CS 1571 Introduction to AI
Lecture 6

Informed search methods

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Announcements

Homework assignment 2 is out

- Due on Thursday, September 18, 2014 before the class
- **Two parts:**
  - Pen and pencil part
  - Programming part (Puzzle 8): informed search methods

Course web page:

http://www.cs.pitt.edu/~milos/courses/cs1571/

Search methods

- **Uninformed search methods**
  - **Breadth-first search (BFS)**
  - **Depth-first search (DFS)**
  - **Iterative deepening (IDA)**
  - **Bi-directional search**
  - **Uniform cost search**
- **Informed (or heuristic) search methods:**
  - **Best first search with the heuristic function**

Evaluation-function driven search

- A search strategy can be defined in terms of **a node evaluation function**
  - Similarly to the path cost for the uniform cost search
- **Evaluation function**
  - Denoted $f(n)$
  - Defines the **desirability of a node to be expanded next**
- **Evaluation-function driven search:**
  - **expand the node (state) with the best evaluation-function value**
- **Implementation:**
  - **priority queue** with nodes in the decreasing order of their evaluation function value

Uniform cost search

- **Uniform cost search (Dijkstra's shortest path):**
  - A special case of the evaluation-function driven search

$$f(n) = g(n)$$

- **Path cost function** $g(n)$;
  - path cost from the initial state to n
- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - **weights or costs** associated with operators (links).
- **Note:** Uniform cost search relies on the problem definition only
  - It is an uninformed search method

Additional information to guide the search

- **Uninformed search methods**
  - use only the information from the problem definition; and
  - past explorations, e.g. cost of the path generated so far
- **Informed search methods**
  - incorporate additional measure of a potential of a specific state to reach the goal
  - a potential of a state (node) to reach a goal is measured by a **heuristic function**
- Heuristic function is denoted $h(n)$

Best-first search

Best-first search = evaluation-function driven search

- Typically incorporates a **heuristic function,** $h(n)$, into the evaluation function $f(n)$ to guide the search.

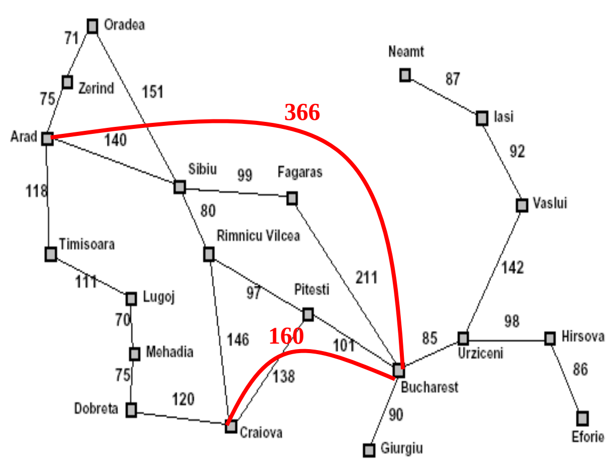
Heuristic function *h(n)*:

- Measures a potential of a state (node) to reach a goal
- Typically expressed in terms of some distance to a goal estimate

Example of a heuristic function:

- Assume a shortest path problem with city distances on connections
- Straight-line distances between cities give additional information we can use to guide the search

Example: traveler problem with straight-line distance information



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Straight-line distances** give an estimate of the cost of the path between the two cities

Best-first search

Best-first search = evaluation-function driven search

- Typically incorporates a **heuristic function,** $h(n)$, into the evaluation function $f(n)$ to guide the search.
- **heuristic function:** measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):

– **Greedy search**

$$f(n) = h(n)$$

– **A* algorithm**

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

\+ **iterative deepening** version of A* : **IDA***

Greedy search method

- Evaluation function is equal to the heuristic function

$$f(n) = h(n)$$

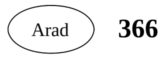
- **Idea:** the node that seems to be the closest to the goal is expanded first

Greedy search

f(n)=h(n)

queue ⇨

Arad	**366**

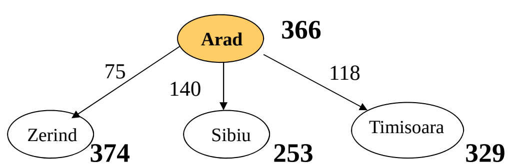


Greedy search

f(n)=h(n)

queue ⇨

Sibiu	**253**
Timisoara	**329**
Zerind	**374**

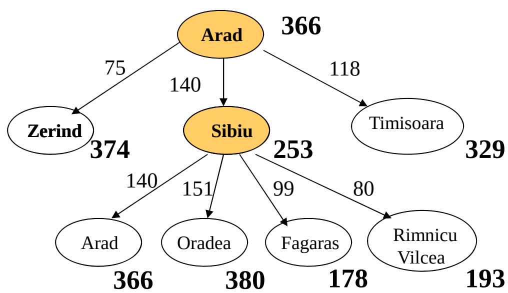


Greedy search

f(n)=h(n)

queue

Fagaras	**178**
Rimniciu V.	**193**
Timisoara	329
Arad	**366**
Zerind	374
Oradea	**380**

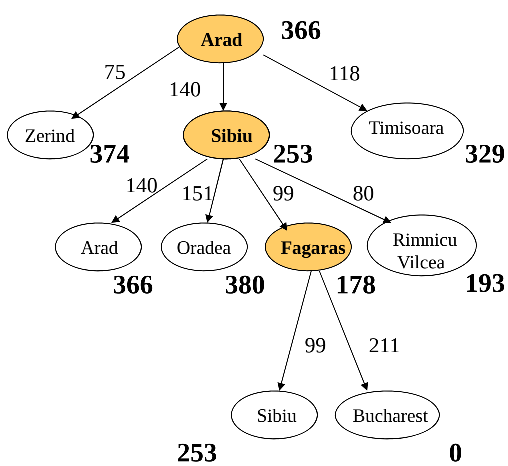


Greedy search

f(n)=h(n)

queue

Bucharest	**0**
Rimniciu V.	193
Sibiu	**253**
Timisoara	329
Arad	366
Zerind	374
Oradea	380



Goal !!!

Properties of greedy search

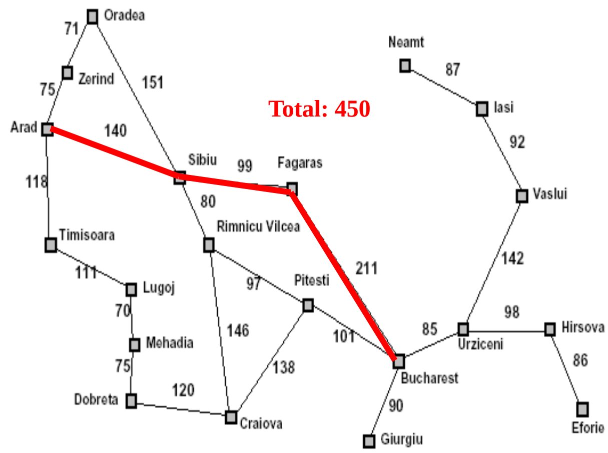
- **Completeness:** ?
- **Optimality: ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of greedy search

- **Completeness:**
  - **No**. We can loop forever. Nodes that seem to be the best choices can lead to cycles.

    Yes. Elimination of state repeats can solve the problem.
- **Optimality:** ?
- **Time complexity:**
- **Memory (space) complexity:**

Example: traveler problem with straight-line distance information

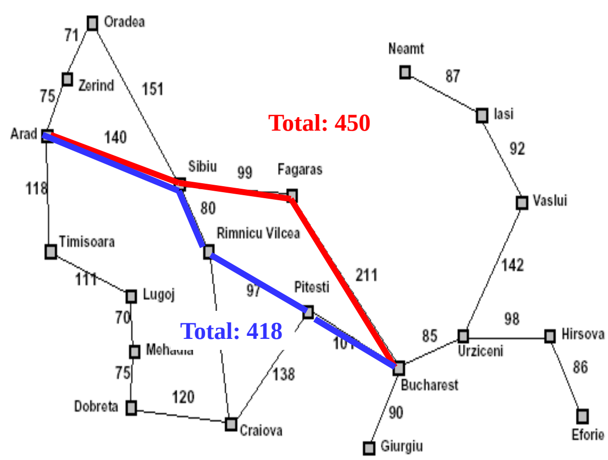


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Greedy search result**

Example: traveler problem with straight-line distance information



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Greedy search and optimality**

Properties of greedy search

- **Completeness:**
  - **No**. We can loop forever. Nodes that seem to be the best choices can lead to cycles.

    Yes. Elimination of state repeats can solve the problem.
- **Optimality: No.**

  Even if we reach the goal, we may be biased by a bad heuristic estimate. **Evaluation function disregards the cost of the path built so far.**
- **Time complexity:** $O(b^m)$

  Worst case !!! But often better!
- **Memory (space) complexity:** $O(b^m)$

  Often better!

A* search

- The problem with the **greedy search** is that it can keep expanding paths that are already very expensive.
- The problem with the **uniform-cost search** is that it uses only past exploration information (path cost), no additional information is utilized
- **A* search**

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

$g(n)$ - cost of reaching the state

$h(n)$ - estimate of the cost from the current state to a goal

$f(n)$ - estimate of the path length

- **Additional A*condition**: admissible heuristic

$$h(n) \le h^*(n) \quad \text{for all } n$$

A* search example

f(n)

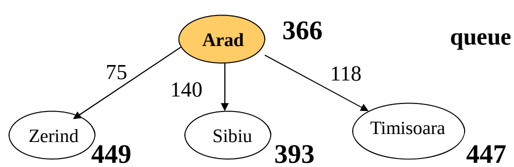
Arad 366

queue ⇨

Arad	366

A* search example

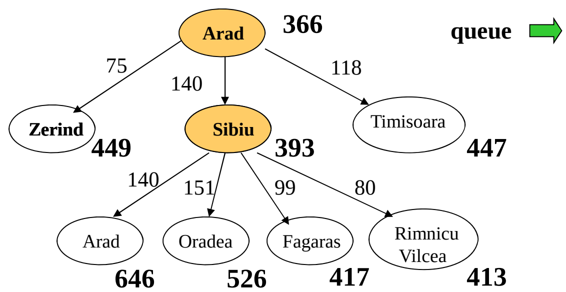
f(n)



queue ⇨

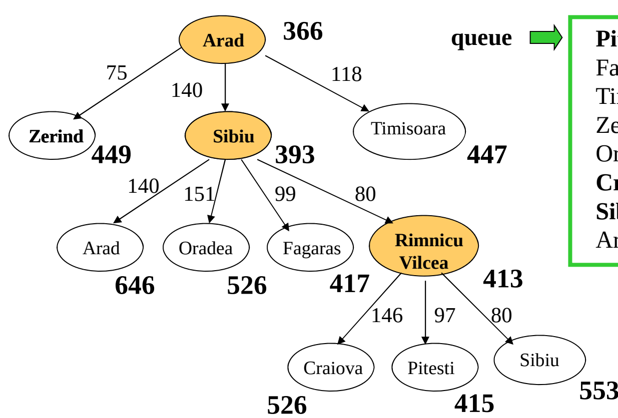
Sibiu	393
Timisoara	447
Zerind	449

A* search example



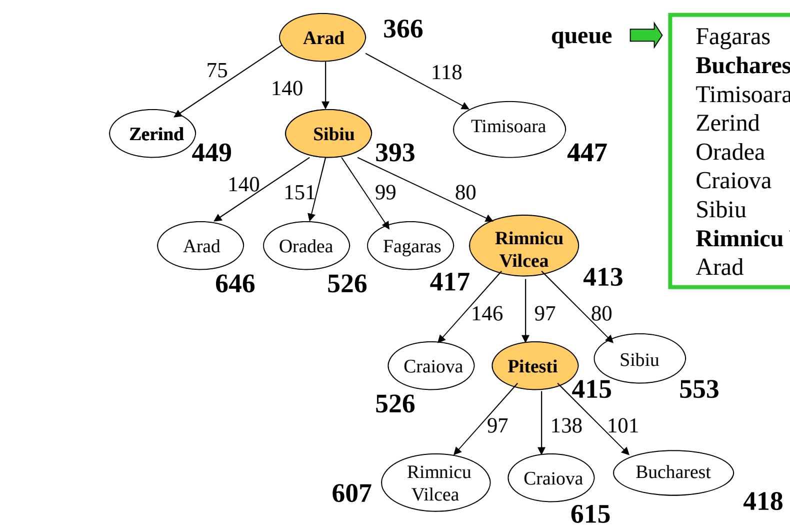
	f(n)
Rimnicu V.	**413**
Fagaras	**417**
Timisoara	447
Zerind	449
Oradea	**526**
Arad	**646**

A* search example



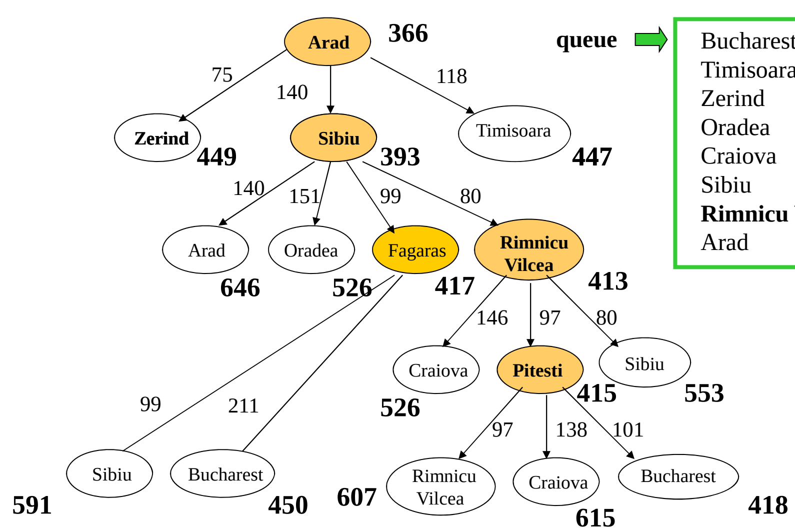
	f(n)
Pitesti	**415**
Fagaras	417
Timisoara	447
Zerind	449
Oradea	526
Craiova	**526**
Sibiu	**553**
Arad	646

A* search example



queue	*f(n)*
Fagaras	417
Bucharest	**418**
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

A* search example



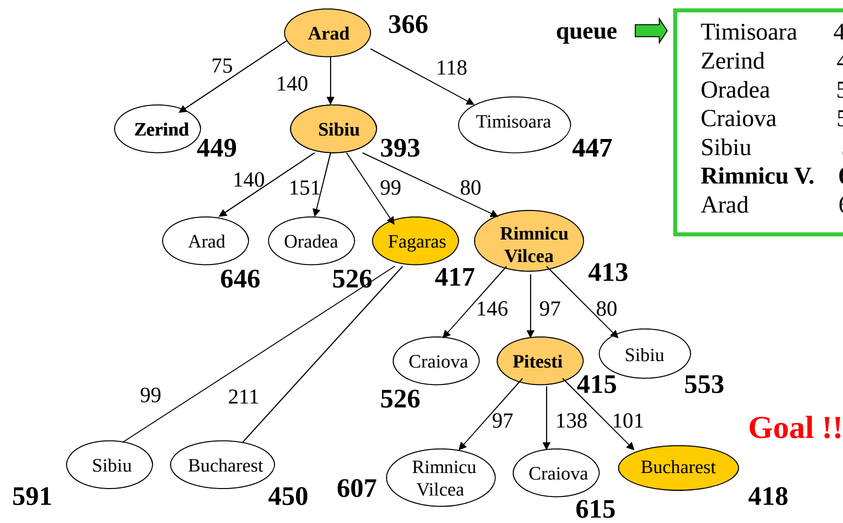
queue	*f(n)*
Bucharest	418
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

A* search example

f(n)

queue

Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646



Properties of A* search

- **Completeness: ?**
- **Optimality: ?**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Properties of A* search

- **Completeness:** **can we get stuck in the infinite loop?**
- **Optimality: ?**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Properties of A* search

- **Completeness:** **can we get stuck in the infinite loop? No!**
  - Then the algorithm is complete even with out repeat checks.
- **Optimality: ?**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Properties of A* search

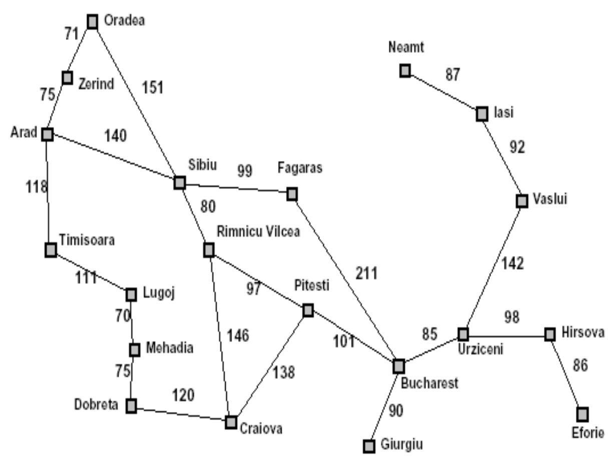
- **Completeness: Yes.**
- **Optimality: ?**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Optimality of A*

- In general, a heuristic function $h(n)$:

  It can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- Is the A* optimal for an arbitrary heuristic function?

Example: traveler problem with straight-line distance information



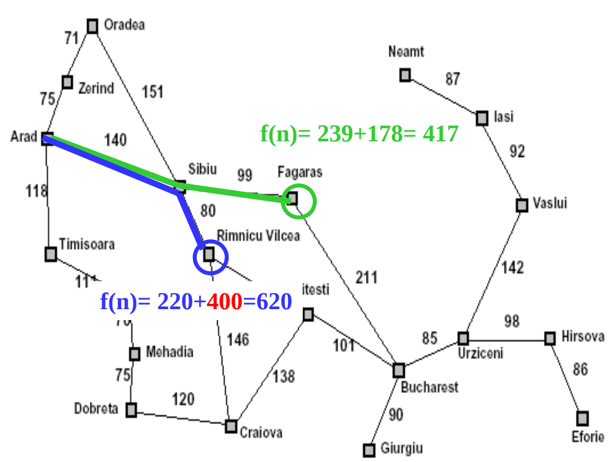
Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	400
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Admissible heuristics**

overestimate

Example: traveler problem with straight-line distance information

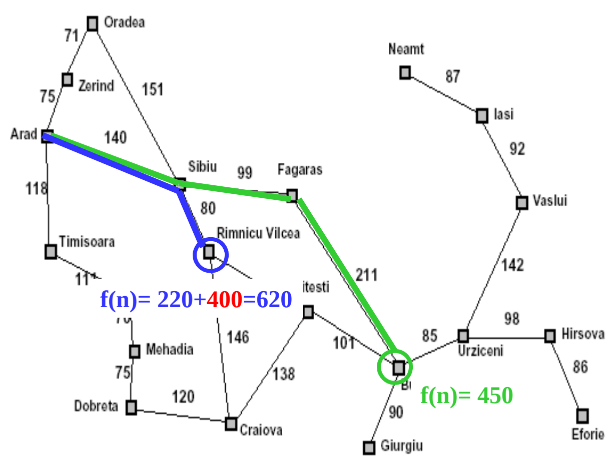


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	400
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Admissible heuristics**

Example: traveler problem with straight-line distance information



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	400
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Admissible heuristics**

Total path: 450 is suboptimal

Optimality of A*

- In general, a heuristic function $h(n)$:

  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- Is the A* optimal for an arbitrary heuristic function?
- **No!**

Optimality of A*

- In general, a heuristic function $h(n)$:

  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- **Admissible heuristic condition**
  - **Never overestimate the distance to the goal !!!**

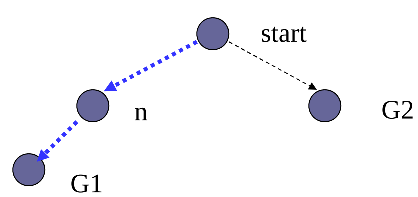
$$h(n) \leq h^*(n) \quad \text{for all } n$$

Example: the straight-line distance in the travel problem never overestimates the actual distance

Is A* search with an admissible heuristic optimal ??

Optimality of A* (proof)

- Let G1 be the optimal goal (with the minimum path distance). Assume that we have a sub-optimal goal G2. Let *n* be a node that is on the optimal path and is in the queue together with G2



Then:

$$\begin{aligned} f(G2) &= g(G2) && \text{since } h(G2) = 0 \\ &> g(G1) && \text{since G2 is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

And thus **A* never selects G2 before *n***

Properties of A* search

- **Completeness:** **Yes.**
- **Optimality:** **Yes (with the admissible heuristic)**
- **Time complexity:**
  - **?**
- **Memory (space) complexity:**
  - **?**

Properties of A* search

- **Completeness:** **Yes.**
- **Optimality:** **Yes (with the admissible heuristic)**
- **Time complexity:**
  - **Order roughly the number of nodes with *f(n)* smaller than the cost of the optimal path *g****
- **Memory (space) complexity:**
  - **Same as time complexity (all nodes in the memory)**

Admissible heuristics

- Heuristics can be designed based on relaxed version of problems
- **Example:** the 8-puzzle problem

Initial position

4	5	
6	1	8
7	3	2

Goal position

1	2	3
4	5	6
7	8	

- **Admissible heuristics:**
  1. number of misplaced tiles
  2. Sum of distances of all tiles from their goal positions (Manhattan distance)

Admissible heuristics

Heuristics 1: number of misplaced tiles

Initial position

4	5	
6	1	8
7	3	2

Goal position

1	2	3
4	5	6
7	8	

h(n) for the initial position: 7

Admissible heuristics

- **Heuristic 2:** Sum of distances of all tiles from their goal positions (Manhattan distance)

Initial position

4	5	
6	1	8
7	3	2

Goal position

1	2	3
4	5	6
7	8	

h(n) for the initial position:
2 + 3 + 3 + 1 + 1 + 2 + 0 + 2 = 14

For tiles: 1 2 3 4 5 6 7 8

Admissible heuristics

- We can have multiple admissible heuristics for the same problem
- **Dominance:** Heuristic function h_1 dominates h_2 if

$$\forall n \quad h_1(n) \geq h_2(n)$$

- **Combination:** two or more admissible heuristics can be combined to give a new admissible heuristics
  - Assume two admissible heuristics h_1, h_2

Then: $h_3(n) = \max(h_1(n), h_2(n))$

is admissible