

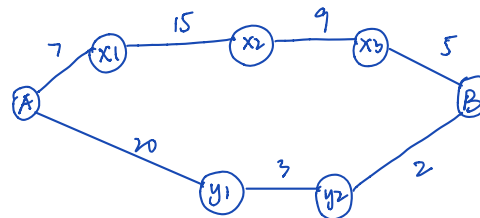
# INFORMED SEARCH ALGORITHMS

## CHAPTER 3, SECTIONS 5–6

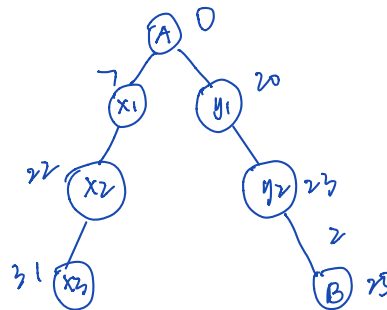
# Outline

- ◇ Best-first search
- ◇ A\* search
- ◇ Heuristics
- ◇ Hill-climbing

Example of UCS



UCS tree



## Review: General search

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution, or failure
  nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
  loop do
    if nodes is empty then return failure
    node ← REMOVE-FRONT(nodes)
    if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
    nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
  end
```

A strategy is defined by picking the *order of node expansion*

BFS	FIFO
DFS	LIFO

# Best-first search

Idea: use an *evaluation function* for each node

– estimate of “desirability”

⇒ Expand most desirable unexpanded node

Implementation:

QUEUEINGFN = insert successors in decreasing order of desirability

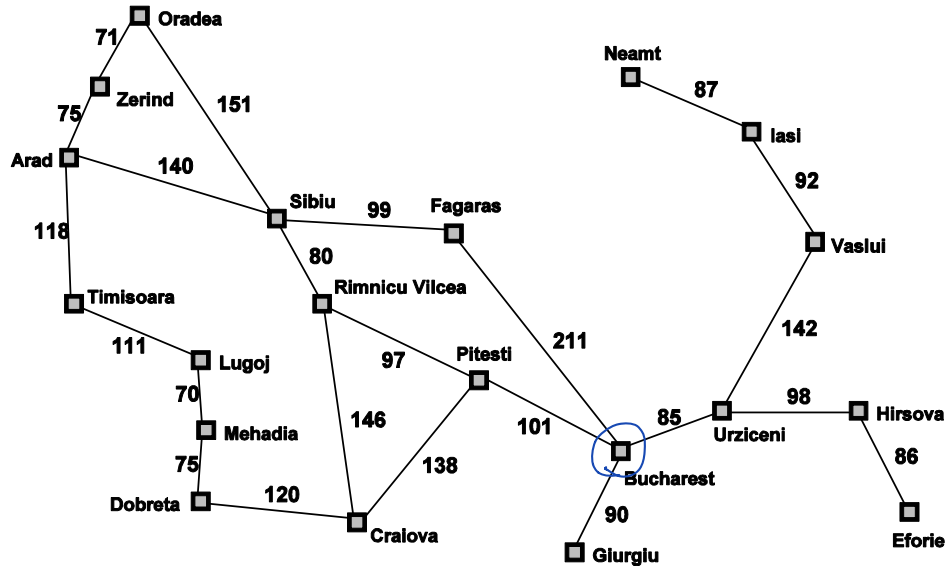
Special cases:

greedy search

A\* search

# Romania with step costs in km

*know the straight-line distance*



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

*use in NP hard.  
tree deepy  
time limitation*

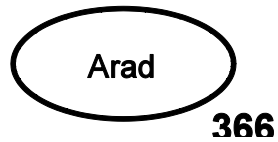
# Greedy search

Evaluation function  $h(n)$  (heuristic)  
= estimate of cost from  $n$  to goal

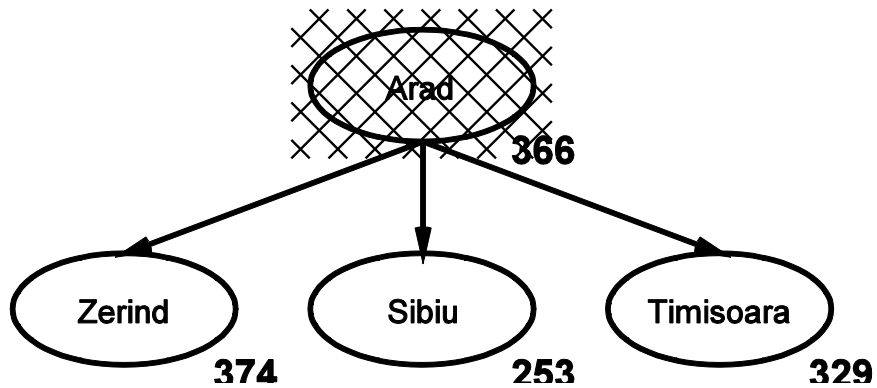
E.g.,  $h_{\text{SLD}}(n)$  = straight-line distance from  $n$  to Bucharest

Greedy search expands the node that appears to be closest to goal

## Greedy search example

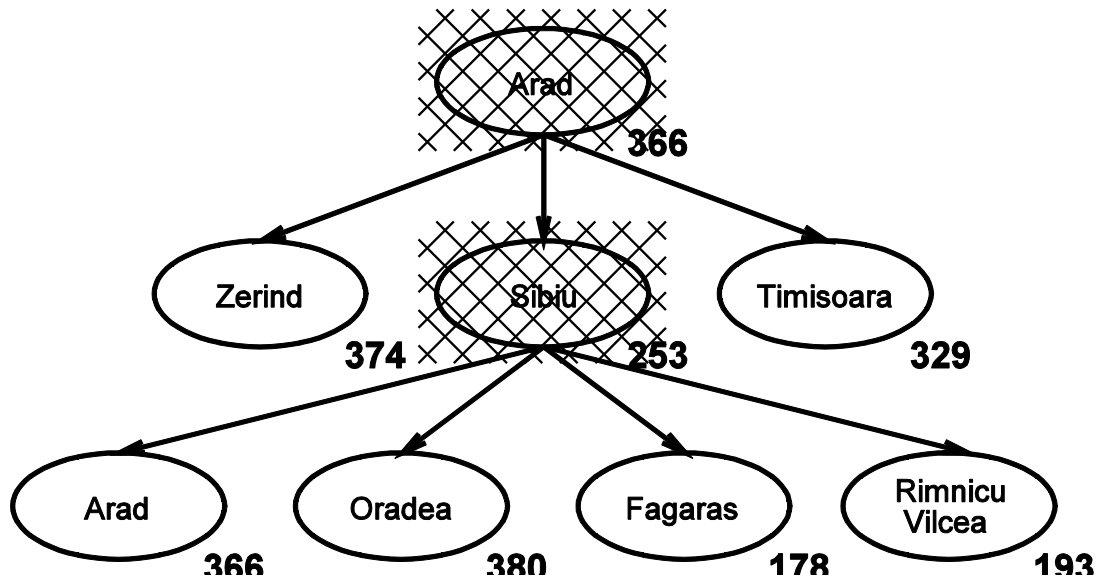


## Greedy search example

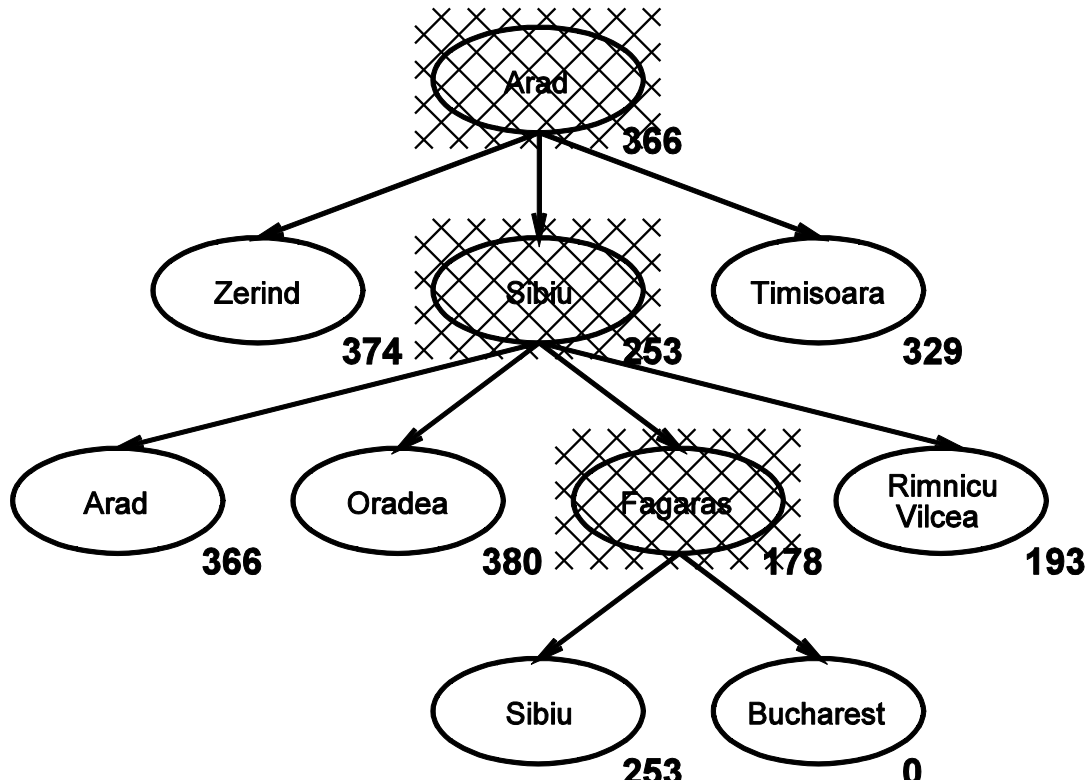




## Greedy search example



## Greedy search example



# Properties of greedy search

Complete??

Time??

Space??

Optimal??

# Properties of greedy search

Complete?? No – can get stuck in loops, e.g., lasi to Fagaras  
lasi  $\rightarrow$  Neamt  $\rightarrow$  lasi  $\rightarrow$  Neamt  $\rightarrow$

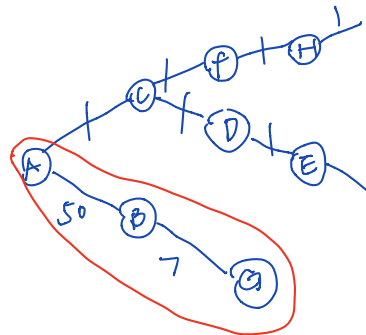
Complete in finite space with repeated-state checking

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

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

Optimal?? No  $\rightarrow$  return what looks good rather than least cost solution

problem of VCS



## A\* search

Idea: avoid expanding paths that are already expensive

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

$g(n)$  = cost so far to reach  $n$  (path cost)

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

$f(n)$  = estimated total cost of path through  $n$  to goal

A\* search uses an *admissible* heuristic

i.e.,  $h(n) \leq h^*(n)$  where  $h^*(n)$  is the *true* cost from  $n$ .

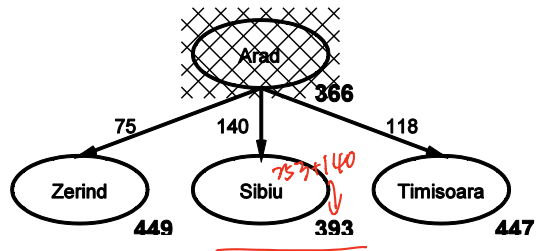
E.g.,  $h_{\text{SLD}}(n)$  never overestimates the actual road distance

Theorem: A\* search is optimal

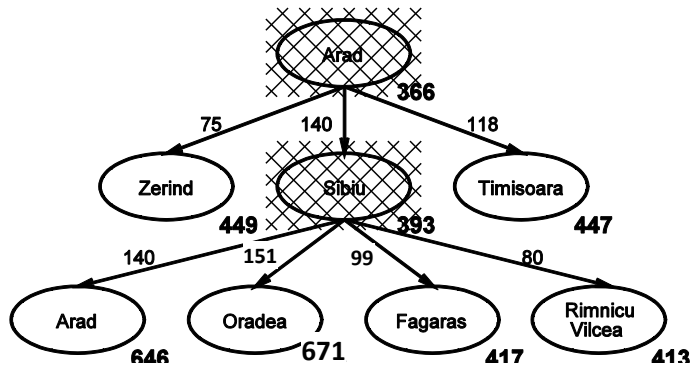
# A\* search example

Arad  
**366**

# A\* search example

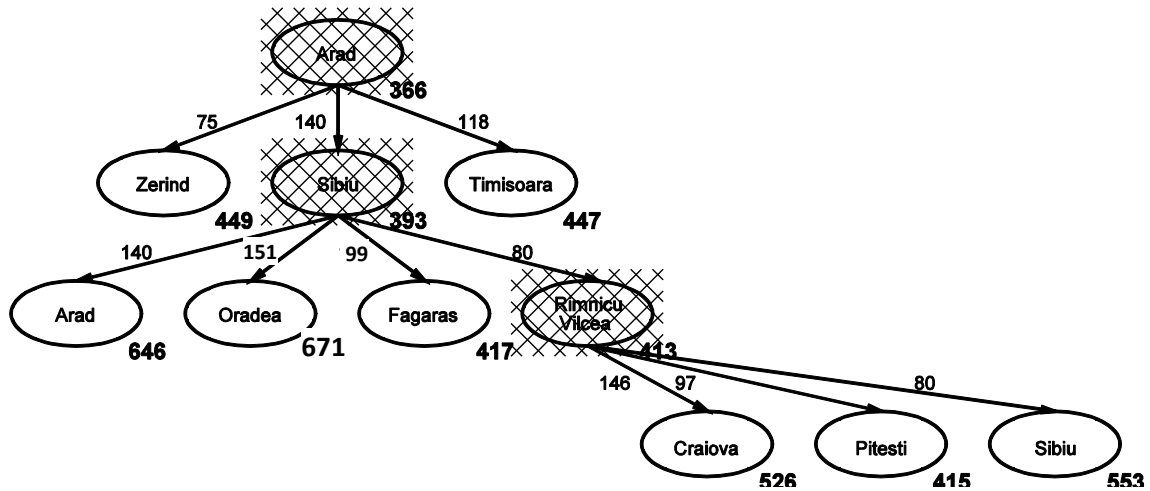


# A\* search example

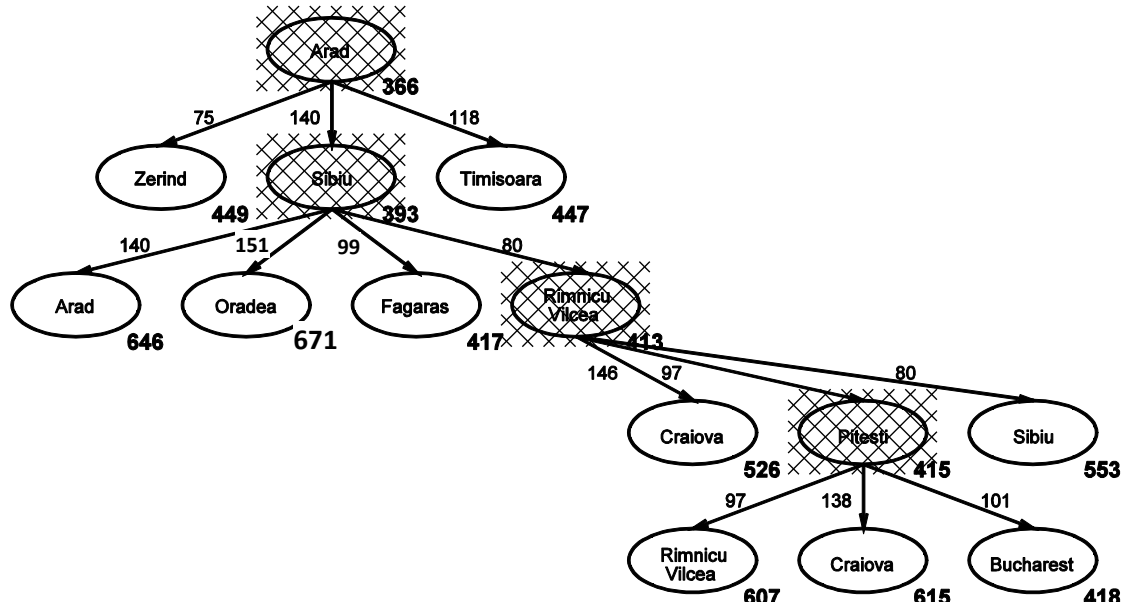




# A\* search example



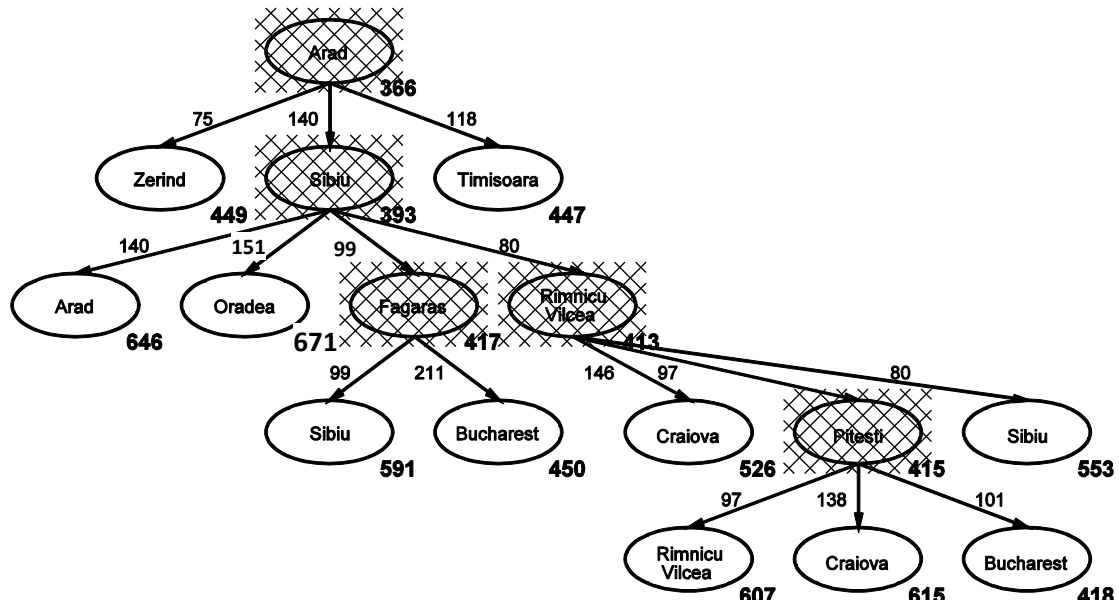
# A\* search example



find the optimal solution

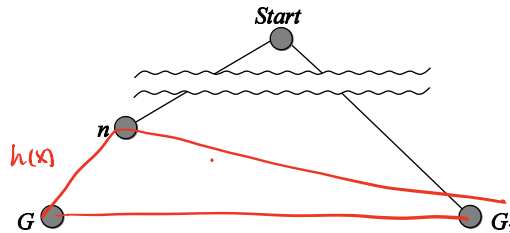
→ if find node < 418, expand it.  
eg. Fagaras 417 → expand

# A\* search example



# Optimality of A\* (standard proof)

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



$$\begin{aligned}
 f(G_2) &= g(G_2) \quad \text{since } h(G_2) = 0 \quad \checkmark \\
 &> \underline{g(G)} \quad \text{since } G_2 \text{ is } \underline{\text{suboptimal}} \quad \checkmark \\
 &\geq f(n) \quad \text{since } h \text{ is admissible}
 \end{aligned}$$

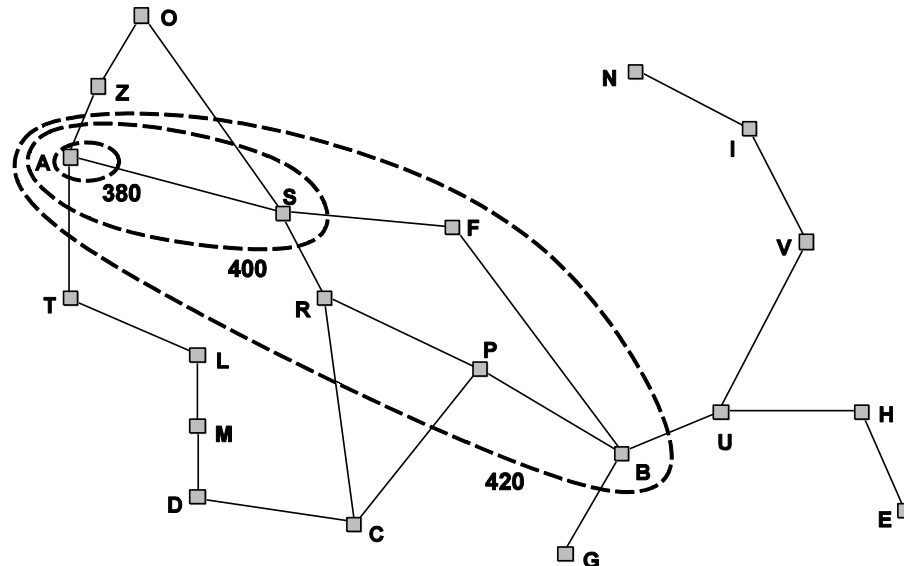
Since  $f(G_2) > f(n)$ , A\* will never select  $G_2$  for expansion

## Optimality of $A^*$ (more useful)

Lemma:  $A^*$  expands nodes in order of increasing  $f$  value

Gradually adds “ $f$ -contours” of nodes (cf. breadth-first adds layers)

Contour  $i$  has all nodes with  $f = f_i$ , where  $f_i < f_{i+1}$



## Properties of $A^*$

Complete?? Yes, unless there are infinitely many nodes with  $f \leq f(G)$

*have to expand all nodes  $\leq f(G)$ .*

? Time?? Exponential in  $[relative\ error\ in\ h \times length\ of\ soln.]$

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand  $f_{i+1}$  until  $f_i$  is finished

## The heuristic can control $A^*$ 's behaviour

- ◇ If  $h(n)$  is very high relative to  $g(n)$ , then only  $h(n)$  plays a role, and  $A^*$  turns into ... *greedy search*
- ◇ If  $h(n)$  is 0, then only  $g(n)$  plays a role, and  $A^*$  turns into ... *uniform cost search* ~~BFS~~
- ◇ If  $h(n)$  is always lower than (or equal to) the cost of moving from  $n$  to the goal, then  $A^*$  ...
- ◇ If  $h(n)$  is sometimes greater than the cost of moving from  $n$  to the goal, then  $A^*$  ...
- ◇ If  $h(n)$  is exactly equal to the cost of moving from  $n$  to the goal, then  $A^*$  ...

## The heuristic can control A\*'s behaviour

- ◇ If  $h(n)$  is very high relative to  $g(n)$ , then only  $h(n)$  plays a role, and A\* turns into Greedy Best-First-Search.
- ◇ If  $h(n)$  is 0, then only  $g(n)$  plays a role, and A\* turns into Uniform Cost Search, which finds the optimal solution.
- ◇ If  $h(n)$  is always lower than (or equal to) the cost of moving from  $n$  to the goal, then A\* is guaranteed to find a shortest path. The lower  $h(n)$  is, the more node A\* expands, making it slower.
- ◇ If  $h(n)$  is sometimes greater than the cost of moving from  $n$  to the goal, then A\* is not guaranteed to find a shortest path, but it can run faster.
- ◇ If  $h(n)$  is exactly equal to the cost of moving from  $n$  to the goal, then A\* will only follow the best path and never expand anything else, making it very fast.

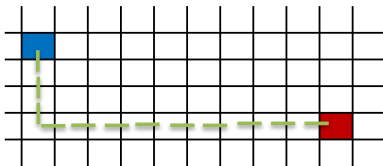


## Examples of well-known heuristic functions

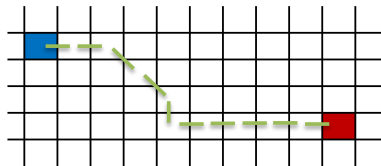
- ◇ Manhattan distance ( $L_1$ ): On a square grid that allows 4 directions of movement.
- ◇ Diagonal distance ( $L_\infty$ ): On a square grid that allows 8 directions of movement.  
*allow diagonal movement*
- ◇ Euclidean distance ( $L_2$ ): On a square grid that allows any direction of movement.



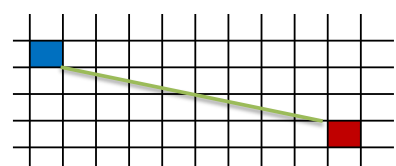
**Manhattan Distance**



**Diagonal Distance**



**Euclidean Distance**



# Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$  = number of misplaced tiles

$h_2(n)$  = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

$$h_1(S) = ?? \quad 7$$

$$\underline{\underline{h_2(S) = ??}} \quad 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18.$$

# Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$  = number of misplaced tiles

$h_2(n)$  = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

$$h_1(S) = ?? \quad 7$$

$$\underline{\underline{h_2(S) = ?? \quad 2+3+3+2+4+2+0+2 = 18}}$$

# Dominance

If  $h_2(n) \geq h_1(n)$  for all  $n$  (both admissible)  
then  $h_2$  dominates  $h_1$  and is better for search

Typical search costs:

$d = 14$  IDS = 3,473,941 nodes

$A^*(h_1) = 539$  nodes

$A^*(h_2) = 113$  nodes

$d = 24$  IDS = too many nodes

$A^*(h_1) = 39,135$  nodes

$A^*(h_2) = 1,641$  nodes

## Relaxed problems

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then  $h_1(n)$  gives the shortest solution

If the rules are relaxed so that a tile can move to *any adjacent square*, then  $h_2(n)$  gives the shortest solution *no other tiles.*

# Iterative improvement algorithms

In many optimization problems, *path* is irrelevant;  
the goal state itself is the solution

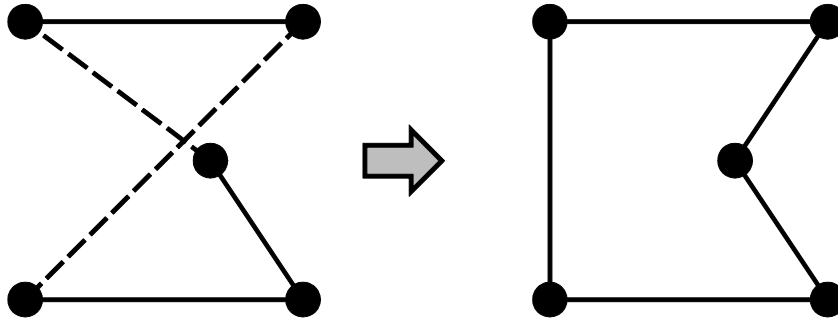
Then state space = set of “complete” configurations;  
find *optimal* configuration, e.g., Travelling Salesperson Problem  
or, find configuration satisfying constraints, e.g., n-queens

In such cases, can use *iterative improvement* algorithms;  
keep a single “current” state, try to improve it

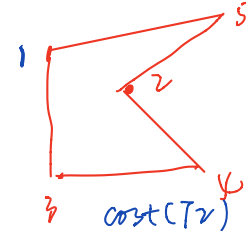
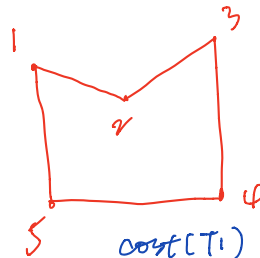
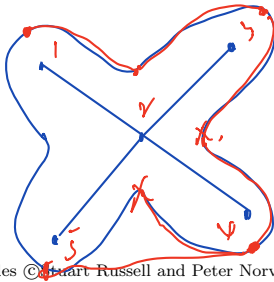
Constant space, suitable for online as well as offline search

# Example: Travelling Salesperson Problem

Find the shortest tour that visits each city exactly once



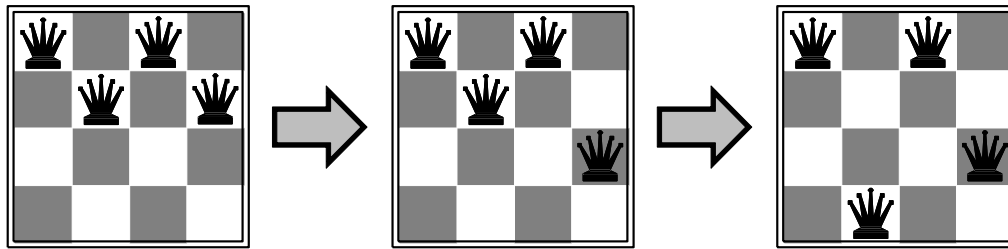
Relaxed problem: let path be *any* structure that connects all cities  
 $\Rightarrow$  use minimum spanning tree as heuristic for the TSP



calculate each loop and choose minimum

## Example: $n$ -queens

Put  $n$  queens on an  $n \times n$  board with no two queens on the same row, column, or diagonal





# Hill-climbing (or gradient ascent/descent)

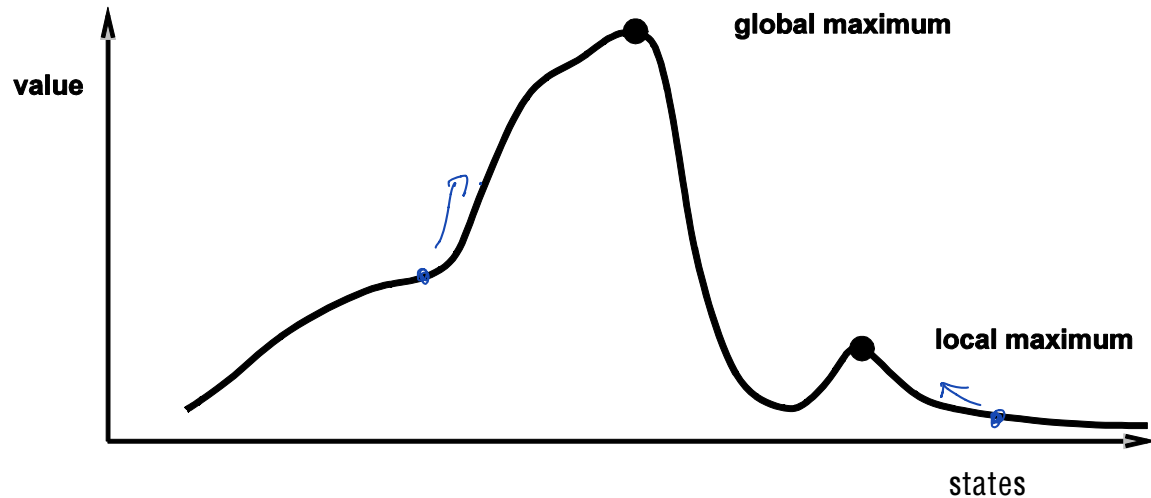
“Like climbing Everest in thick fog with amnesia”

```
function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  local variables: current, a node
                     next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    next ← a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current ← next
  end
```

## Hill-climbing contd.

Problem: depending on initial state, can get stuck on local maxima



## Summary

Heuristics help reduce search cost,  
however, finding an optimal solution is still difficult.

Greedy best-first search is not optimal, but can be efficient.

$A^*$  search is complete and optimal, but is prohibitive in memory.

Hill-climbing methods operate on complete-state formulations,  
require less memory, but are not optimal.

Examples of skills expected:

- ◇ Demonstrate operation of search algorithms
- ◇ Discuss and evaluate the properties of search algorithms
- ◇ Derive and compare heuristics for a problem