Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 8- Algoritmo de busca A*

Prof. José Reinaldo Silva

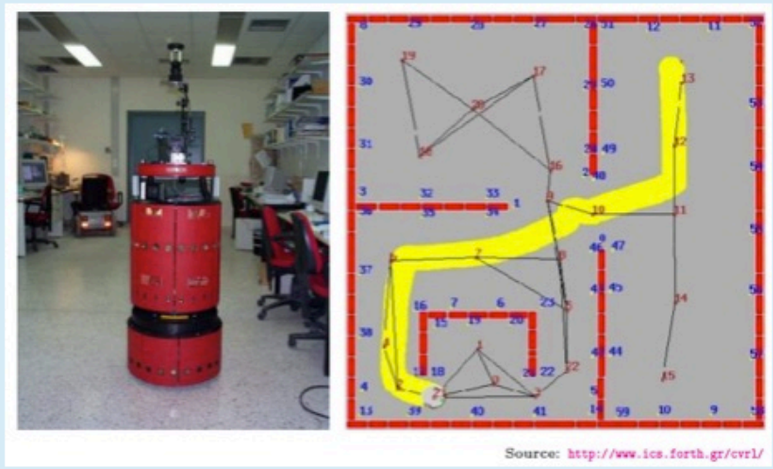
reinaldo@usp.br

Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será divido em "milestones":

Upload do trabalho em grupo

Algoritmos de busca informada



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

Vamos a partir desta aula mudar o foco da discussão sobre os algoritmos de busca usados em IA da busca clássica não-informada para a busca informada, onde se pode usar heurísticas obtidas de conhecimento tácito ou explicito para orientar a escolha do caminho na árvore de busca. Esperamos portanto obter uma melhor performance, comparado aos métodos clássicos e assim justificar o uso de IA para problemas que têm um custo proibitivo por métodos convencionais, ou mesmo que não chegam a uma resposta, utilizando métodos otimizados. A barganha é portanto obter uma solução próxima do ótimo mas em um tempo aceitável.

Começaremos a discussão pelo best-first, e aplicaremos este algoritmo para um exemplo já bem conhecido que o 8-Puzzle, ou o jogo dos tiles, que é o exercício em grupo que estamos considerando. Aproveito para lembrar que o milestone para este exercício, que seria hoje foi adiado de uma semana, em função de termos perdido a aula passada.

Aula7

Transparencias da Aula 7

Search Techniques

Milestone 1

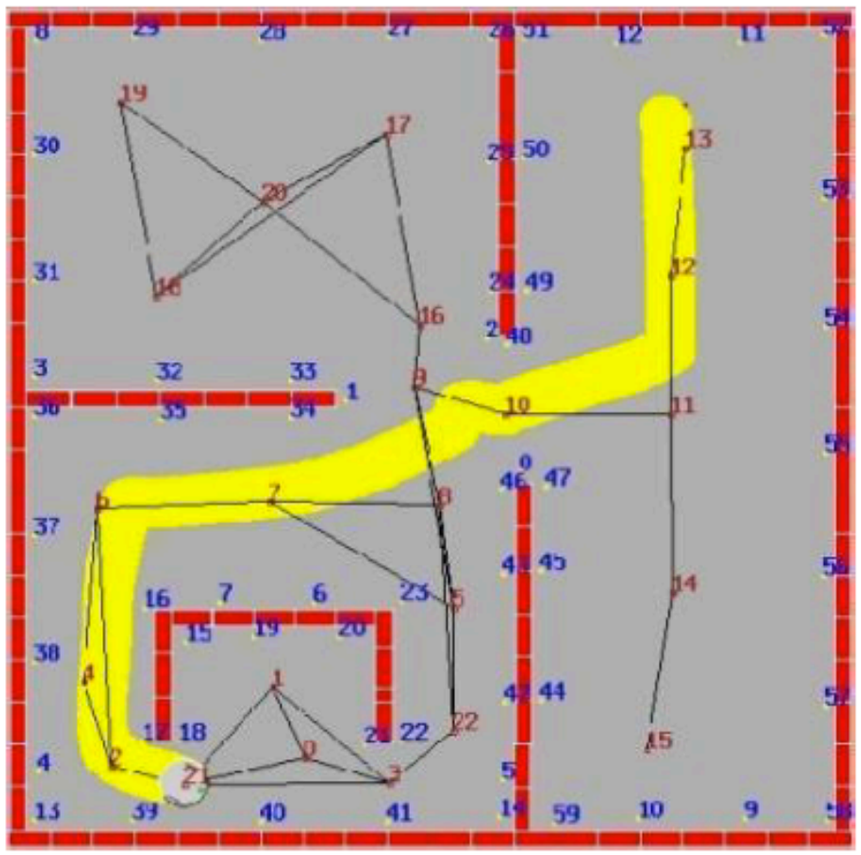
Este é o milestone para o primeiro trabalho em grupo. Faça o upload até a data limite (10 de outubro próximo) da descrição da heuristica que deverá usar no seu trabalho com a respectiva justificativa de performance (porque e como irá acelerar a busca). A coleta funciona até a meianoite do dia 10/10.

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

Robot Navigation



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

Estratégias de Busca Informada

Análise de custo

$f(x) = g(x)$

Heurística

$f(x) = h(x)$

Algorítmo A*

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

Heuristic search

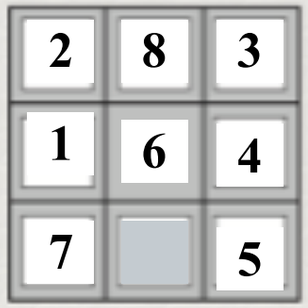
In computer science, artificial intelligence, and mathematical optimization, a **heuristic** (from Greek εὑρίσκω "I find, discover") is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any 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

Heuristic-guided Search

- Our complexity analysis of the various basic search algorithms has shown that they are unlikely to produce results for slightly more complex problems than we have considered here.
- In general, there is no way around this problem. In practice, however, good *heuristics* that tell us which part of the search tree to explore next, can often help to find solutions also for larger problem instances.
- In this final chapter on search techniques for AI, we are going to discuss one such heuristic, which leads to the well-known A* algorithm.

Voltando ao projeto dos grupos...



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 função de seleção

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

custo para chegar ao nó n da árvore de busca

custo estimado para chegar ao nó objetivo

The A* Algorithm

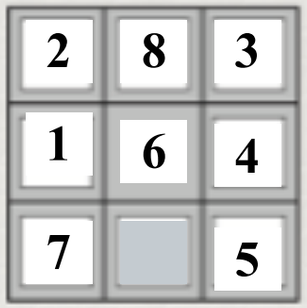
The central idea in the so-called *A* algorithm* is to guide best-first search both by

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

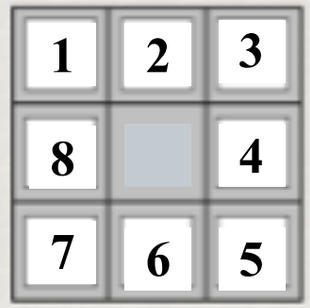
Let n be a node, $g(n)$ the cost of moving from the initial node to n along the current path, and $h(n)$ the estimated cost of reaching a goal node from n. Define $f(n)$ as follows:

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

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.*



g(0) = 0 + 4

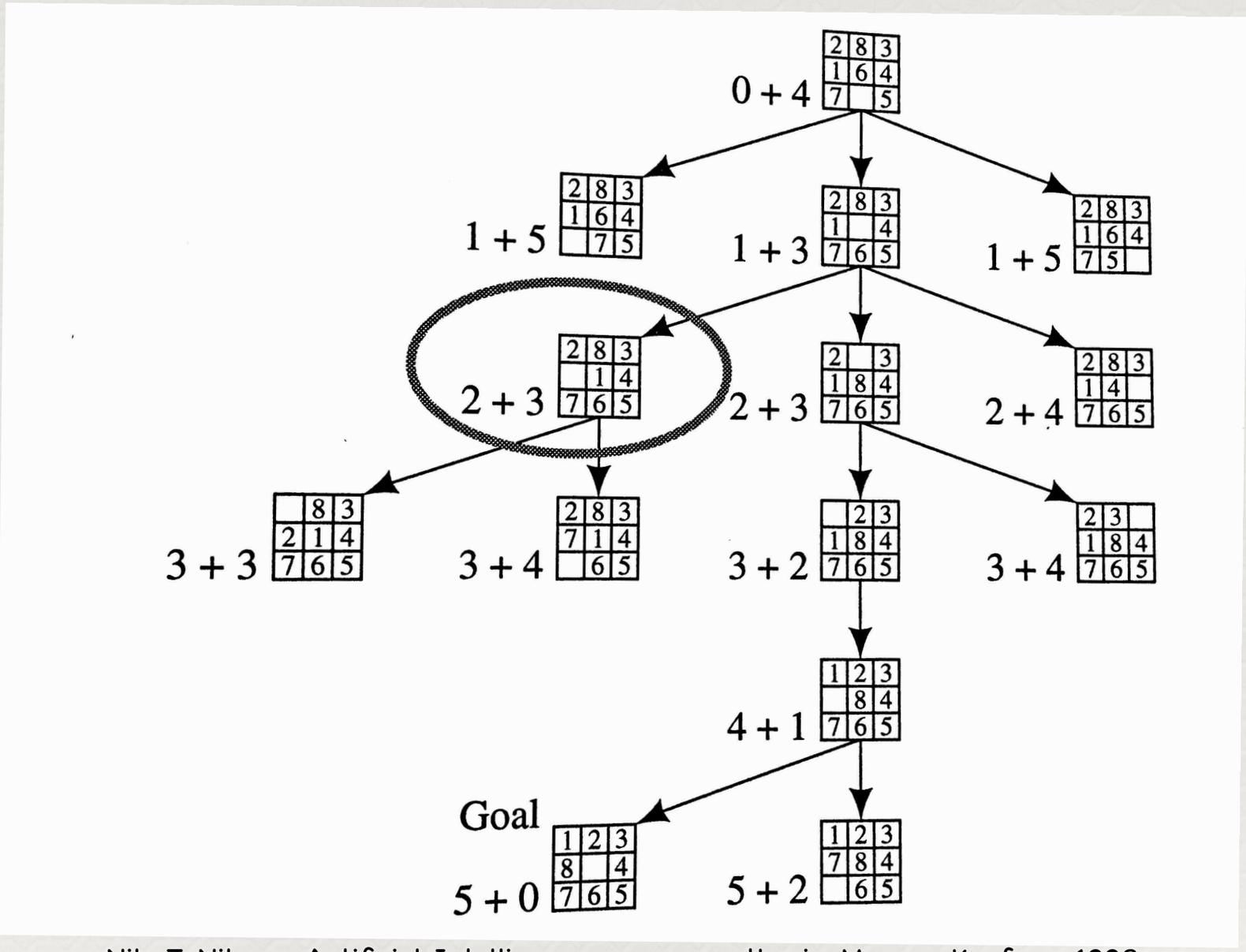


estado final

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 Kaufann, 1998

Árvore de busca e a busca informada



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

Critérios para a escolha da heurística

O objetivo do A* é guiar a busca para a solução de modo a atingir o estado alvo mais rapidamente e com menor custo. Portanto a heurística escolhida deve ser admissível.

Uma heurística h(x) é dita admissível se para cada nó n da árvore de busca $h(n) \leq h^*(n)$, ond onde h(n) é o custo estimado para atingir a solução e $h^*(n)$ é o custo real. Portanto, o algoritmo não deve superestimar o custo para atingir o objetivo.

formalmente...

Teorema: Se h(n) é admissível, o algoritmo A* baseado em árvore de busca é otimizante, isto é, pode chegar a um caminho ótimo de solução.

Comparando diferentes propostas de heurísticas

Sejam duas heurísticas admissíveis $h_1(x)$ e $h_2(x)$. Ambas devem ter como limite o valor real $h^*(x)$ e tendem para este limite. Portanto, se para qualquer nó x $h_2(x) \geq h_1(x)$, significa que $h_2(x)$ está mais próxima do limite e dizemos que h_2 "domina" h_1.

A heurística dominante deve gerar uma busca melhor, e expandir um número menor de nós alternativos para a busca.

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:

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

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.*

Dado um nó n, vamos definir o custo para ir deste nó para o seu sucessor n', realizando a ação a, como sendo C(n, a, n').

Uma heurística h é dita consistente, se para qualquer sucessor n' do nó n, vale a seguinte inequação:

$$h(n) \leq C(n, a, n') + h(n')$$

Teorema: Se uma heurística h é consistente, então usando A* em uma busca em grafo, isto é, uma busca onde se checa o sucessor já foi visitado antes para evitar loops, é ótima.

Portanto o A* é um algoritmo de busca informada otimizante, que, em situações práticas pode ter uma abordagem heurística relaxada para ter uma reposta mais rápida.

```
/* 8_puzzle.pl */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%   A* Algorithm
%%%
%%%
%%%   Nodes have form    S#D#F#A
%%%         where S describes the state or configuration
%%%               D is the depth of the node
%%%               F is the evaluation function value
%%%               A is the ancestor list for the node

:- op(400,yfx,'#').    /* Node builder notation */

solve(State,Soln) :- f_function(State,0,F),
             search([State#0#F#[]],S), reverse(S,Soln).

f_function(State,D,F) :- h_function(State,H),
               F is D + H.

search([State#_#_#Soln|_], Soln) :- goal(State).
search([B|R],S) :- expand(B,Children),
           insert_all(Children,R,Open),
           search(Open,S).

insert_all([F|R],Open1,Open3) :- insert(F,Open1,Open2),
                  insert_all(R,Open2,Open3).
insert_all([],Open,Open).

insert(B,Open,Open) :- repeat_node(B,Open), ! .
insert(B,[C|R],[B,C|R]) :- cheaper(B,C), ! .
insert(B,[B1|R],[B1|S]) :- insert(B,R,S), !.
insert(B,[],[B]).

repeat_node(P#_#_#_, [P#_#_#_|_]).

cheaper( _#_#F1#_ , _#_#F2#_ ) :- F1 < F2.

expand(State#D#_#S,All_My_Children) :-
    bagof(Child#D1#F#[Move|S],
        (D1 is D+1,
         move(State,Child,Move),
         f_function(Child,D1,F)),
        All_My_Children).
```

left(A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J).
left(A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J).
left(A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J).
left(A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J).
left(A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J).
left(A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I).

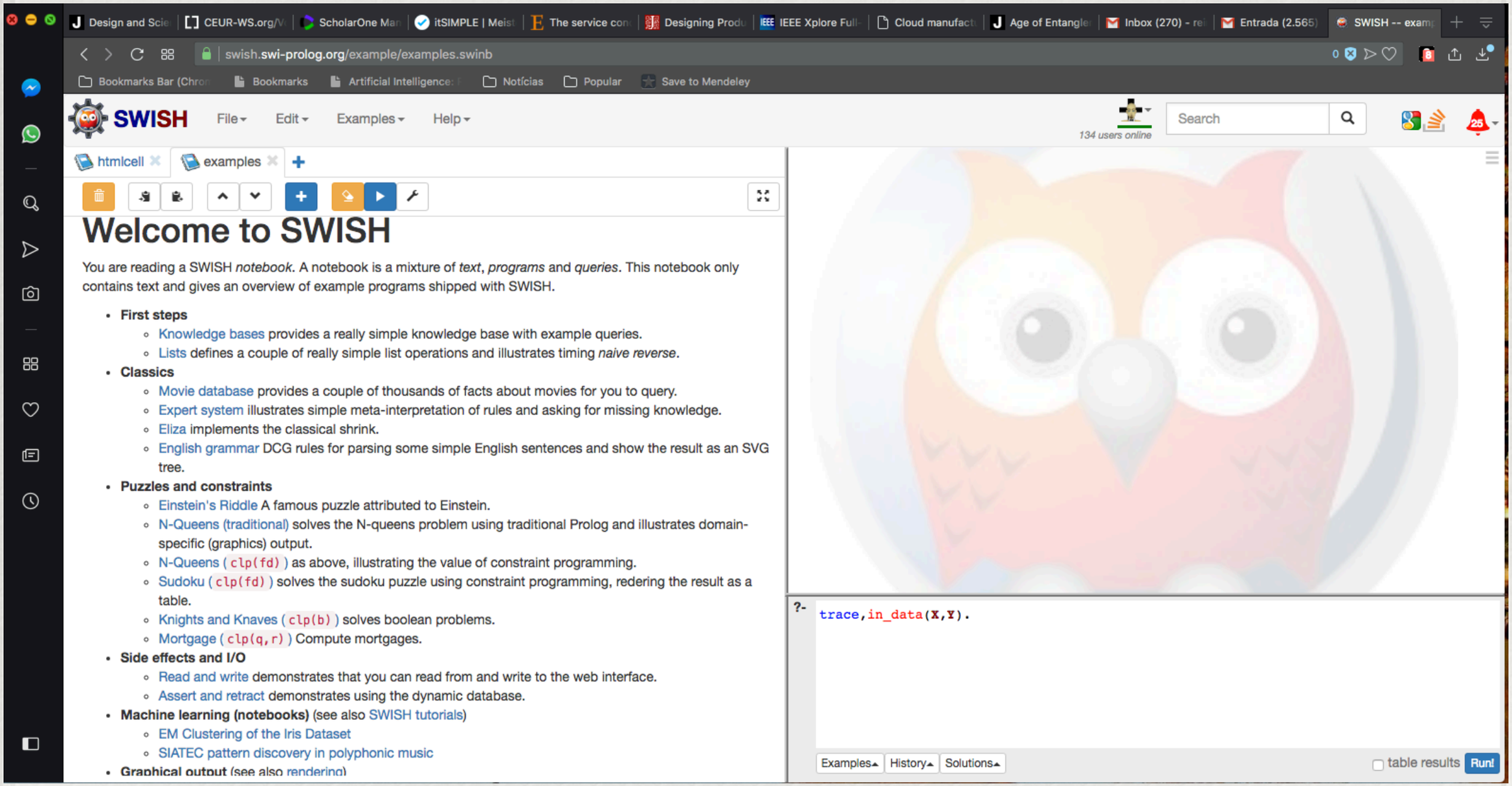
up(A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J).
up(A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J).
up(A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J).
up(A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J).
up(A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J).
up(A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F).

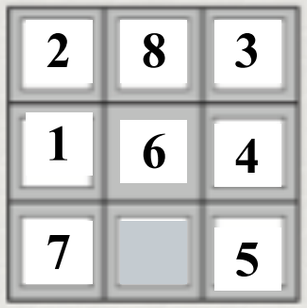
right(A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J).
right(A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J).
right(A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0).
right(0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J).
right(A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J).
right(A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J).

down(A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J).
down(A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J).
down(A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0).
down(0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J).
down(A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J).
down(A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J).

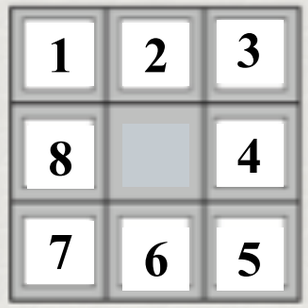
A competição entre equipes

No dia 31, segundo o nosso cronograma, teremos o desfecho do trabalho em grupo e faremos isso na forma de uma competição onde todos os grupos deverão resolver o mesmo problema dado como estado inicial. Todos devem ter o seu programa instalado no sistema Swish e pronto para rodar sua implementação do A*.


SWISH
File
Edit
Examples
Help
Search
134 users online
htmlcell
examples
Welcome to SWISH
You are reading a SWISH notebook. A notebook is a mixture of text, programs and queries. This notebook only contains text and gives an overview of example programs shipped with SWISH.
First steps
Knowledge bases provides a really simple knowledge base with example queries.
Lists defines a couple of really simple list operations and illustrates timing naive reverse.
Classics
Movie database provides a couple of thousands of facts about movies for you to query.
Expert system illustrates simple meta-interpretation of rules and asking for missing knowledge.
Eliza implements the classical shrink.
English grammar DCG rules for parsing some simple English sentences and show the result as an SVG tree.
Puzzles and constraints
Einstein's Riddle A famous puzzle attributed to Einstein.
N-Queens (traditional) solves the N-queens problem using traditional Prolog and illustrates domain-specific (graphics) output.
N-Queens (clp(fd)) as above, illustrating the value of constraint programming.
Sudoku (clp(fd)) solves the sudoku puzzle using constraint programming, redering the result as a table.
Knights and Knaves (clp(b)) solves boolean problems.
Mortgage (clp(q,r)) Compute mortgages.
Side effects and I/O
Read and write demonstrates that you can read from and write to the web interface.
Assert and retract demonstrates using the dynamic database.
Machine learning (notebooks) (see also SWISH tutorials)
EM Clustering of the Iris Dataset
SIATEC pattern discovery in polyphonic music
Graphical output (see also rendering)
?-
trace,in_data(X,Y).
Examples
History
Solutions
table results
Run!



g(0) = 0 + 4



estado final

O estado final será o mesmo que usamos na aula passada no exemplo de resolução da busca baseada no algoritmo Best-first.

O procedimento

- Todos devem estar preparados com o Swish e o seu algoritmo, mas sem tocar no teclado ou no mouse;
- O estado inicial será colocado em um slide;
- Todos esperam o "sinal de largada" para começar o processo;
- Devem inserir o estado inicial no programa e resolvê-lo;
- Chegando ao estado final o programa deve descrever este estado e colocar na tela o custo da busca (soma de g(x)).
- com esta informação na saída do programa a equipe faz um sinal de termino e o tempo é anotado.

Após a competição

Após a competição cada equipe deve fazer o upload no sistema e-disciplinas de um texto que descreve a heurística (g(x) e h(x)), suas propriedades, e uma listagem do código Prolog. A nota final será a média da nota atribuída a este documento e a nota da competição.

A nota da competição

A nota da competição será a composição da classificação por tempo (de zero a cinco), e cinco pontos se a equipe conseguiu resolver mesmo que em um tempo mais longo, e zero se não fechou o processo de busca.

Trabalho em grupo

Os grupos estão já definidos e registrados no e-disciplinas. A configuração é a seguinte:

Grupo 1:
Alyson Akio Haro
Alexandre Inoue
Eduardo Kose
Vitor Fukuda

Grupo 2:
Alex Majima
Gabriel Ferreira
Lucas de Angelis
Thiago Ferraz

Grupo 3:
Gabriel Tutia
Lucas Palopoli
Pedro dos Santos Melo
Samuel Monção

Grupo 4:
Gustavo Novello
Luiz Guilherme Sabino
Alessandro Ezequiel Junior
Gabriel Negre

Grupo 5:
Gabriel Yida
Vinicius Santiago
Danilo Polidoro

Grupo 6:
Guilherme Aires de França
Henrique Peterlevitz
Wagner Geraldo Ferreira

Até a próxima aula!