



Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 7- Algoritmos de busca informada

Prof. José Reinaldo Silva

reinaldo@usp.br





Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será dividido em "milestones":

Setembro 2018						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

7: Independência do Brasil 22: Início da primavera
02 - Quarto Minguante 09 - Lua Nova 16 - Quarto Crescente 24 - Lua Cheia

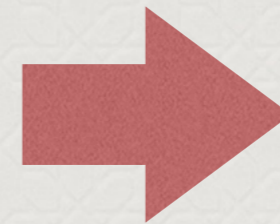
Outubro 2018						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

21: Início do horário de verão 12: Nsa. Sra. Aparecida 15: Dia dos Professores
02 - Quarto Minguante 09 - Lua Nova 16 - Quarto Crescente 24 - Lua Cheia 31 - Quarto Minguante



Uma outra forma seria gerar o espaço de estados enquanto se busca a solução

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.



estrutura



Search Techniques for Artificial Intelligence

Search is a central topic in Artificial Intelligence. This part of the course will show why search is such an important topic, present a general approach to representing problems to do with search, introduce several search algorithms, and demonstrate how to implement these algorithms in Prolog.

- Motivation: Applications and Toy Examples
- The State-Space Representation
- Uninformed Search Techniques:
 - Depth-first Search (several variations)
 - Breadth-first Search
 - Iterative Deepening
- Best-first Search with the A* Algorithm

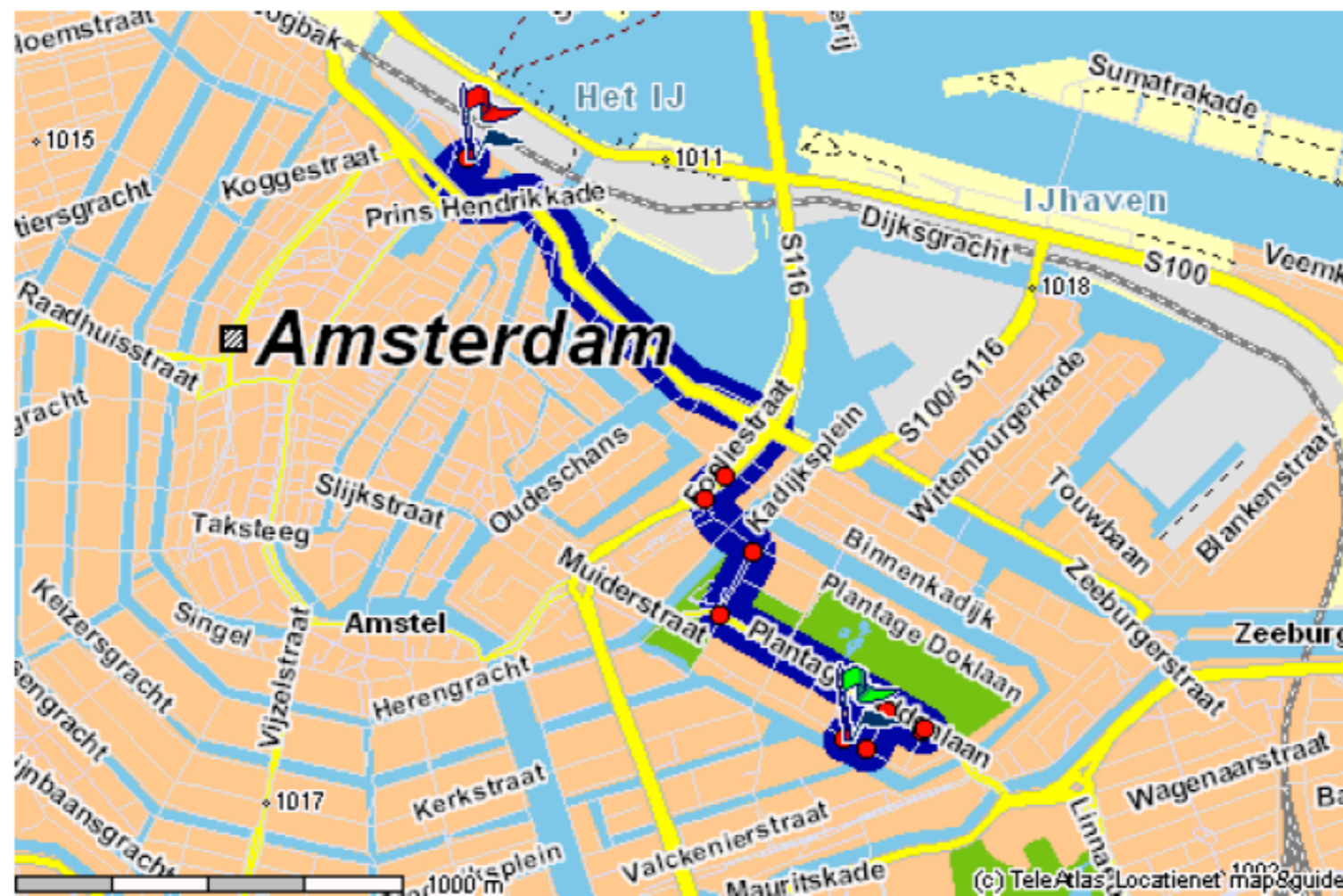


Exemplo de problemas que podem ser resolvidos com busca:

Search Techniques

LP&ZT 2005

Route Planning



Ulle Endriss (ulle@illc.uva.nl)

2









Aplicações práticas de “intelligent routing planning”

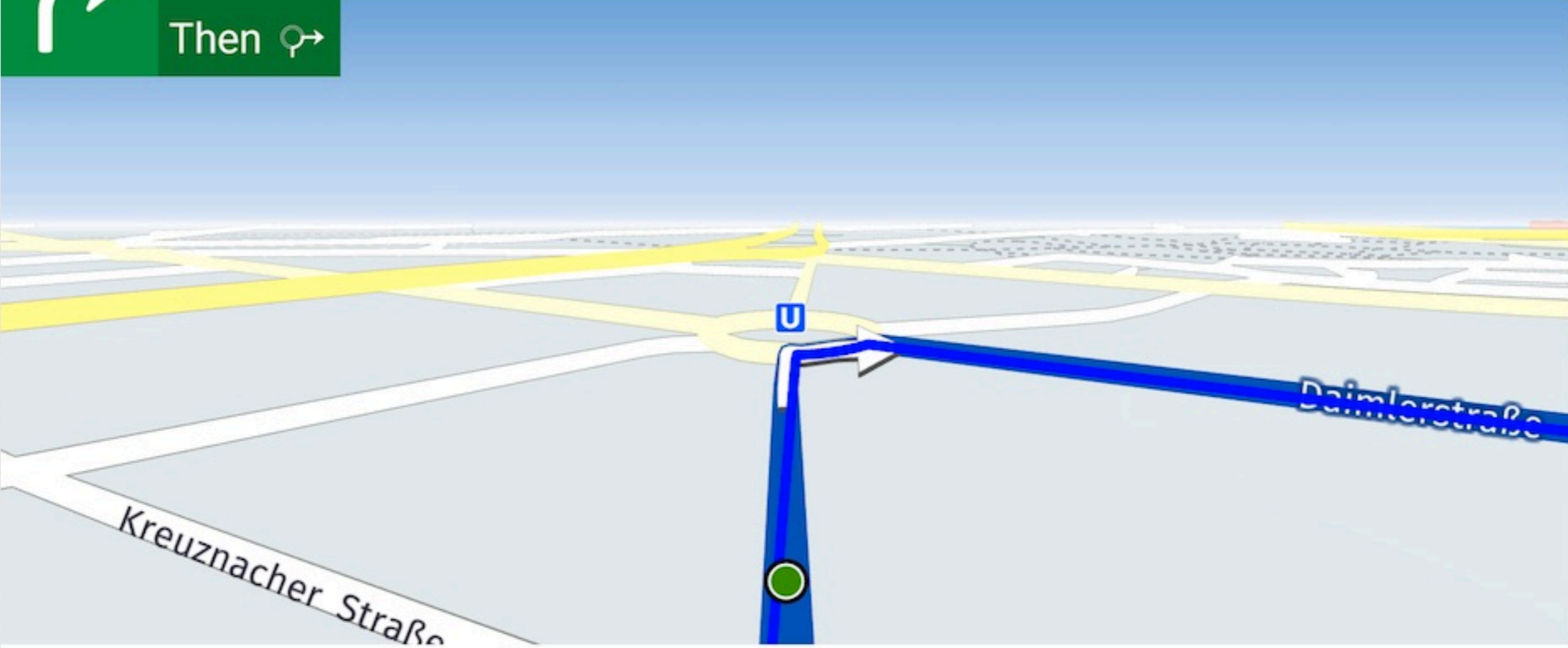


A medium-sized town somewhere in Germany, where residents separate their rubbish into glass, paper, food waste, plastic, old clothes and various other categories. All the bins are fitted with sensors to measure the fill level, and all the data is transmitted to a central server. We're talking here about thousands of bins. Millions, even, in large urban areas. Reliably transmitting signals, day in, day out. But who's supposed to process this information? The sheer amount of data generated would be way too much for even the brainiest human being to cope with.



Telekom.de      ...

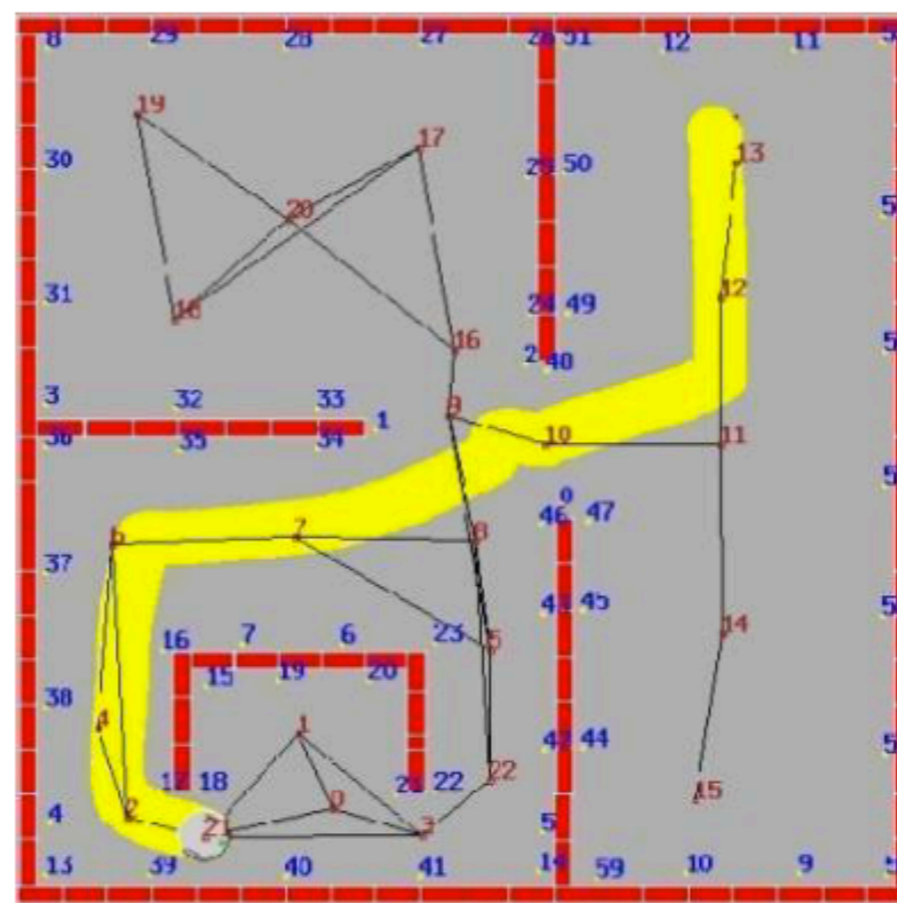
Turn right
Then 



END 2 min • 786 m • 14:33



Robot Navigation

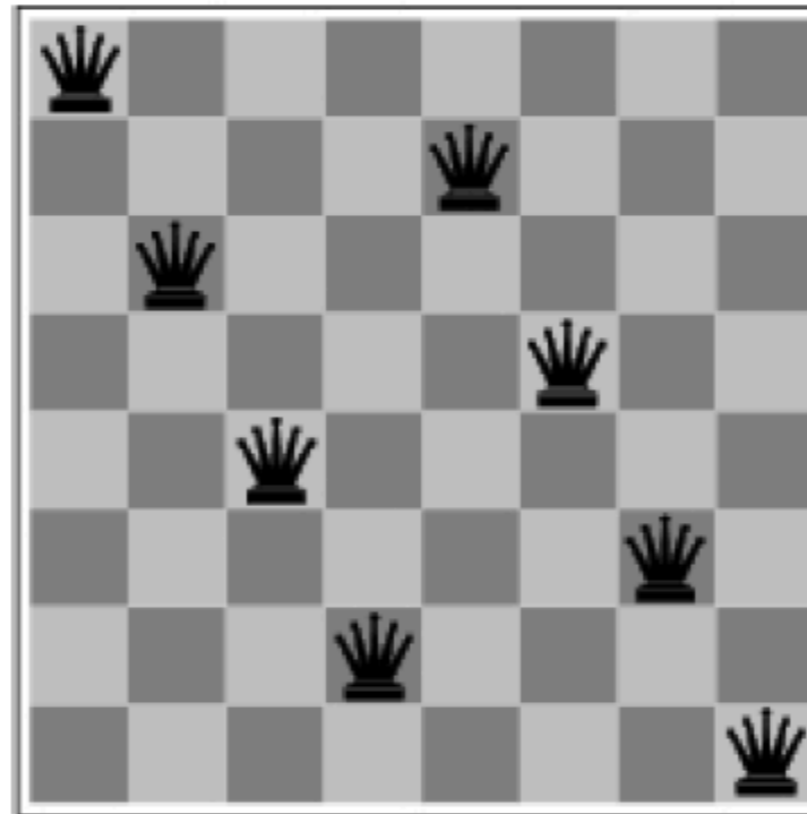


Source: <http://www.ics.forth.gr/cvrl/>



Eight-Queens Problem

Arrange eight queens on a chess board in such a manner that none of them can attack any of the others!



Source: Russell & Norvig, *Artificial Intelligence*

The above is *almost* a solution, but not quite ...



Eight-Puzzle

Yet another puzzle ...

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Source: Russell & Norvig, *Artificial Intelligence*



Exponential Complexity

In general, in Computer Science, anything exponential is considered bad news. Indeed, our simple search techniques will usually not work very well (or at all) for larger problem instances.

Suppose the branching factor is $b = 4$ and suppose it takes us 1 millisecond to check one node. What kind of depth bound would be feasible to use in depth-first search?

Depth	Nodes	Time
2	21	0.021 seconds
5	1365	1.365 seconds
10	1398101	23.3 minutes
15	1431655765	16.6 days
20	1466015503701	46.5 years



Iterative Deepening

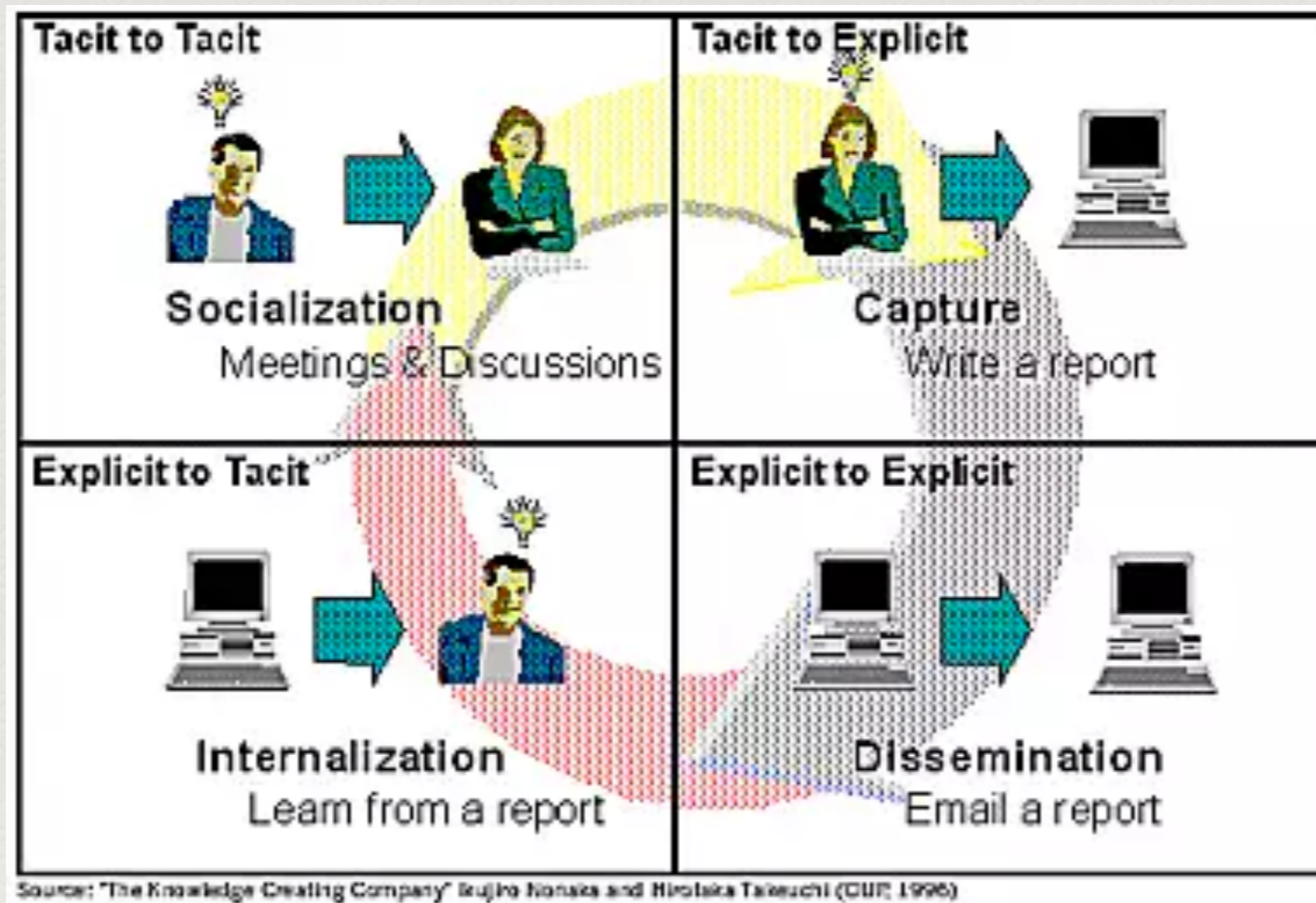
We can specify the iterative deepening algorithm as follows:

- (1) Set n to 0.
- (2) Run depth-bounded depth-first search with bound n .
- (3) Stop and return answer in case of success;
increment n by 1 and go back to (2) otherwise.

However, in Prolog we can implement the same algorithm also in a more compact manner ...



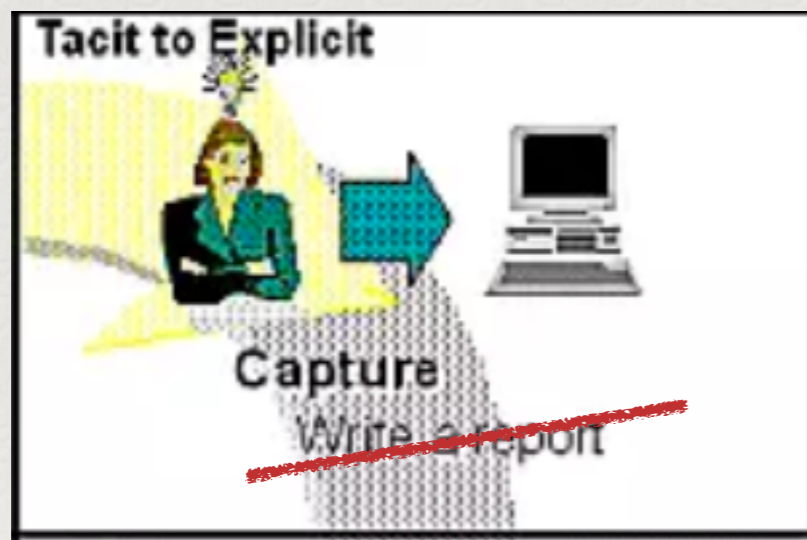
Introduzindo heurísticas





Heuristic search

In computer science, artificial intelligence, and mathematical optimization, a **heuristic** (from Greek εὕρισκω "I find, discover") is a technique designed for **solving a problem** more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, **accuracy**, or **precision** for speed. In a way, it can be considered a shortcut.



find a function



O uso de funções de avaliação

- Vamos assumir que existe uma função (de avaliação) heurística que pode ser usada para escolher o melhor nó para expandir a busca.
- Assim, podemos expandir um nó n que tenha o menor valor depois de aplicada a função de avaliação $f(n)$. Os “empates” podem ser resolvidos arbitrariamente.
- A busca termina quando o nó a ser expandido for o nó objetivo.



O algoritmo “best-first”

Portanto, uma possibilidade de algoritmo com melhor performance que qualquer um daqueles da “busca não-informada” seria aplicar o breadth-first, mas, escolhendo como primeira escolha não o primeiro nó a esquerda mas o melhor segundo a função de avaliação.



Voltando ao projeto dos grupos...

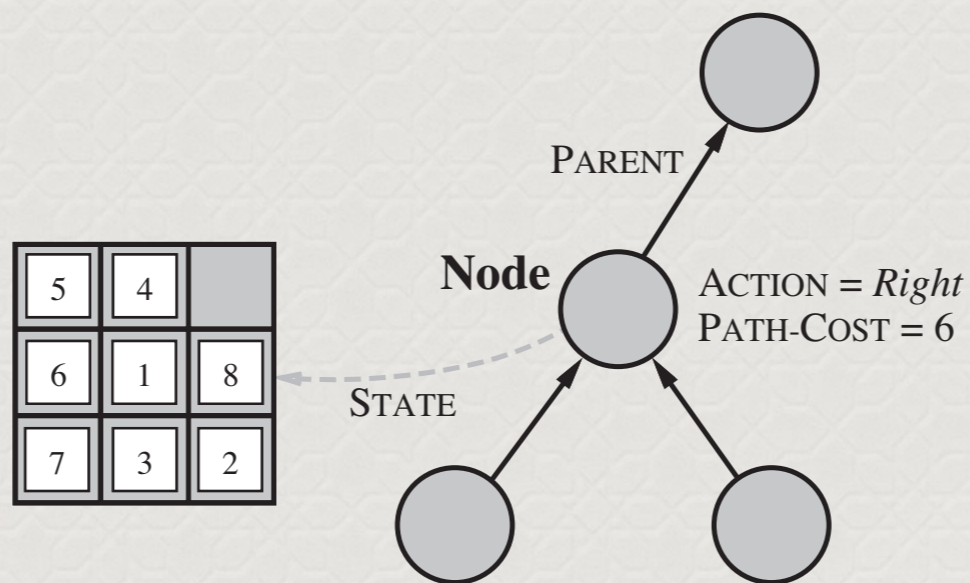
2	8	3
1	6	4
7		5

Vamos supor que temos acima o estado inicial do nosso jogo automático de tiles (8-puzzle) e queremos agora achar uma função de avaliação para fazer o computador resolver automaticamente o enigma. Já vimos em aulas anteriores que o "estado" do sistema é dado pela tabela de tiles com o posicionamento de todos eles. Podemos então dividir a função de avaliação em duas partes: uma que avalia o "custo", isto é, quanto mais profundo o nó, maior o custo; e outra que avalia o grau de desorganização da tabela de tiles.



A tabela de tiles

A tabela de tiles tem NOVE posições e o conteúdo de cada uma destas posições é um inteiro de zero a oito, onde o zero significa o tile para onde se pode mover os demais vizinhos.

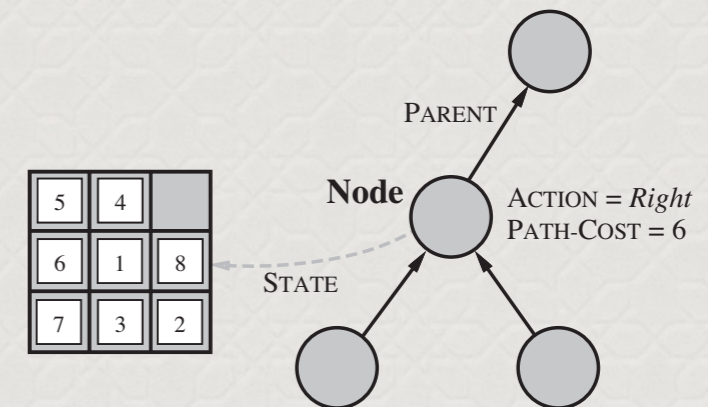




A função de avaliação traduz portanto o fato que gostaríamos de encontrar uma solução “rápida”, isto é, em baixa profundidade da árvore de busca, e que quanto mais nos aproximamos do objetivo mais “organizada” fica a tabela de tiles, isto é, o número de tiles fora do lugar diminui. Assim, a função de avaliação é

$$g(n) = p + h(n)$$

onde o fator p traduz a profundidade do nó n na árvore de busca e $h(n)$ é o número de tiles fora do lugar.





2	8	3
1	6	4
7		5

$$g(0) = 0 + 4$$



1	2	3
8		4
7	6	5

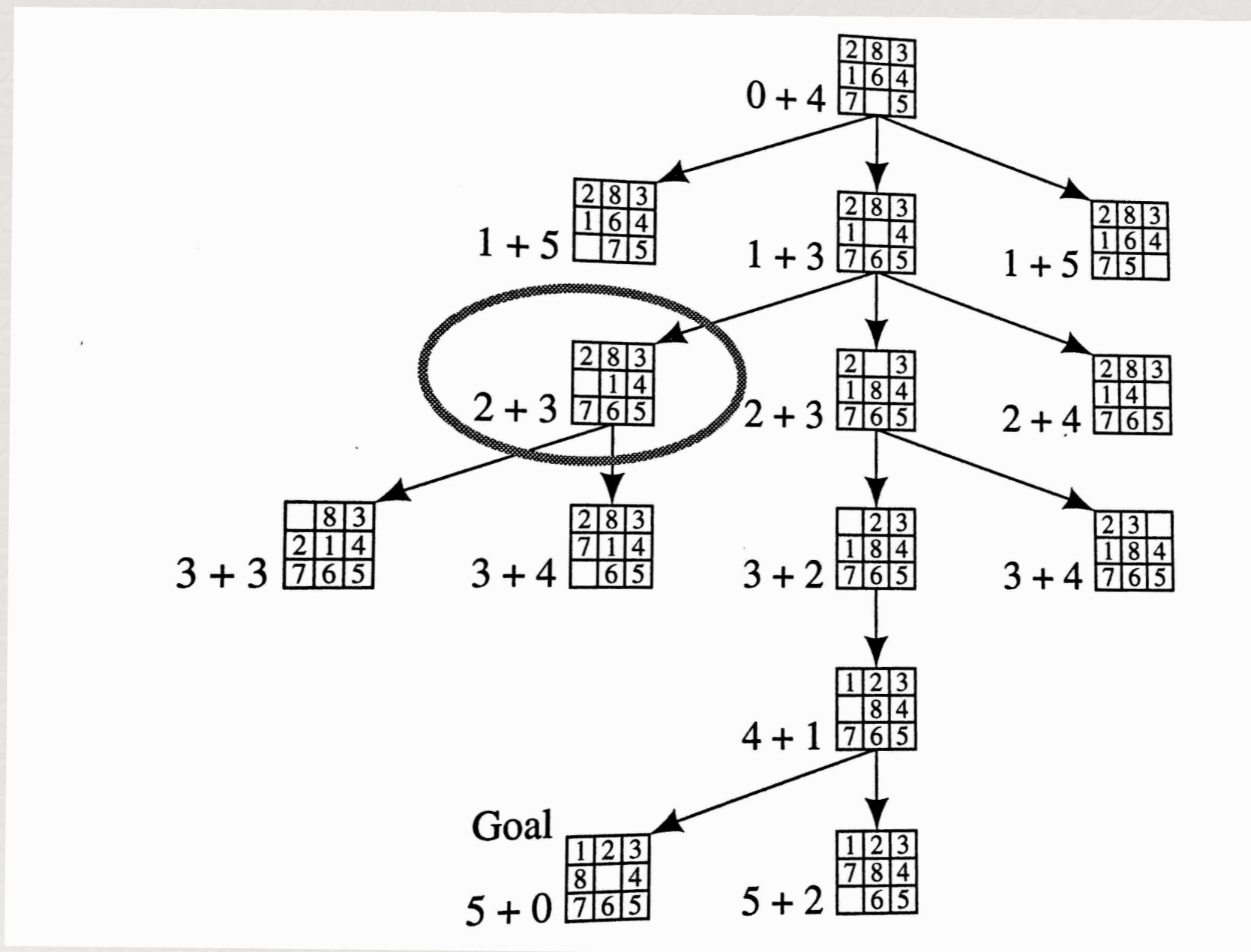
estado final

comparando o estado inicial (a raiz da árvore de busca) com o estado final mostrado à direita, temos que os tiles 1, 2, 8 e 6 estão fora do lugar, portanto neste estado (o nível zero ou raiz, de profundidade zero) $h(0) = 4$, e a função de avaliação é $g(0) = 0 + 4 = 4$

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



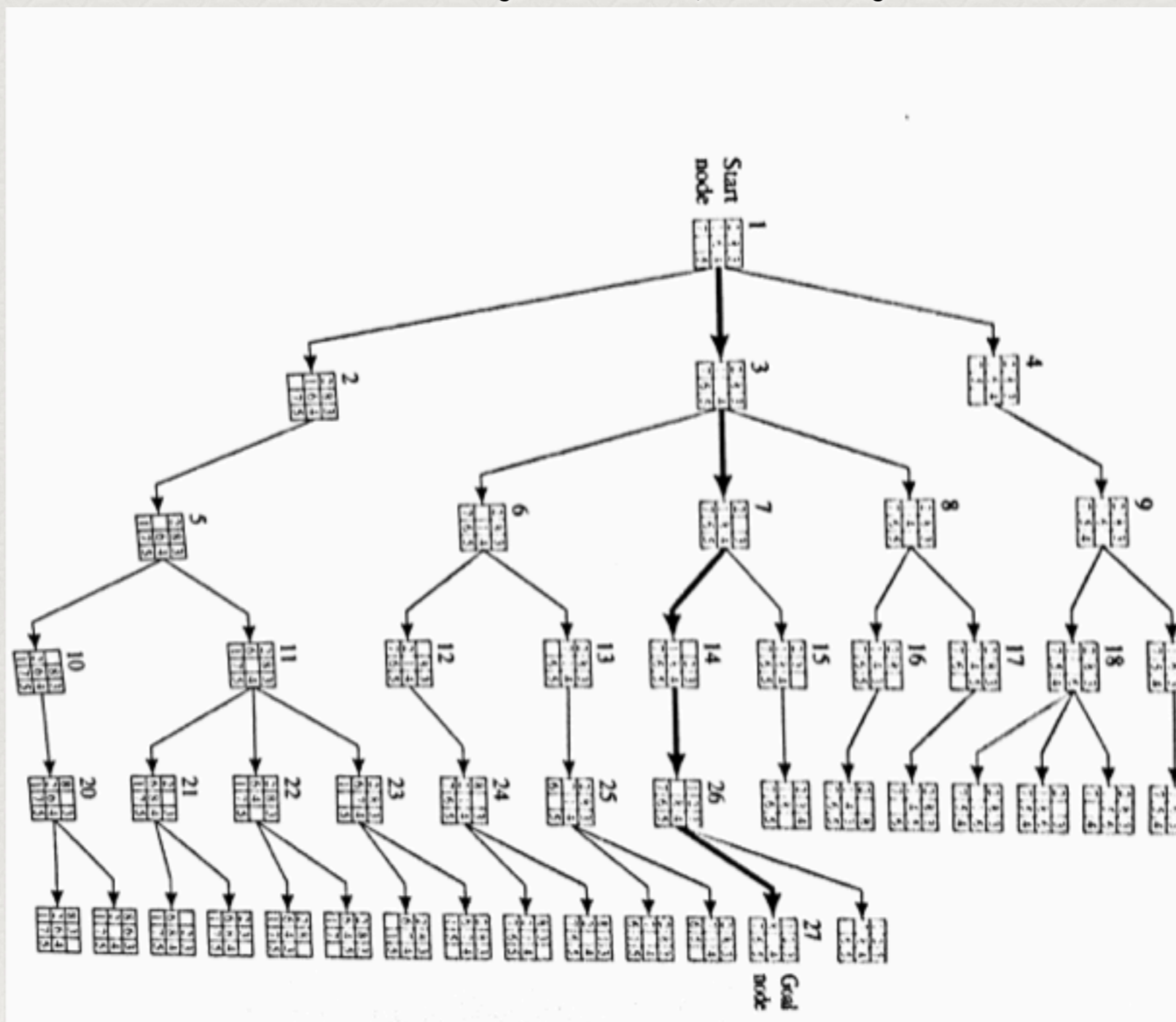
Árvore de busca e a busca informada



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



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





Best-first Search and Heuristic Functions

- For both *depth-first* and *breadth-first* search, which node in the search tree will be considered next only depends on the structure of the tree.
- The rationale in *best-first* search is to expand those paths next that seem the most “promising”. Making this vague idea of what may be promising precise means defining *heuristics*.
- We fix heuristics by means of a *heuristic function* h that is used to *estimate* the “distance” of the current node n to a goal node:

$$h(n) = \text{estimated cost from node } n \text{ to a goal node}$$

Of course, the definition of h is highly application-dependent. In the *route-planning* domain, for instance, we could use the straight-line distance to the goal location. For the *eight-puzzle*, we might use the number of misplaced tiles.



Best-first Search Algorithms

There are of course many different ways of defining a heuristic function h . But there are also different ways of *using* h to decide which path to expand next; which gives rise to different best-first search algorithms.

One option is *greedy* best-first search:

- expand a path with an end node n such that $h(n)$ is *minimal*

Breadth-first and depth-first search may also be seen as special cases of best-first search (which do not use h at all):

- Breadth-first: expand the (leftmost of the) *shortest* path(s)
- Depth-first: expand the (leftmost of the) *longest* path(s)



The A* Algorithm

The central idea in the so-called *A* algorithm* is to guide best-first search both by

- the estimate to the goal as given by the heuristic function h and
- the cost of the path developed so far.

Let n be a node, $g(n)$ the cost of moving from the initial node to n along the current path, and $h(n)$ the estimated cost of reaching a goal node from n . Define $f(n)$ as follows:

$$f(n) = g(n) + h(n)$$

This is the *estimated cost of the cheapest path through n* leading from the initial node to a goal node. *A* is the best-first search algorithm that always expands a node n such that $f(n)$ is minimal.*



Até a próxima aula!