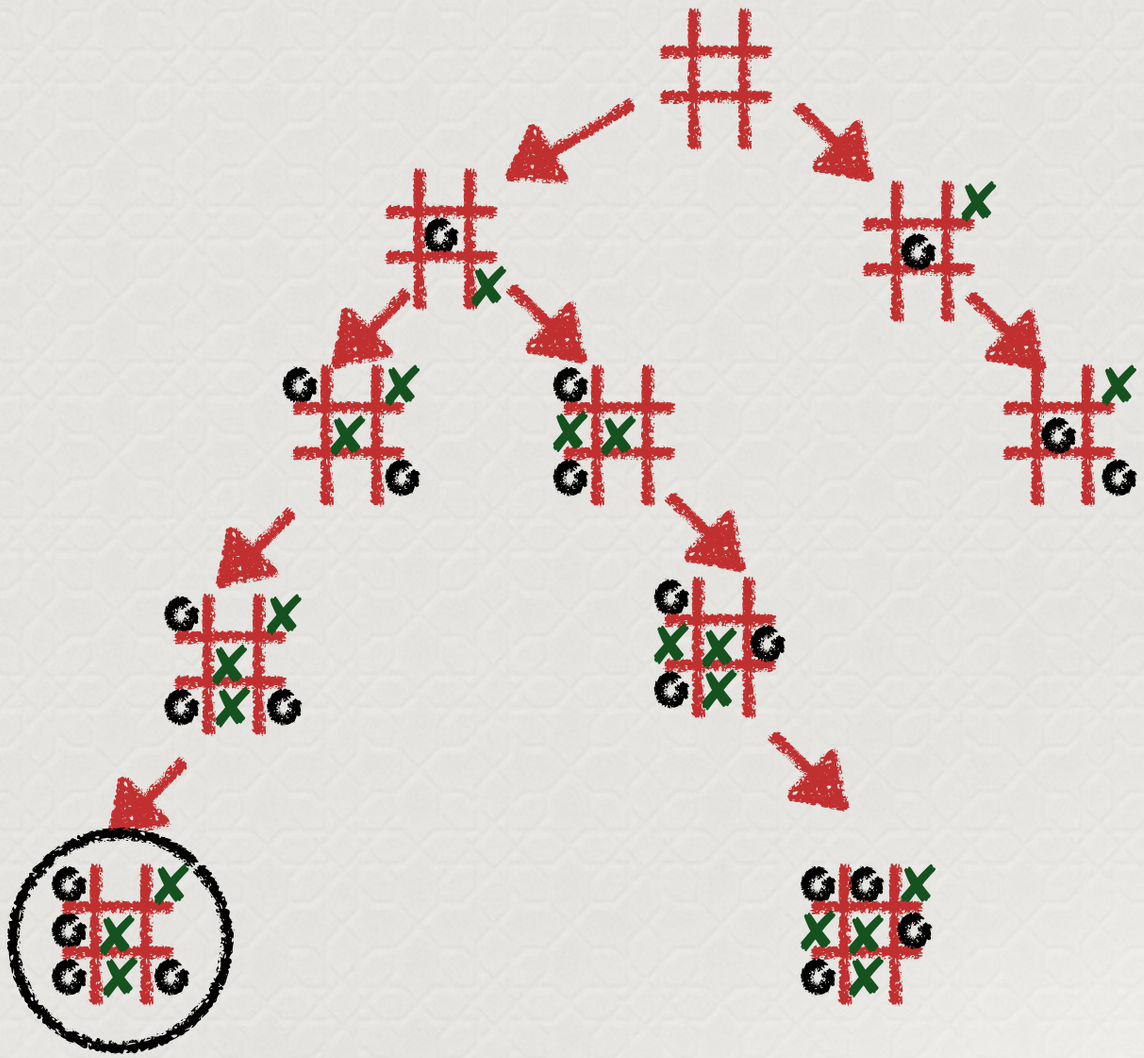
solution is a contingent plan or a policy

often **interleave** search, execution

Unknown state space $\Longrightarrow$ exploration problem ("online")

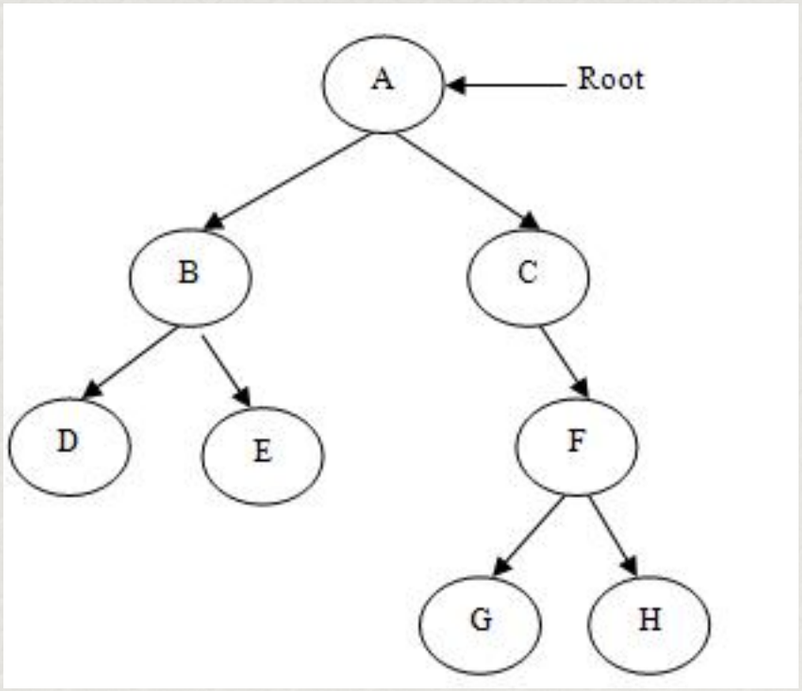
AIMA - UC Berkeley

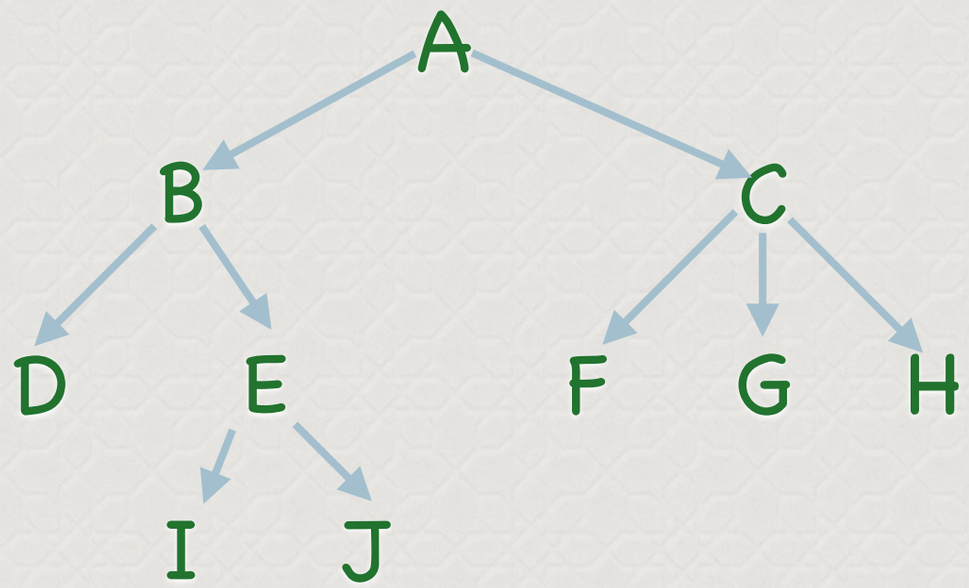

7 2 4
5 6
8 3 1
Start State
7 2 4
3 6 1
5 8
2 6 4
5 3 1
7 8
5 2 4
3 1
6 7 8
1 2
3 4 5
6 7 8
Goal State



Usando árvores como base para a solução

Uma opção muito interessante é modelar o espaço de estados na forma de uma árvore





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

www.roadef.org/challenge/2018/en/

Bookmarks Bar (Chro… | Bookmarks | Artificial Intelligence: F… | Notícias | Popular | Save to Mendeley

ROADEF

Société française de Recherche Opérationnelle et Aide à la Décision

Challenge 2018 | 2016 | 2014 | 2012 | 2010 | 2009 | 2007 | 2005 | 2003 | 2001 | 1999

FEATURES

- Home
- Subject
- Instances and Checker
- Schedule
- Rules
- Awards
- Organizing commitee
- Registration
- Sprint submission
- Sprint results
- Qualification submission
- Scientific Prize

ROADEF/EURO CHALLENGE 2018: CUTTING OPTIMIZATION PROBLEM

Thanks to the success of the previous challenges, the French Operational Research (OR) and Decision Support Society (ROADEF) organizes exceptionaly jointly with the European Operational Research Society (EURO) the ROADEF/EURO challenge 2018 dedicated to cutting optimization problem in collaboration with Saint-Gobain.

This challenge is ***open*** to everyone, and particularly to young researchers, excluding people professionally involved with the industrial partners.

The goal of this challenge has multiple aspects.

First, it allows some of our industrial partners to follow recent developments in the fields of Operations Research and Decision Analysis.

Second, through the **junior category** young researchers have the opportunity to face up to a complex industrial optimization problem. The challenge will give them an opportunity to explore the requirements and difficulties encountered in industrial applications. We hope that this challenge will help to establish a permanent partnership between manufacturers and young scientists on industrial-sized projects which require both high scientific qualification and the real-life practices in companies making use of decision analysis.

Third, through the **senior category**, this challenge allows qualified researchers to demonstrate their knowledge and share their know-how and expertise on the practical problems. This also gives them the opportunity to establish partnerships with industrial companies.

Last, a **scientific prize** dedicated to qualitative submissions is proposed

ROADEF | EURO THE ASSOCIATION OF EUROPEAN OPERATIONAL RESEARCH SOCIETIES | SAINT-GOBAIN

Home | Roadef.org | Contact

www.roadef.org

Exemplos: i) ROADEF 2009

ROADEF 2009 Challenge: Disruption Management for Commercial Aviation

M. Palpant, M. Boudia, C.-A. Robelin, S. Gabteni, F. Laburthe

Amadeus S.A.S., Operations Research Division
485 route du pin montard, 06902 Sophia Antipolis Cedex, France
mireille.palpant@amadeus.com, mourad.boudia@amadeus.com,
charles-antoine.robelin@amadeus.com, semi.gabteni@amadeus.com,
francois.laburthe@amadeus.com

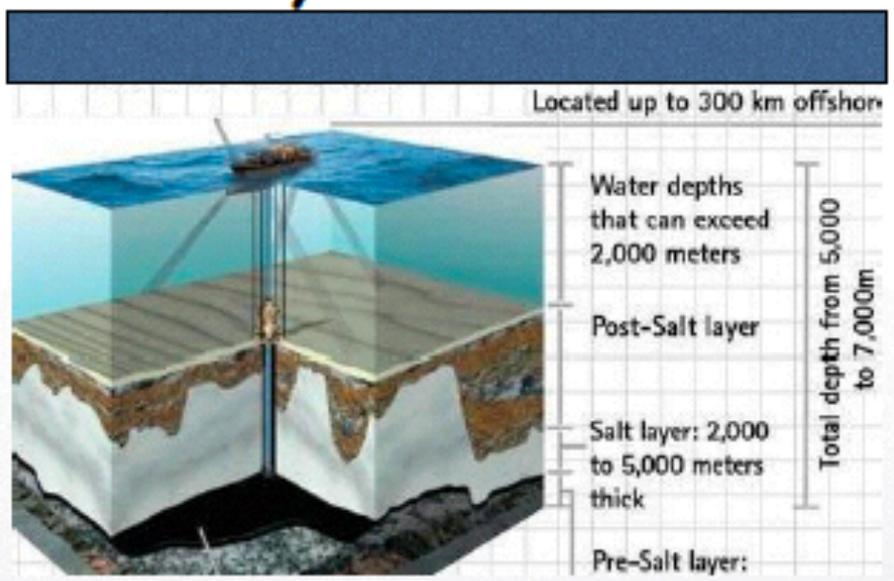
The objective function includes parameters related to additional costs or gains due to the modification of the flight plan (operating costs of added flights, deduction of operating costs for cancelled flights, costs associated with delays and cancellations of flights included in the original schedule), as well as a measure of the disutility to passengers. The objective is to minimize a weighted sum of those factors.

Exemplos: ii) ROADEF 2005

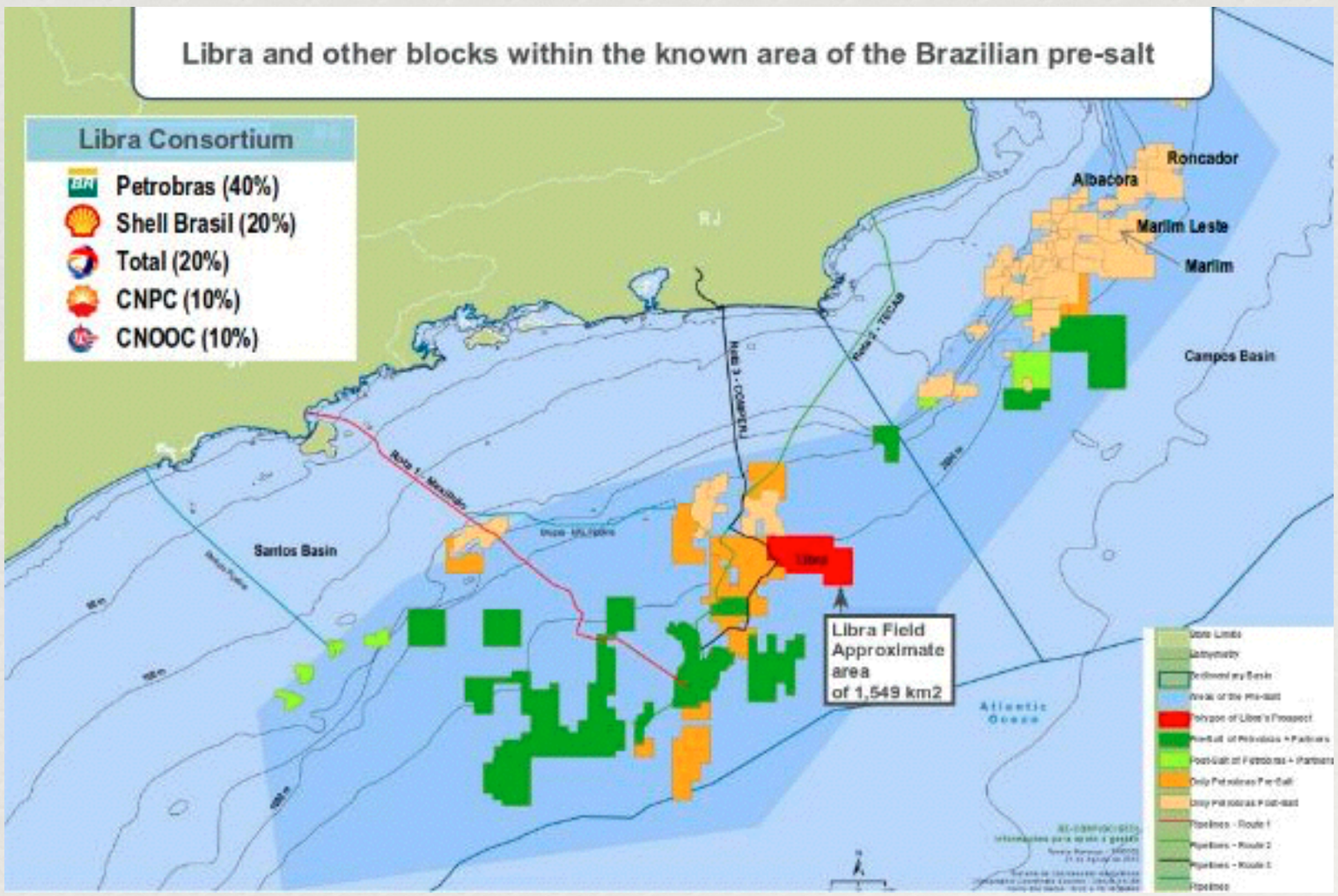
RENAULT

Exemplos: ii) SIPROV

Petroleum exploitation in the pre-salt layer



Exemplos: ii) SIPROV



Tree search algorithms

Basic idea:

offline, simulated exploration of state space

by generating successors of already-explored states

(a.k.a. expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

AIMA - UC Berkeley

Até a próxima aula a turma deverá se dividir em grupos de quatro alunos para o primeiro trabalho. Os grupos devem escolher uma aplicação e propor um programa inteligente (usando somente regras e problem solving) para ela. A solução deve ser programada em Prolog usando o sistema adotado no curso ou qualquer outro.

Até a próxima aula!