Intelligent Systems

Ինտելեկտուալ տեղեկ. համակարգեր

NUACA/ՃՇՀԱՀ

2017

LECTURE 4

Informed Search Techniques

AIMA: CHAPTER 3.5-3.6

Problems

A **problem** can be formulated by specifying:

–The **state space** *("the set of all possible states we might get into") - վիճակների բազմություն*

–The **initial state** *("where we are") - սկզբնական վիճակ*

–A test for the **goal** *("are we where we want to be?") - արդյոք հասել ենք նպատակին*

- *Formally, we define a Boolean function* Goal *on the set of states*

–Available **actions** at any state *("what can we do?") - հասանելի գործողությունները բազմություն (ամեն վիճակում)*

- *Formally, we define a set-valued function* Actions *on the set of states*

–A **transition model** *("what will that achieve?") - անցում նոր վիճակի*

- *Formally, we define a state-valued function* Result *on the set of (state,action) pairs*

–A **cost measure** for sequences of actions *("is it worth it?") - գործողության արժեք*

- *Formally, we define a real-valued function* Cost *on the set of actions, or sequences*

Problems

From a problem specification we can derive:

–A **solution** *("a sequence of actions that take us to a goal state")* - *լուծում*

- *Formally, this is a **path** (ճանապարհ) in the state space, a sequence* $s_0,\ a_1, s_1,\ a_2, s_2,\ \ldots, a_n, s_n$ *where:*
- $s_0, s_1, s_2, \ldots, s_n$ *are **states**,* s_0 *is the **initial state**,* s_n *is a **goal state*** ($\mathrm{Goal}(s_n)$ = TRUE)
- $a_1,\ a_2, \ldots,\ a_n$ *are **actions**,*
- $a_i \in \mathrm{Actions}(s_{i-1})$ and $s_i = \mathrm{Result}(s_{i-1};\ a_i)$ for each 1 ≤ i ≤ n

– An **optimal solution** *("a solution with minimal cost")* – *օպտիմալ լուծում*

Tree-search

The basic idea is to explore the state space of a problem by generating the states that are reachable from the current state (known as expanding the state) and systematically examining them in some order

```
function TREE-SEARCH(problem) returns a solution or failure
    set frontier to {{node(initial state, empty path)}}

    repeat
        choose a node from frontier (and remove it)
        if this node contains a goal state then return solution

        expand this node, adding resulting nodes to the frontier

    until frontier is empty
    return failure
```

Graph-search

The basic idea is to explore the state space of a problem by generating **new** states that are reachable from the current state (known as expanding the state) and systematically examining them in some order

```
function GRAPH-SEARCH(problem) returns a solution or failure
    set frontier to {{node(initial state, empty path)}}
    set explored_set to an empty set (of states)
    repeat
        choose a node from frontier (and remove it)
        if this node contains a goal state then return solution
        add (state of) this node to the explored_set
        expand this node, adding resulting nodes to the frontier
            (if state not already in the explored_set or frontier)
    until frontier is empty
    return failure
```

Uninformed Search Techniques

- **Uninformed search** (also called blind search) indicates that the strategies have no additional information about states beyond that provided in the problem definition
- All they can do is generate successors and distinguish a goal state from a non-goal state.
- All search strategies are distinguished by the order in which order the nodes are expanded.

Informed Search Techniques

This section shows how an **informed search** strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.

best-first search - node is selected for expansion based on an **evaluation function**, *f(n).*

The node with lowest *f(n)* is expanded first.

Best-first search
Լավագույնը- սկզբում փնտրում

Idea: use an evaluation function *f(n)* for each node

- estimate of "desirability"
- Expand most desirable unexpanded node

Implementation:

Which data structure do we need to use?

The implementation of best-first graph search is identical to that for uniform-cost search (next slide), except for the use of *f* instead of *g* to order the priority queue.

Best-first search
Լավագույնը- սկզբում փնտրում

The choice of f determines the search strategy

Most best-first algorithms include as a component of *f* a **heuristic function**, denoted *h(n)*:

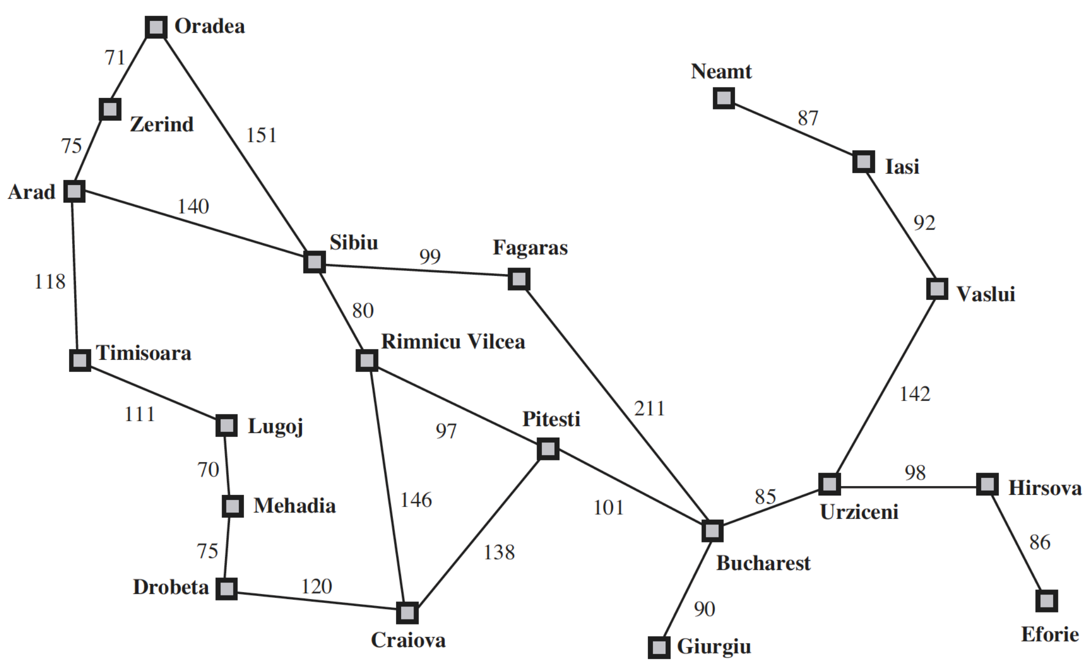
h(n) = estimated cost of the cheapest path from the state at node n to a goal state.

For now, let's consider h(n) to be arbitrary problem-specific functions that is:

- Non-negative: $h(n) \geq 0$
- if *n* is a goal node, then $h(n) = 0$.

Romania with step costs in km

Values of $h_{SLD}(n)$ —straight-line distances to Bucharest



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search
Ագահ լավագույնը- սկզբում փնտրում

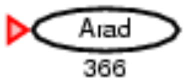
Evaluation function $f(n) = h(n)$ (heuristic)

$h(n)$ = estimate of cost from n to *goal*

e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

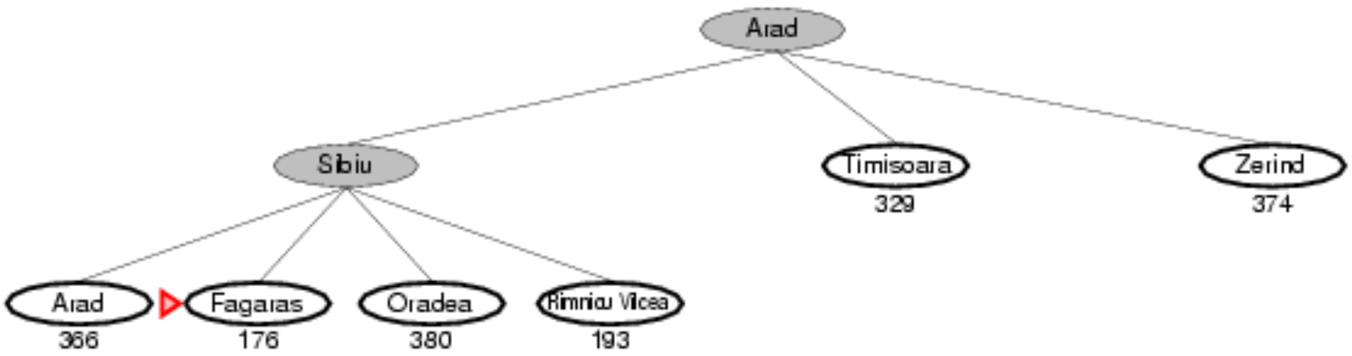
Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

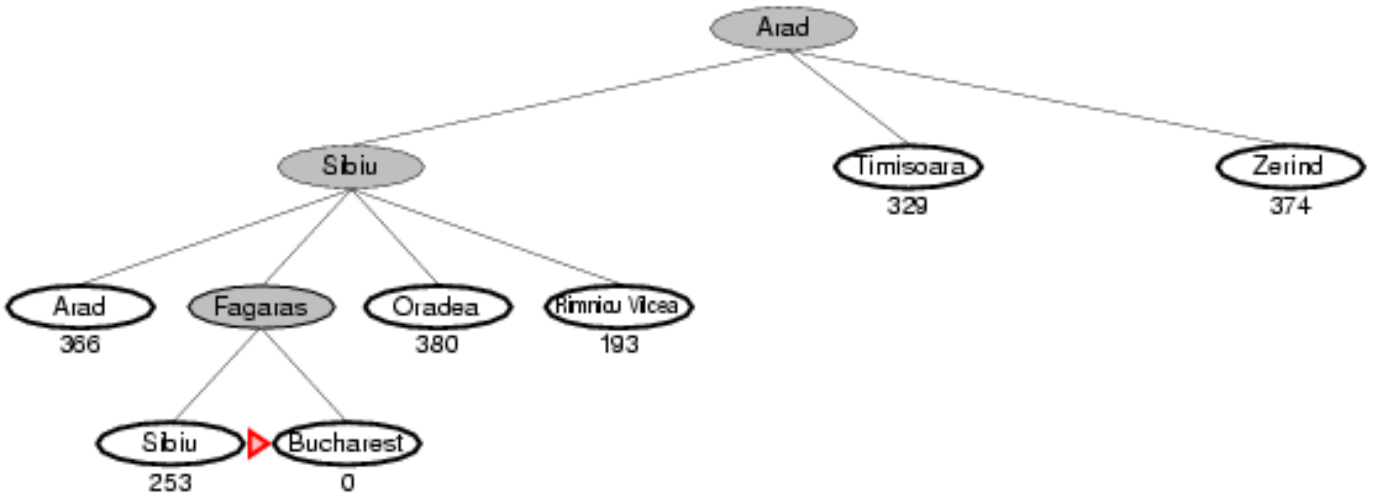


Greedy best-first search example

Greedy best-first search example

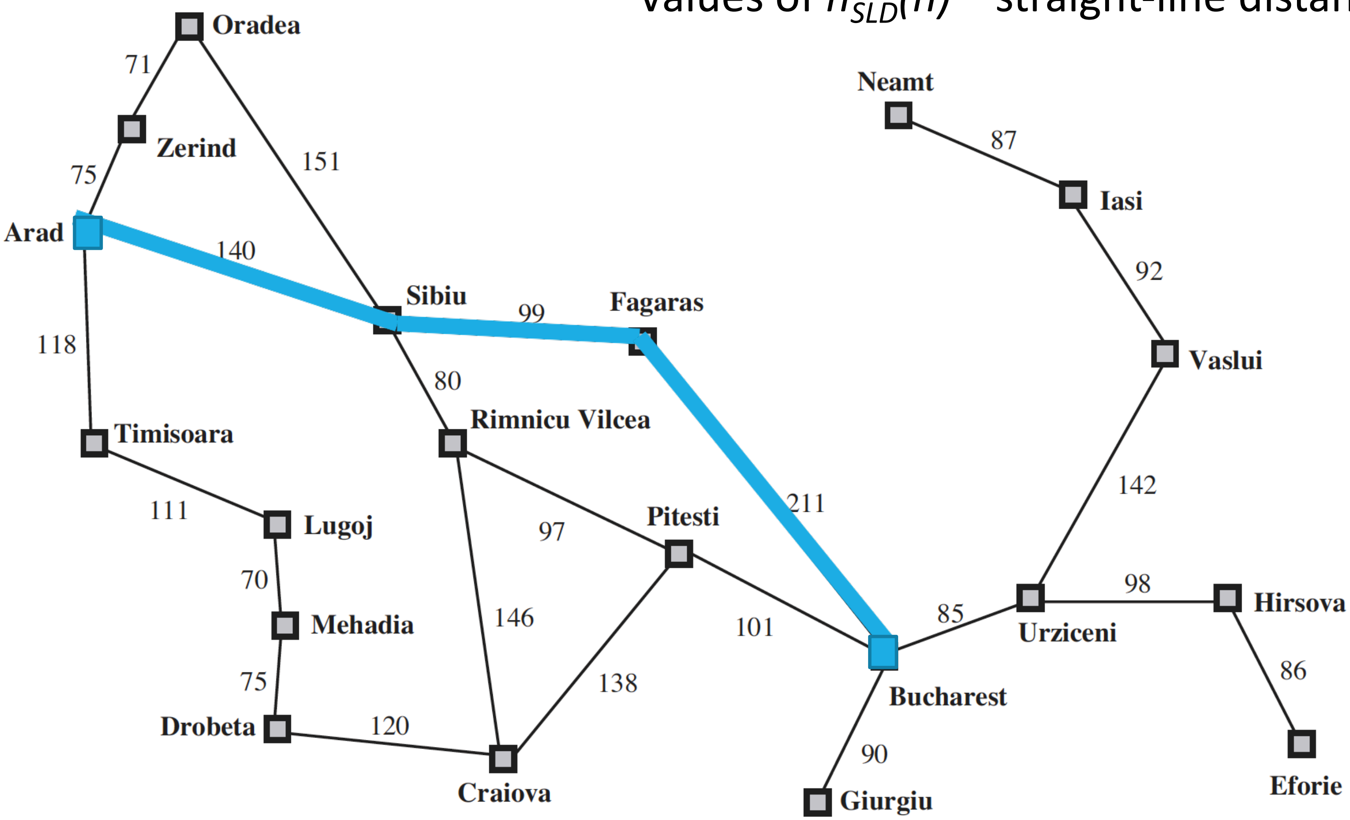


Greedy best-first search example



Romania with step costs in km

Values of $h_{SLD}(n)$—straight-line distances to Bucharest

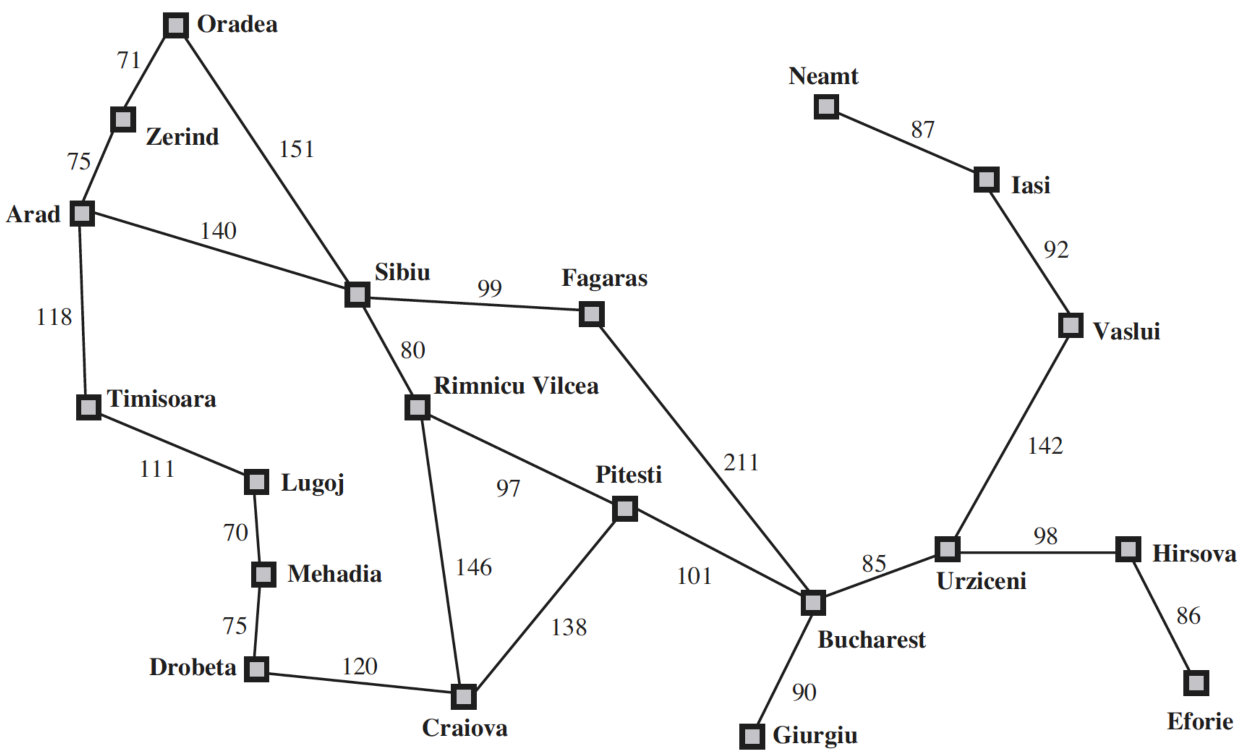


Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

NOT OPTIMAL! path through Rimnicu Vilcea and Pitesti. Is 32 km shorter!

Tree-Search Complete?

Consider the problem of getting from Iasi to Fagaras.



NOT COMPLETE!

Graph-Search Complete?

Finite spaces – yes

Infinite spaces - no

Complexities

Time? *$O(b^m)$*, but a good heuristic can give dramatic improvement

Space? *$O(b^m)$* -- keeps all nodes in memory

where is the maximum depth of the search space.

With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.

A* search

The most widely known form of best-first search is called A∗ search (pronounced "A-star search").

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

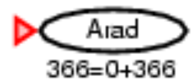
g(n) = cost so far to reach *n*

h(n) = estimated cost from *n* to goal

f(n) = estimated cost of the cheapest solution through n.

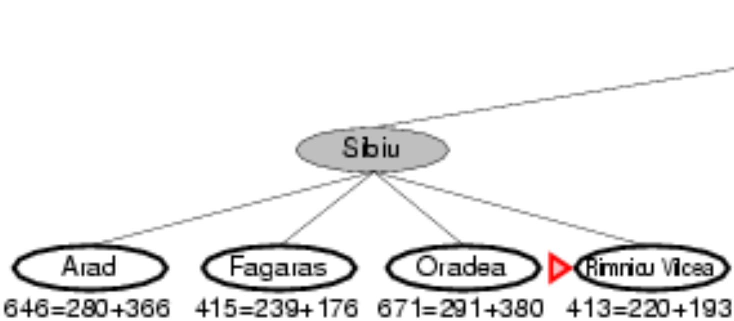
Provided that the heuristic function *h(n)* <u>satisfies certain conditions</u>, A∗ search is both <u>complete and optimal</u>.

A^* search example

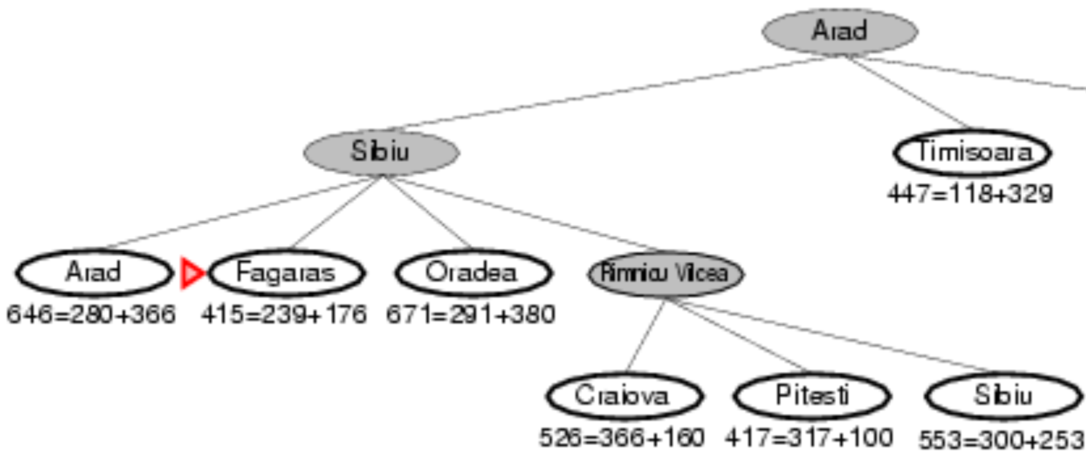


A* search example

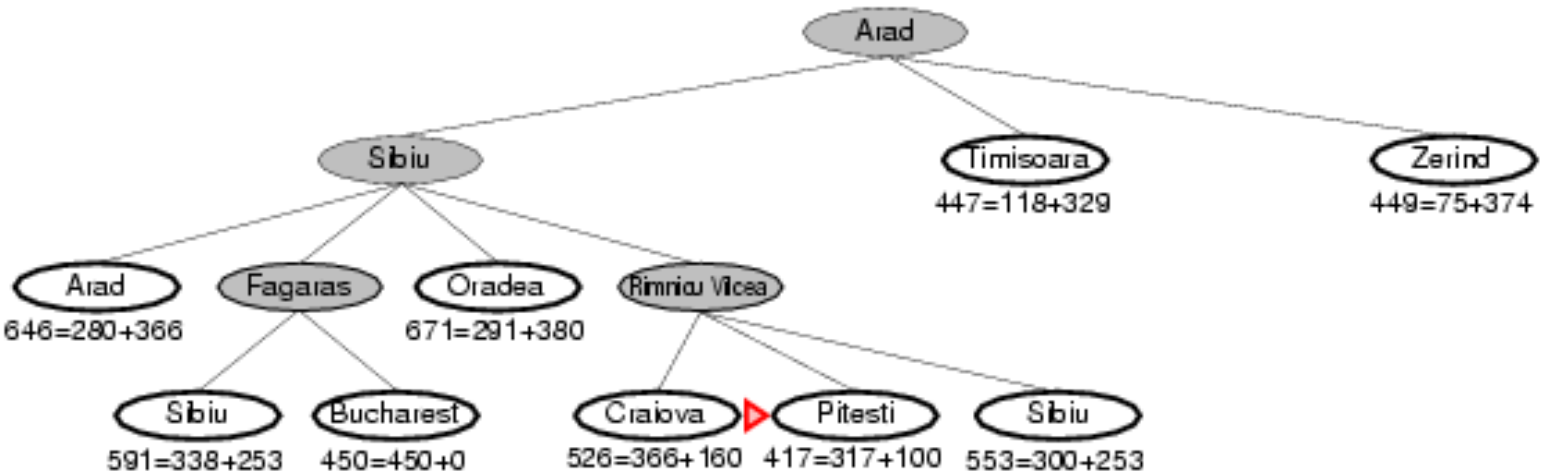
A^* search example



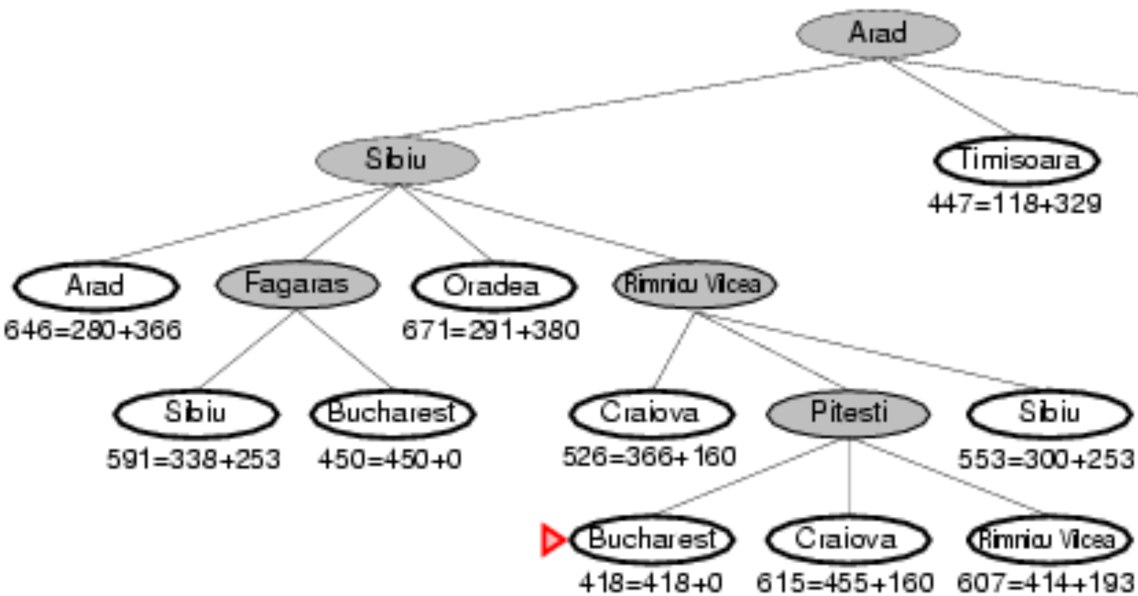
A* search example



A* search example



A* search example



Conditions for optimality
1) admissibility - ընդունելիություն

1) A heuristic $h(n)$ is admissible if for every node n,

$$h(n) \leq h^*(n),$$

where $h^*(n)$ is the true cost to reach the goal state from n.

An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

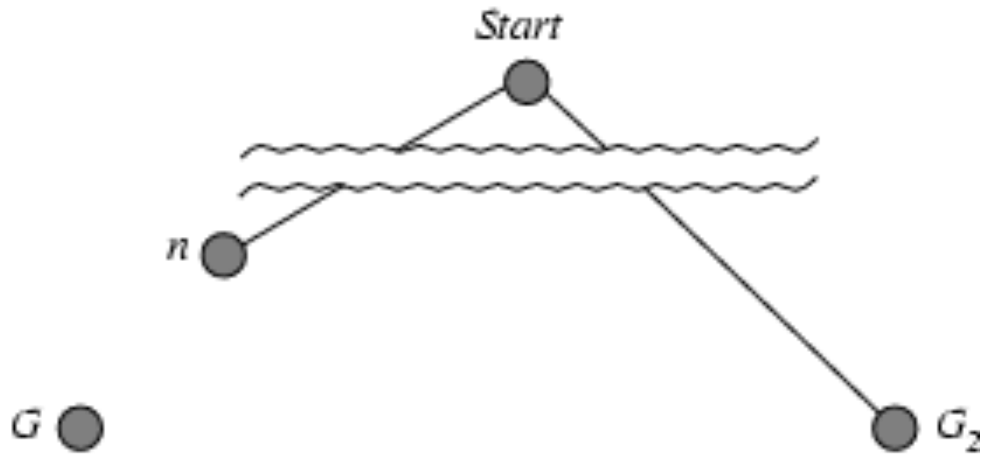
Example. $h_{SLD}(n)$ (never overestimates the actual road distance)

Conditions for optimality
1) admissibility - ընդունելիություն

Theorem: If *h(n)* is admissible, A* `TREE-SEARCH` is optimal.

1) admissibility – ընդունելիություն
Theorem Proof

Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$g(G_2) > g(G)$ since G_2 is suboptimal

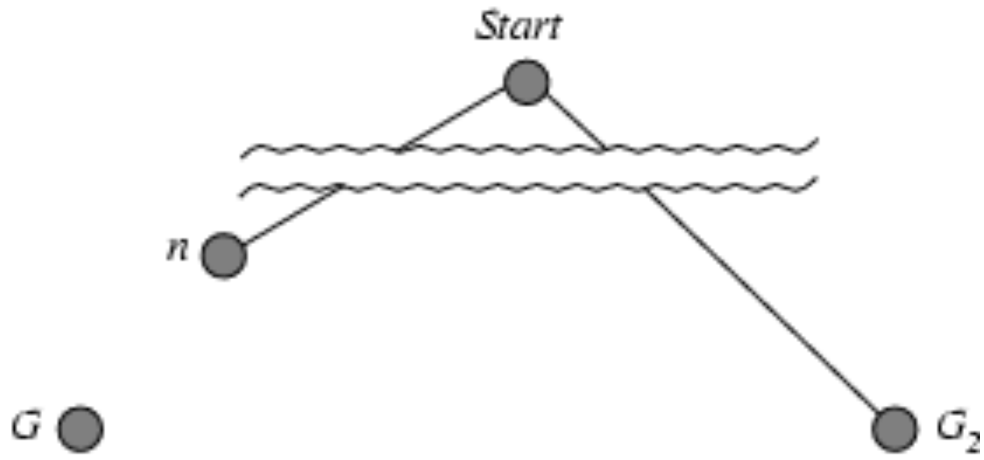
$f(G) = g(G)$ since $h(G) = 0$

$f(G_2) > f(G)$ from above

1) admissibility – ընդունելիություն

Theorem Proof

Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



$f(G_2) > f(G)$ from above

$h(n) \leq h^*(n)$ since h is admissible

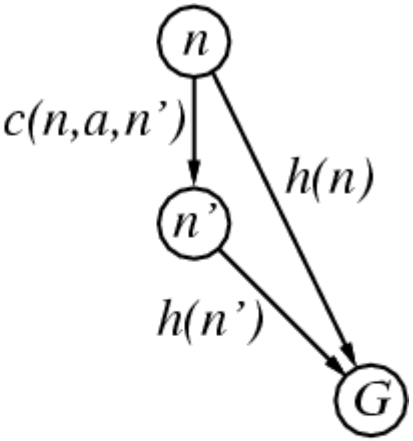
$g(n) + h(n) \leq g(n) + h^*(n)$

$f(n) \leq f(G)$ Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

Conditions for optimality
2) consistency - կայունություն

A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

$h(n) \le c(n,a,n') + h(n')$



If *h* is consistent, we have

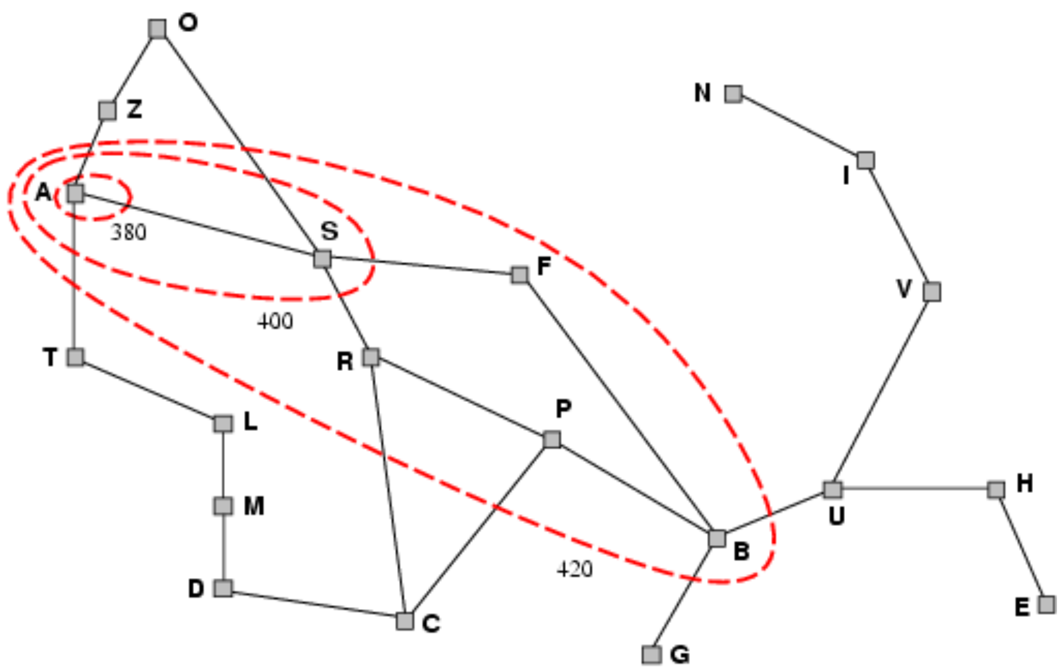
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\ge g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

i.e., *f(n)* is **non-decreasing** along any path.

Optimality of A*

A* expands nodes in order of increasing *f* value

- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Consistency vs admissibility

Prove:

1) Every consistent heuristic is also admissible.

2) Not all admissible heuristics are consistent.

Conditions for optimality
2) consistency - կայունություն

Theorem: If *h(n)* is consistent, A* `GRAPH-SEARCH` is optimal.

Properties of A*

<u>Complete?</u> Yes (unless there are infinitely many nodes with f $\leq f(G)$)

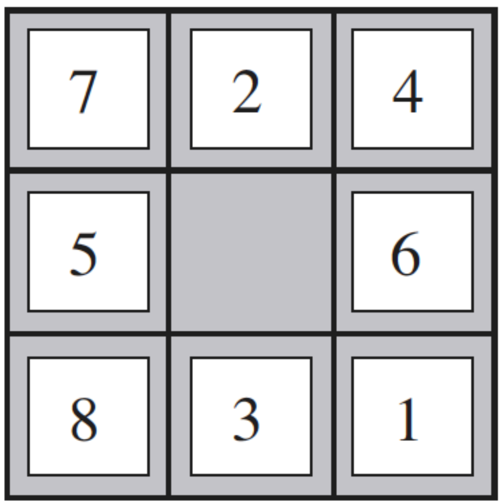
<u>Time?</u> Exponential

<u>Space?</u> Keeps all nodes in memory

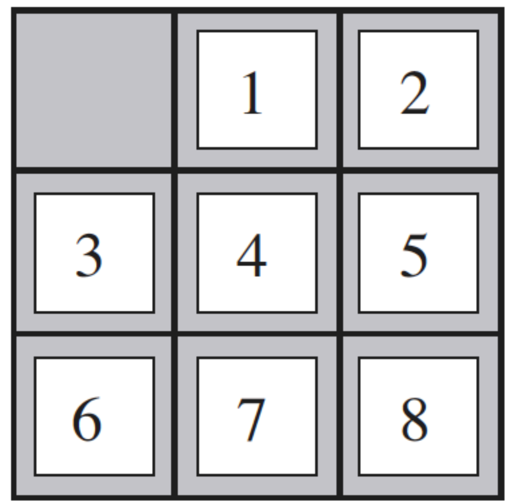
<u>Optimal?</u> Yes(if the corresponding conditions are satisifed)

HEURISTIC FUNCTIONS

We want to find the shortest solutions by using A∗. Therefore we need a heuristic function that never overestimates the number of steps to the goal



Start State

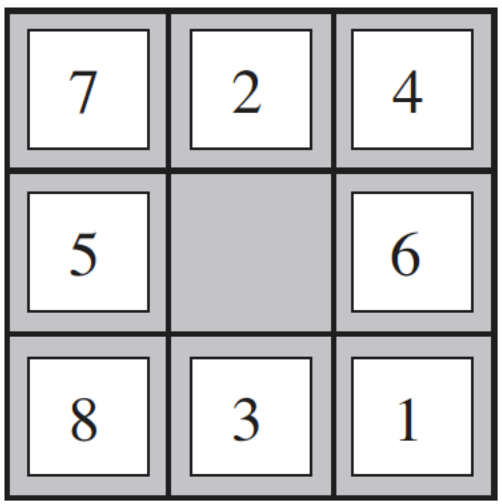


Goal State

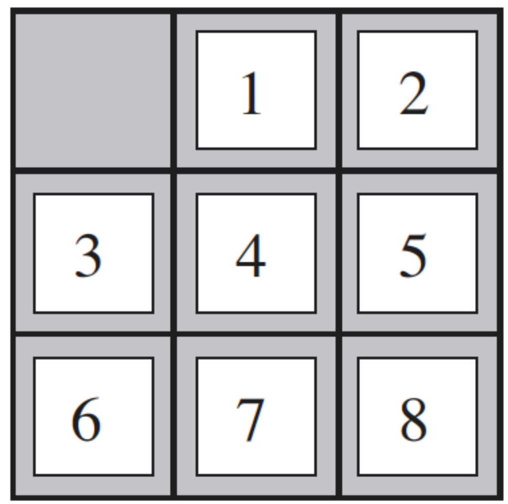
Admissible heuristics

1) $h_1(n)$ = number of misplaced tiles

all of the eight tiles are out of position => the start state would have. Thus $h_1 = 8 < 26$.



Start State



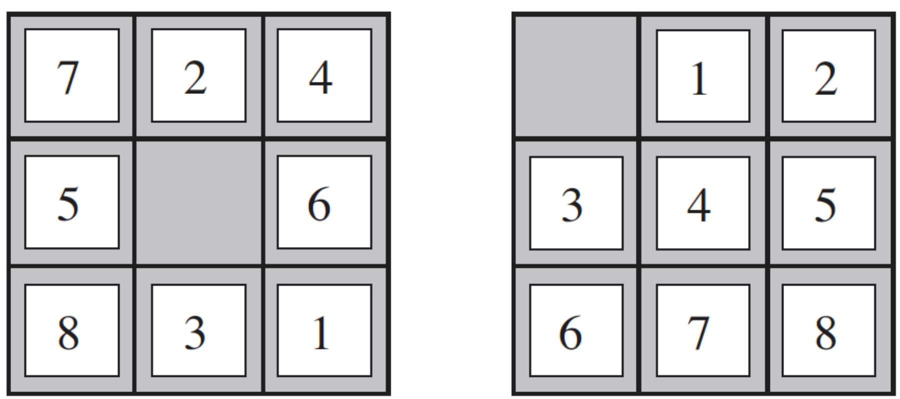
Goal State

Admissible heuristics

2) $h_2(n)$ = the sum of the distances of the tiles from their goal positions.

This is sometimes called the **city block distance** or **Manhattan distance**.

$h_2(n)$ = 4+1+2+2+2+3+3+2=18 < 26.



Start State Goal State

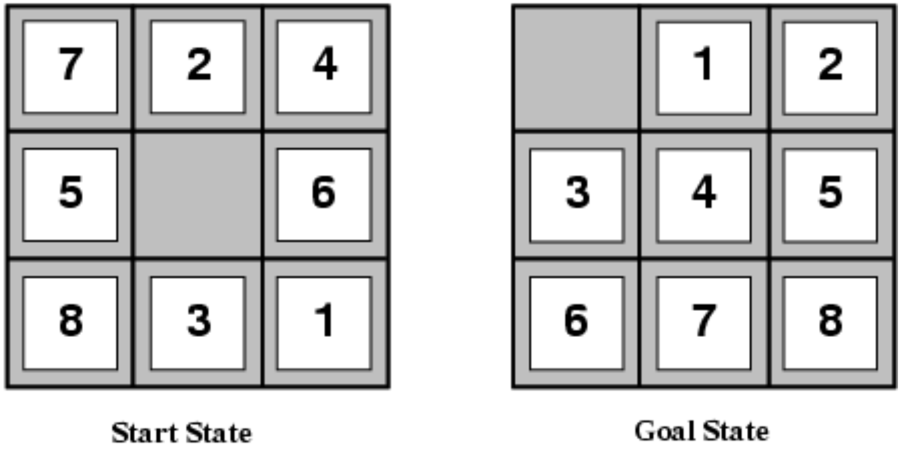
Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance

Which one is better?



Admissible heuristics

Admissible heuristics often come from *relaxing the constraints on the problem* (i.e., making it easier to solve)

A tile can move from square A to square B if

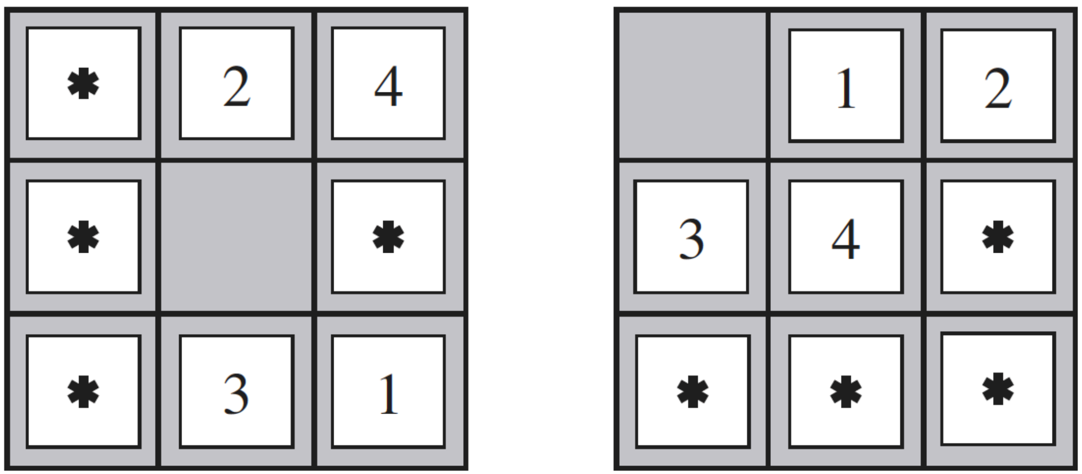
- A is horizontally or vertically adjacent to B and B is blank,

we can generate three relaxed problems by removing one or both of the conditions:

- (a) A tile can move from square A to square B if A is adjacent to B.
- (b) A tile can move from square A to square B if B is blank.
- (c) A tile can move from square A to square B.

Admissible heuristics

Admissible heuristics often come from *solving a subproblem of the original problem* (i.e., considering only part of the problem)



Start State

Goal State

Summary

Heuristic functions estimate costs of *shortest paths to goal*
- Good heuristic can dramatically reduce search cost

Greedy best-first search expands node with lowest h value
- incomplete and not always optimal

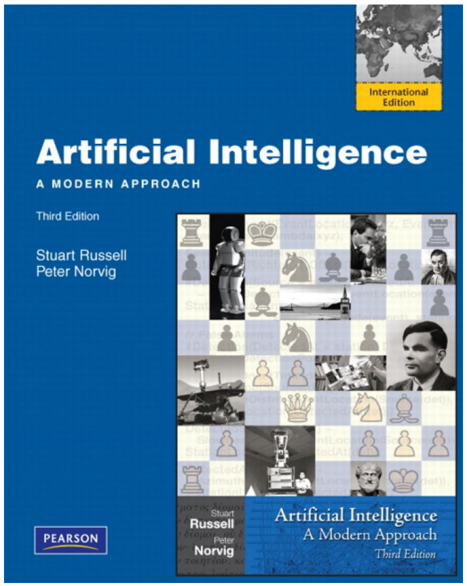
A* tree search expands nodes n with lowest g(n) + h(n)
- *complete* and *optimal* if heuristic is admissible
- also optimally efficient (up to tie-breaks)

A* *graph* search is optimal if heuristic is consistent

Admissible heuristics can be derived from relaxed problems

Sub-problems can be seen as relaxed problems

Reading List
Կարդալ



Chapter 3.5-3.6 (both versions)

https://github.com/samvelyan/Intelligent-Systems

Questions?
Հարցե՞ր