Overview

- Informed search strategies
- Heuristic functions
- Local search and optimization
- Local search in continuous spaces
- Searching with nondeterministic actions
- Searching with partial observations
- Online search agents and unknown environments

Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Best-first search

Idea: use an evaluation function for each node
– estimate of "desirability"

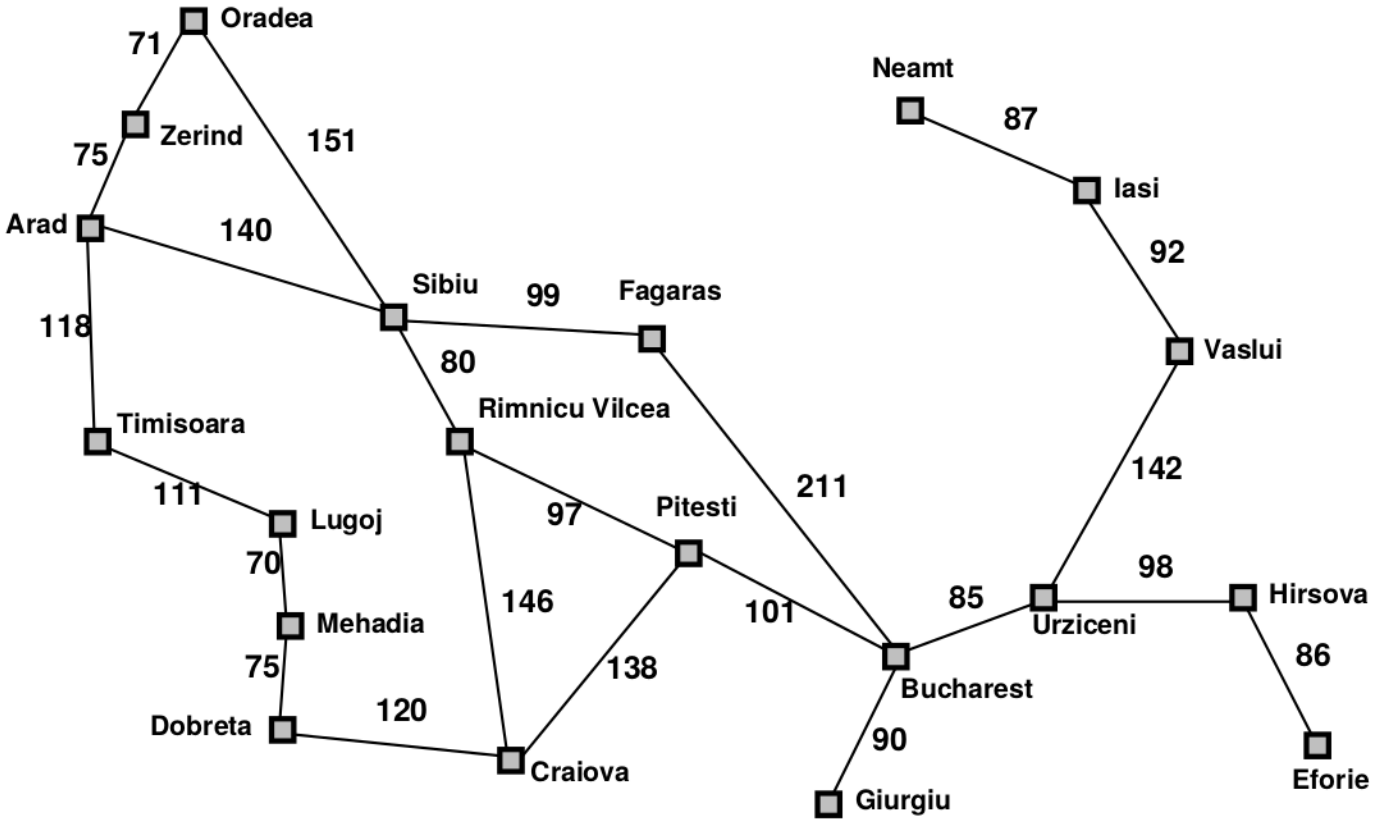
⇒ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A* search

f(n) : evaluation function
lowest evaluation is selected for expansion
maintain fringe in the ascending order of f-values.

Romania with step costs in km



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

Evaluation function $h(n)$ (**h**euristic)

$=$ estimate of cost from n to the closest goal

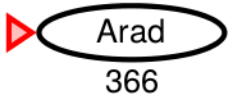
^ of the cheapest path

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

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

if f(n) = h(n) --> greedy search

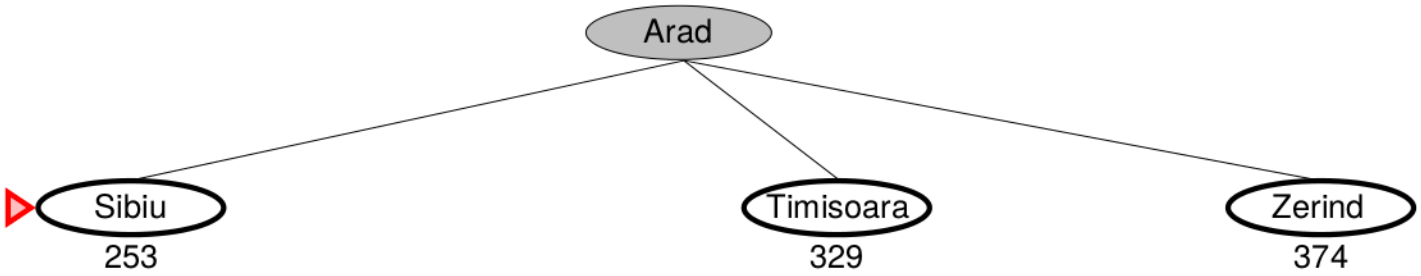
Greedy search example



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

Values of h_{SLD}—straight-line distances to Bucharest.

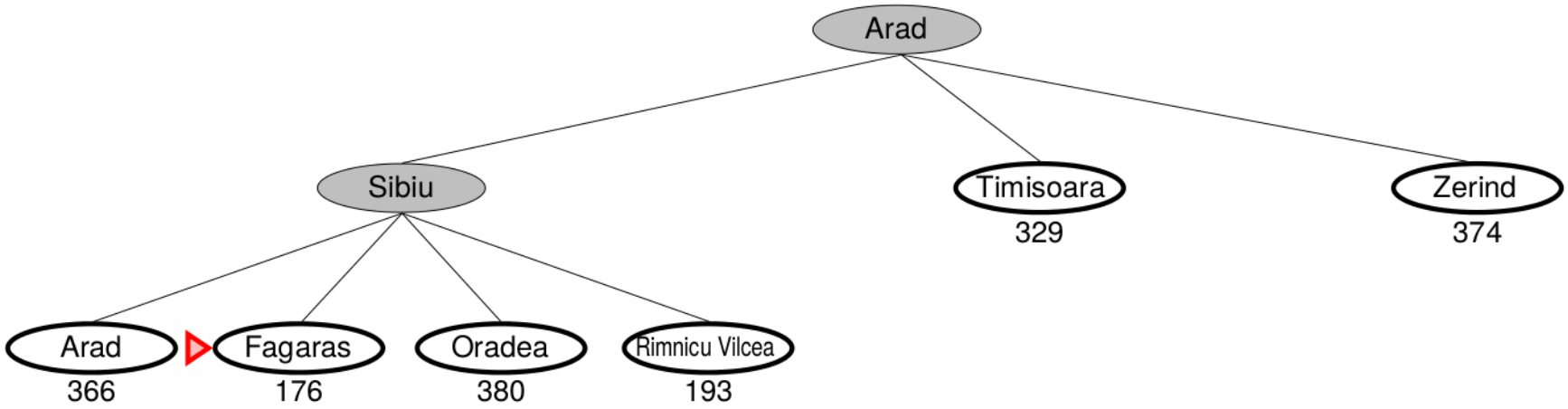
Greedy search example



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

Values of h_{SLD}—straight-line distances to Bucharest.

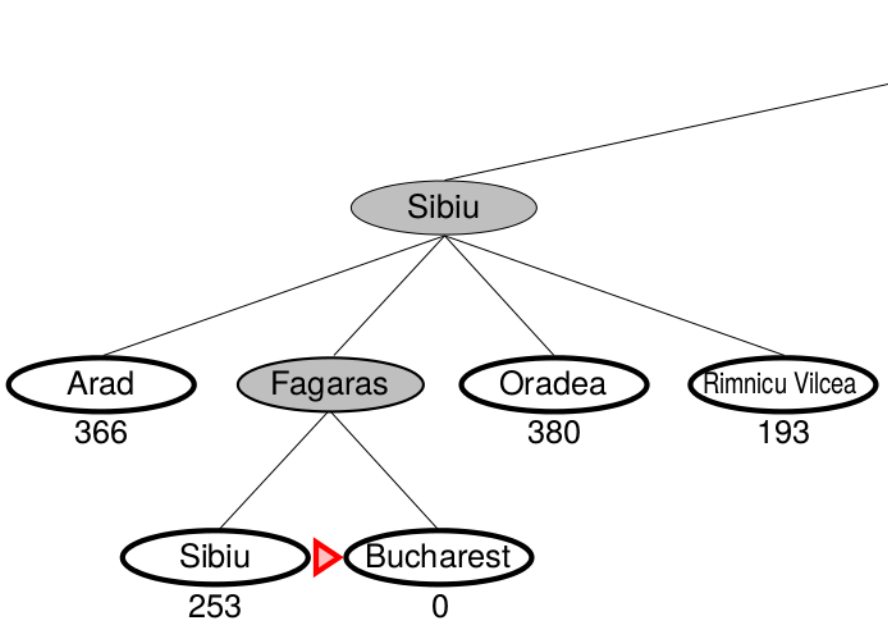
Greedy search example

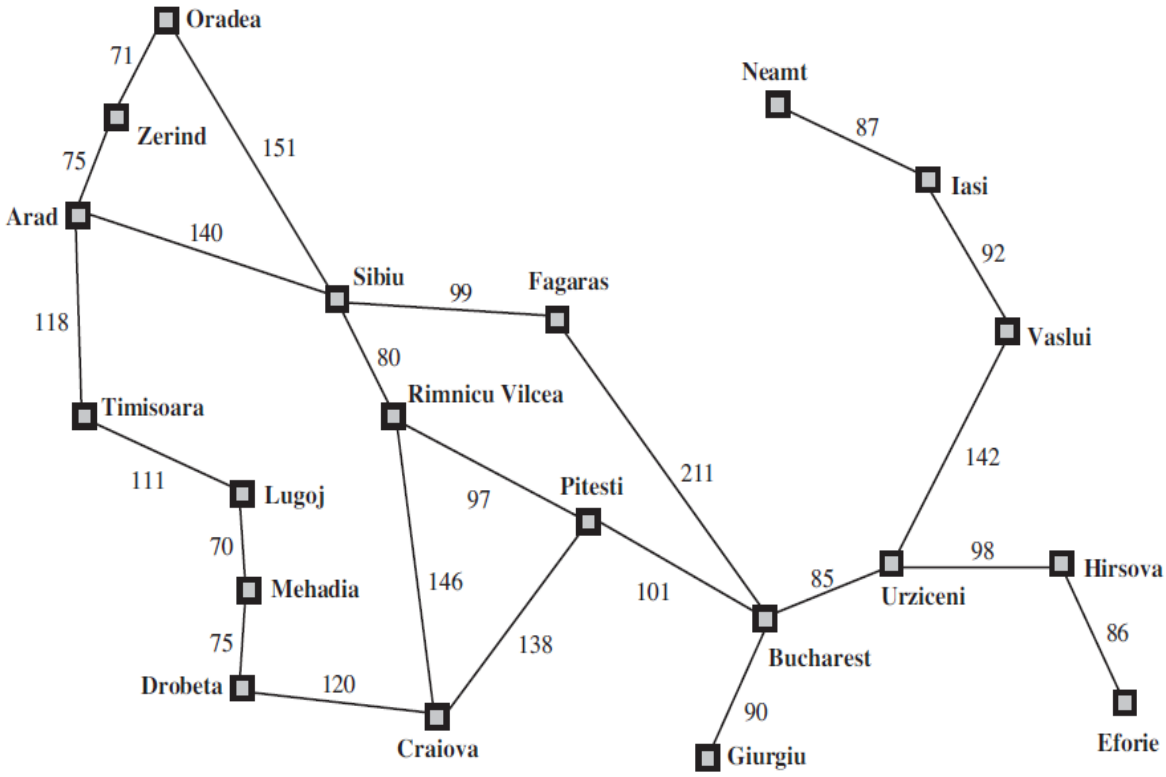


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

Greedy search example

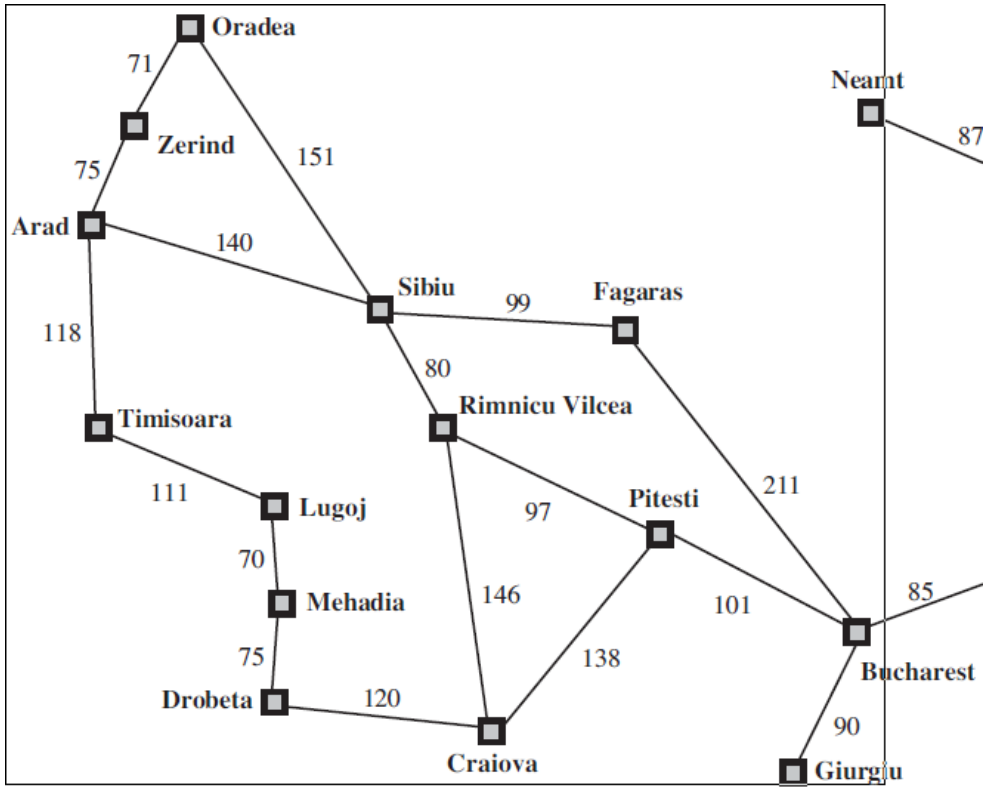




Optimal?
Greedy...
false starts? from Iasi to Fagaras

Properties of greedy search

Complete??



Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

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

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ 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??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ 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

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 -- estimated cost from start to n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

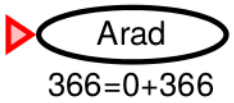
A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

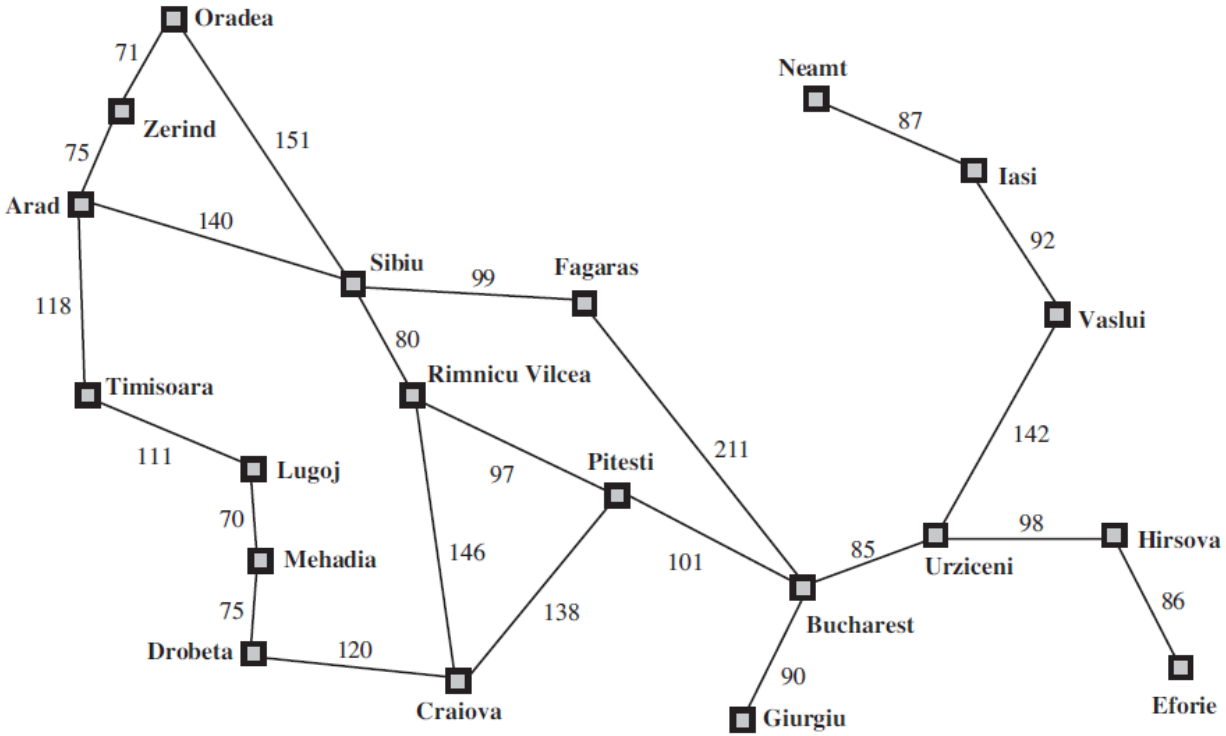
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

g(n) is the actual cost:
--> f(n) never overestimates the cost

A* search example

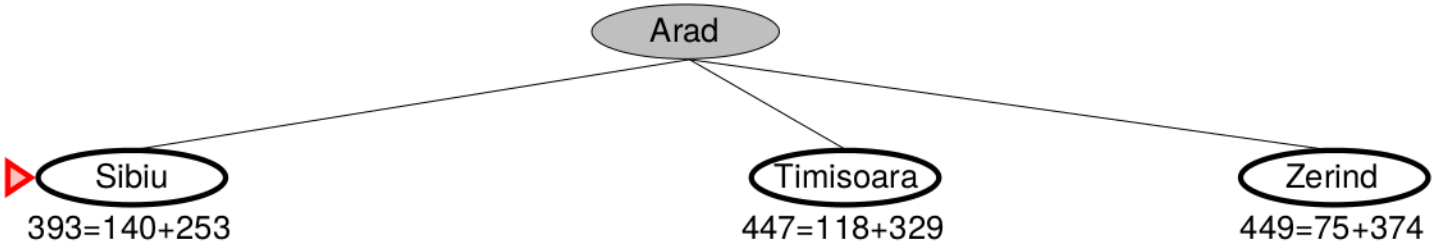


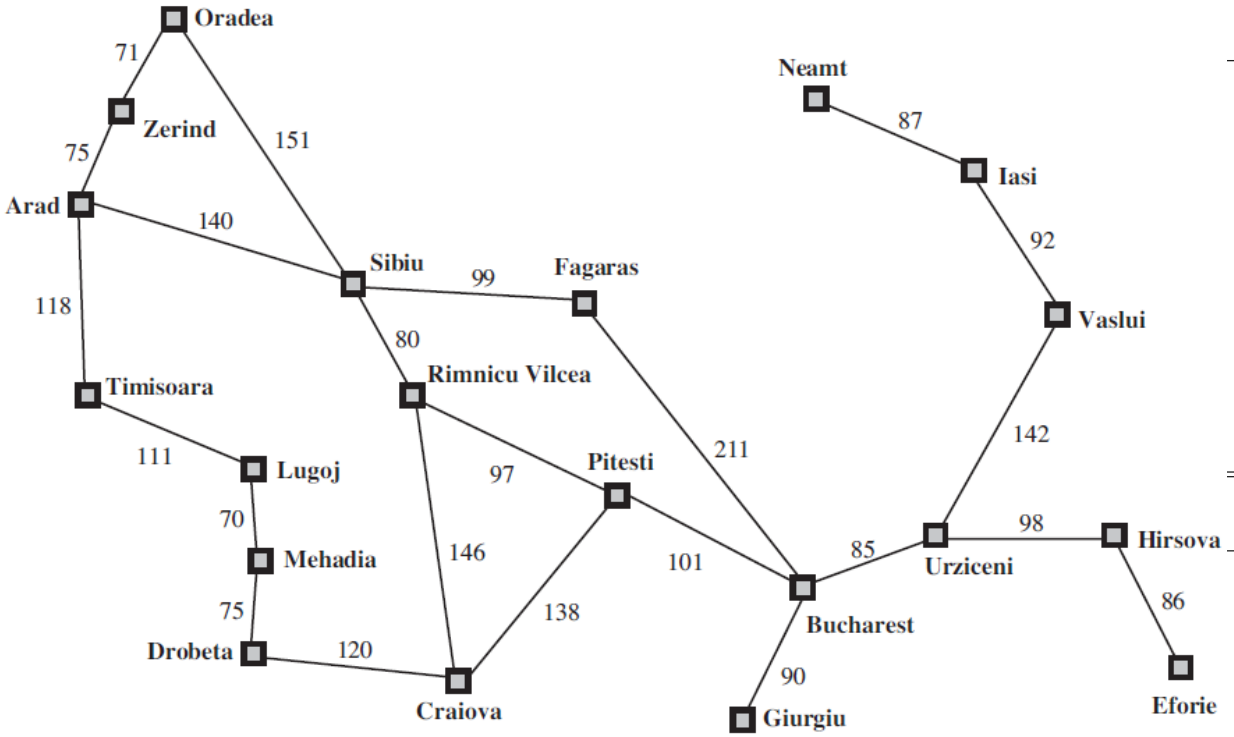


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

A* search example

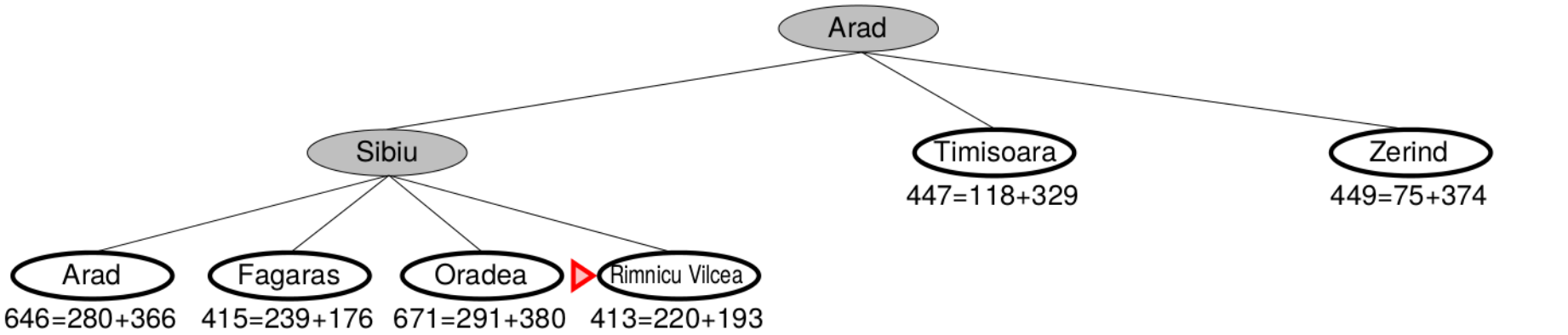


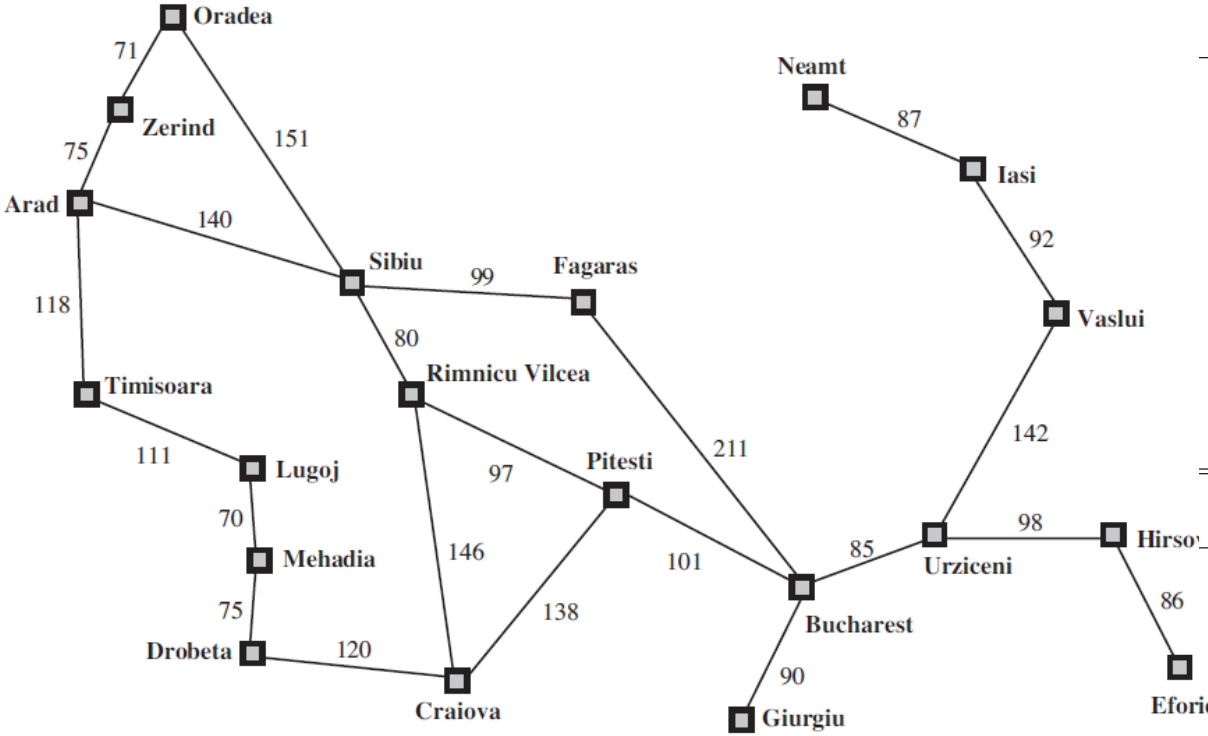


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

A* search example

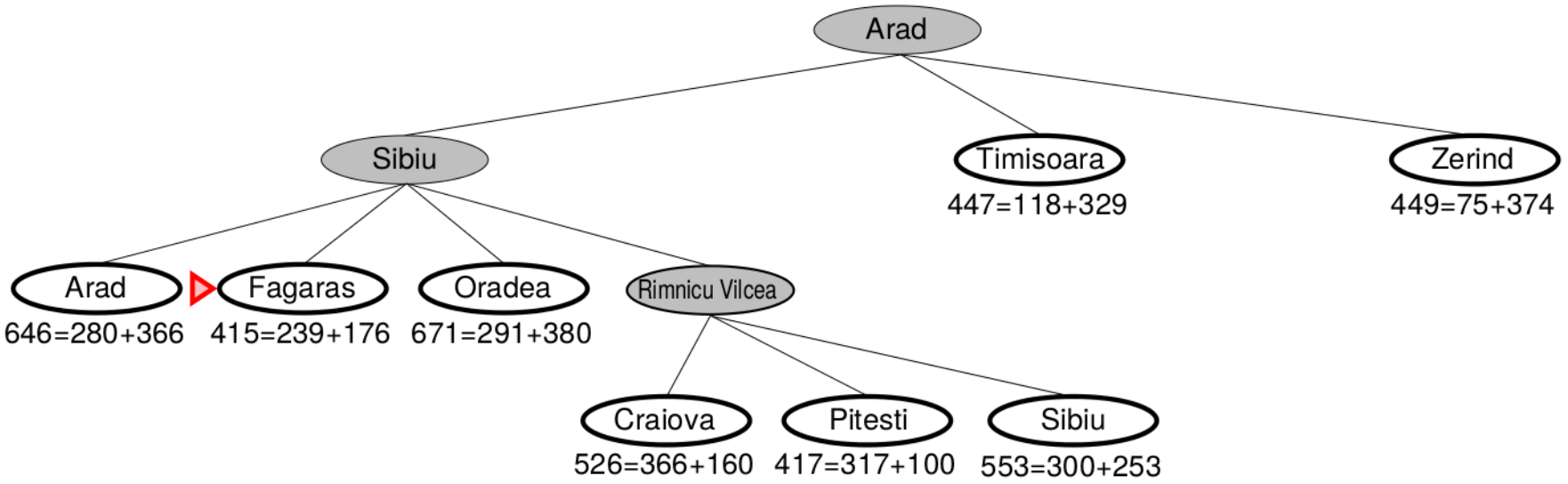




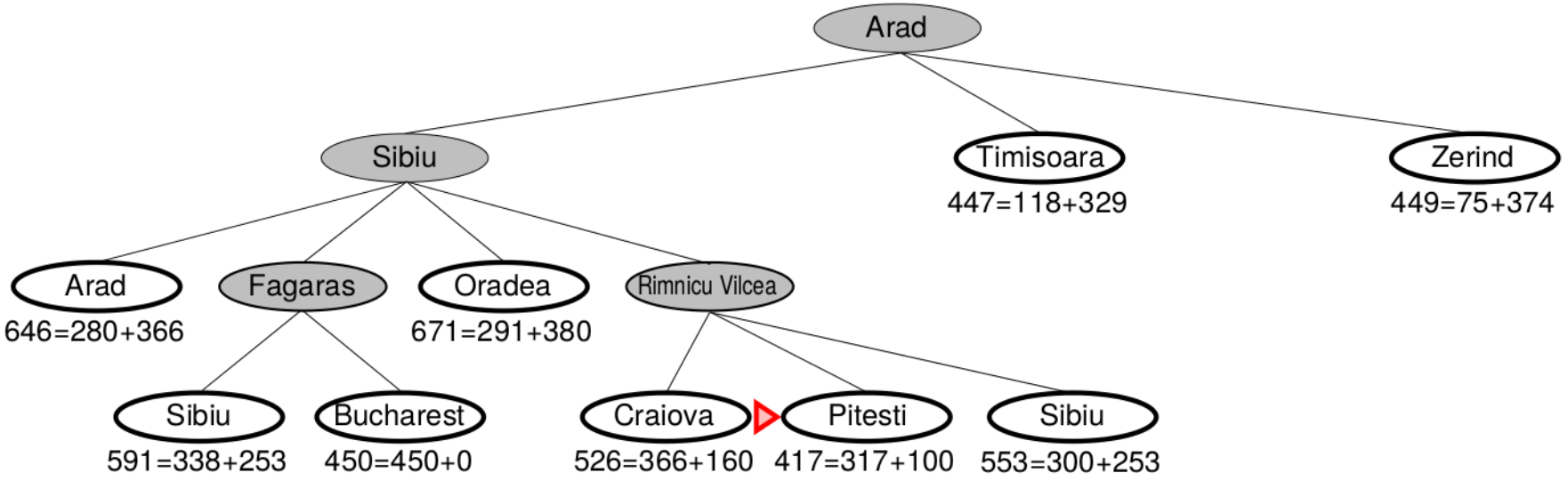
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

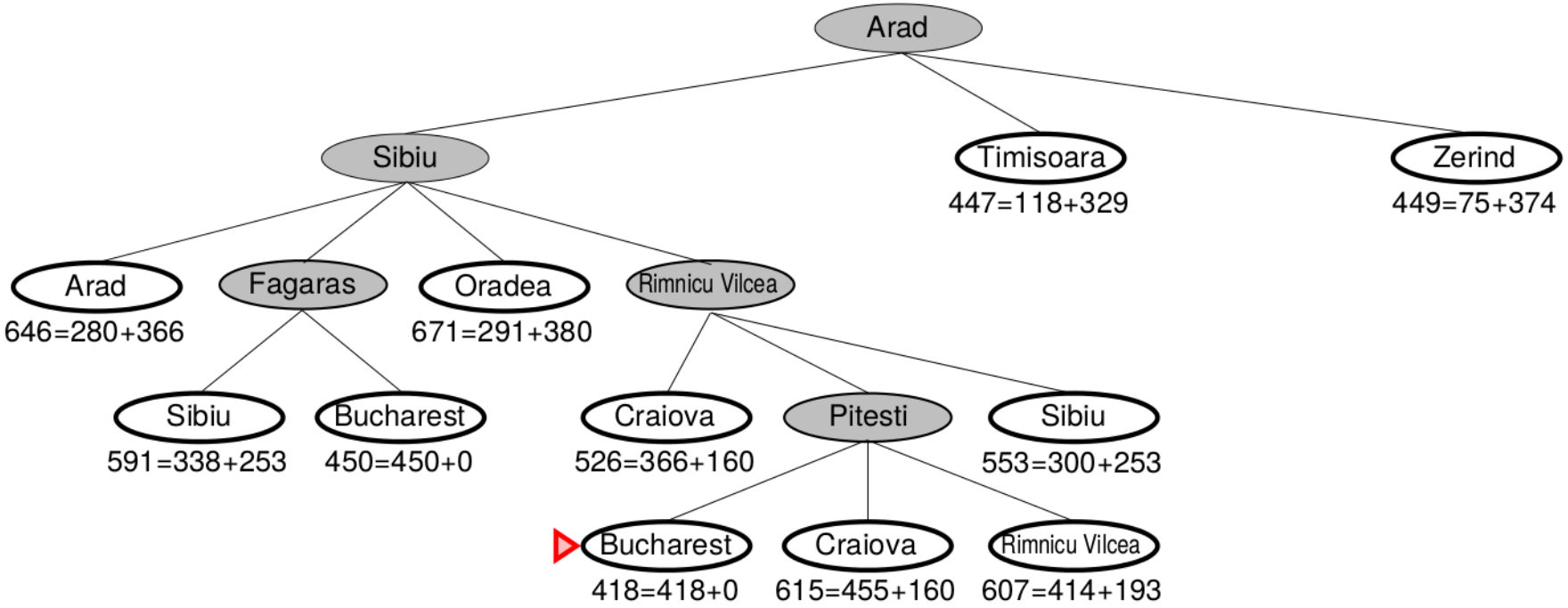
A* search example



A* search example



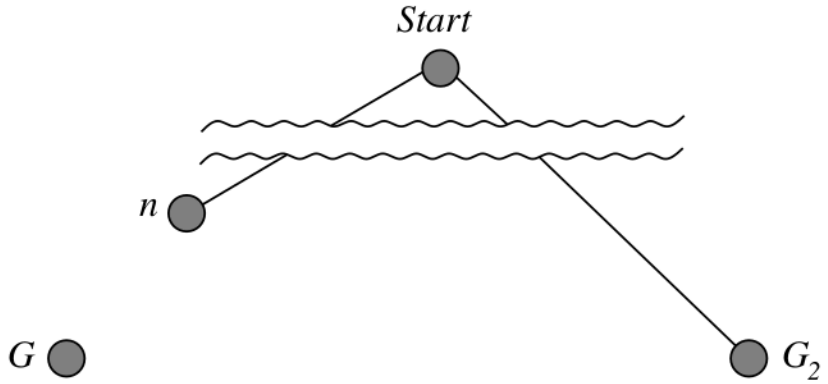
A* search example



Optimal? Yes if heuristic is
1- admissible ... never overestimates: h(n) <= h*(n)
2- consistency... h(n) < h(n') + c(n,a,n')
Every consistent heuristic is admissible.
* Admissible but inconsistent heuristics require bookkeeping to ensure optimality.

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_1.



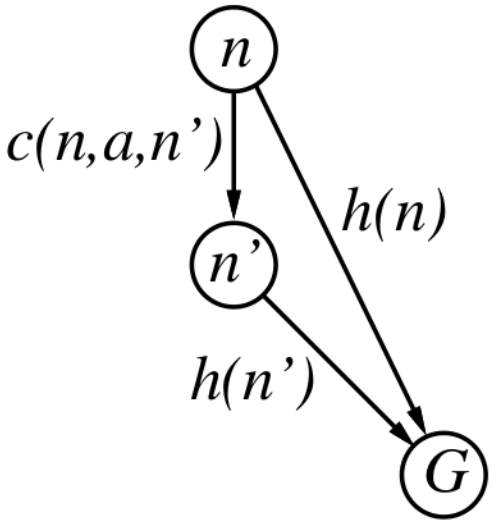
$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

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

Proof of lemma: Consistency

A heuristic is consistent if

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



If h is consistent, we have

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

I.e., $f(n)$ is nondecreasing along any path.

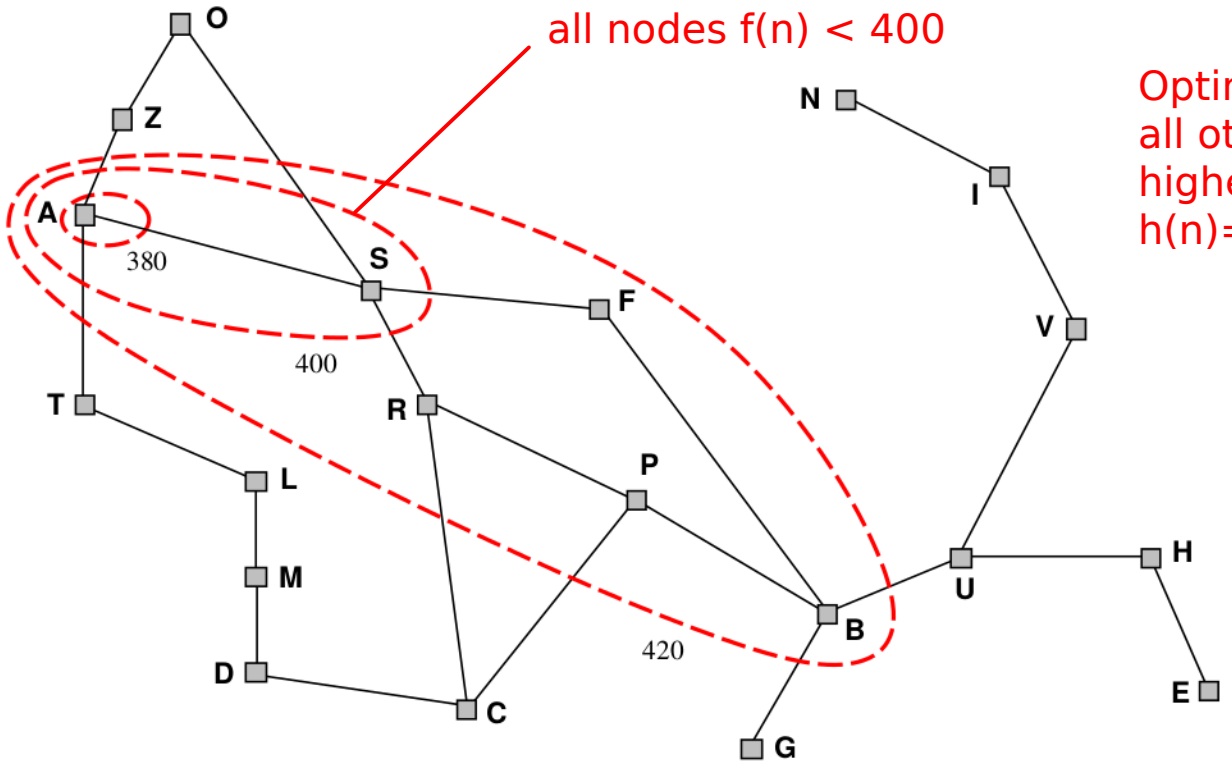
if h(n) is consistent, then the values of f(n) along any path are nondecreasing.

Consistency (Monotonicity): h(n) <= c(n,a,n')+h(n')
---> suppose n' is successor of n
f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') >= g(n) + h(n) = f(n)

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 $f_i < f_{i+1}$



Optimal:
all other goal nodes will have higher f-costs, thus g-costs (as h(n)=0).

A* expands fringe nodes wih lowest f-cost.
Heuristic = 1: shape, heuristic = direct-path .. ?

Properties of A*

Complete??

Properties of A*

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

Time??

Properties of A*

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

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

Space??

Properties of A*

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

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

Space?? Keeps all nodes in memory

Optimal??

Properties of A*

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

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

Space?? Keeps all nodes in memory

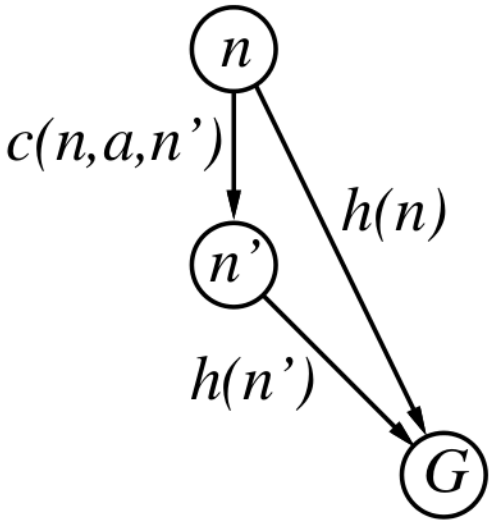
Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

Proof of lemma: Consistency

A heuristic is consistent if

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



If h is consistent, we have

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

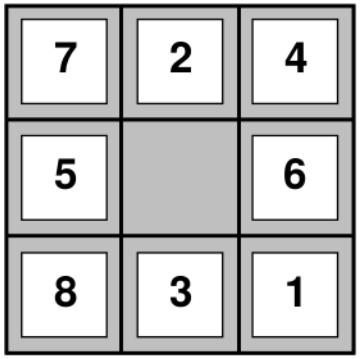
I.e., $f(n)$ is nondecreasing along any path.

if h(n) is consistent, then the values of f(n) along any path are nondecreasing.

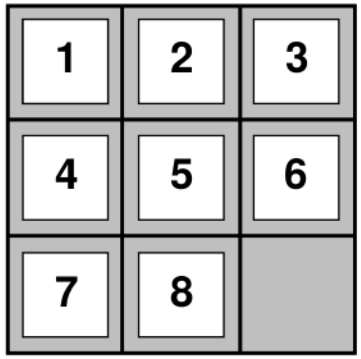
Admissible heuristics

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)



Start State



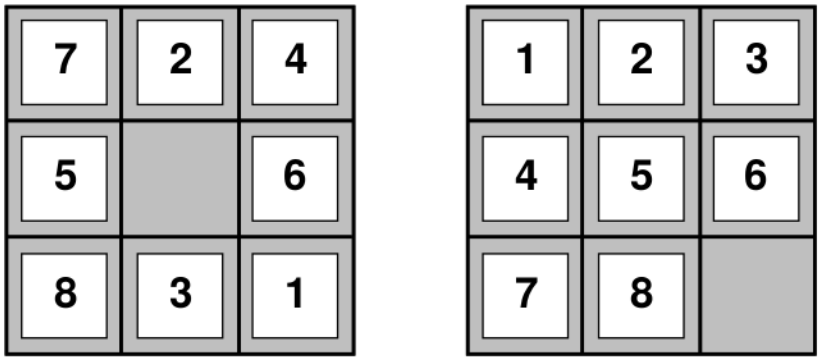
Goal State

$h_1(S) =$??
$h_2(S) =$??

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)



Start State Goal State

$h_1(S) =$?? 6
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

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 $\approx$ 54,000,000,000 nodes
A$^*(h_1)$ = 39,135 nodes
A$^*(h_2)$ = 1,641 nodes

every node expanded by h2 will also be expanded by h1.

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

QUESTION: HOW TO INVENT ADMISSIBLE HEURISTICS?

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

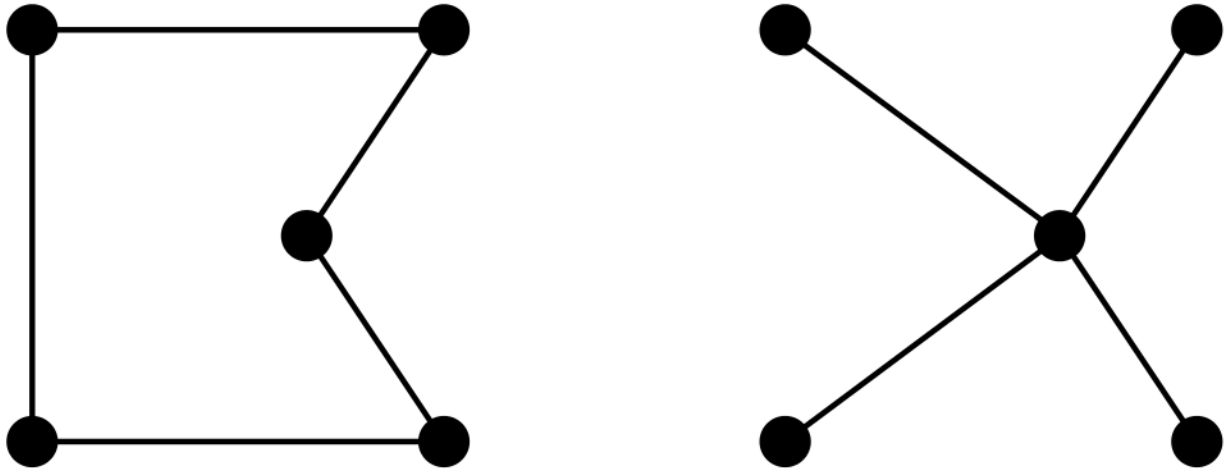
Key point: the optimal solution cost of a relaxed problem
is no greater than the optimal solution cost of the real problem

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

eg. traveling salesman problem

Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once

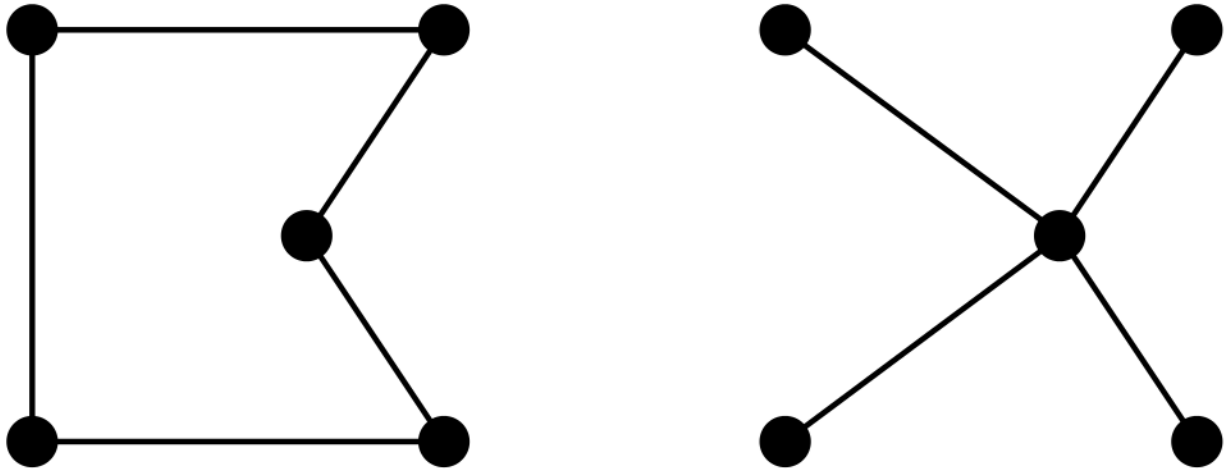


Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

8-puzzle: A tile can move from A to B if
A is adjecent to B
and B is blank

Relaxed problems contd.

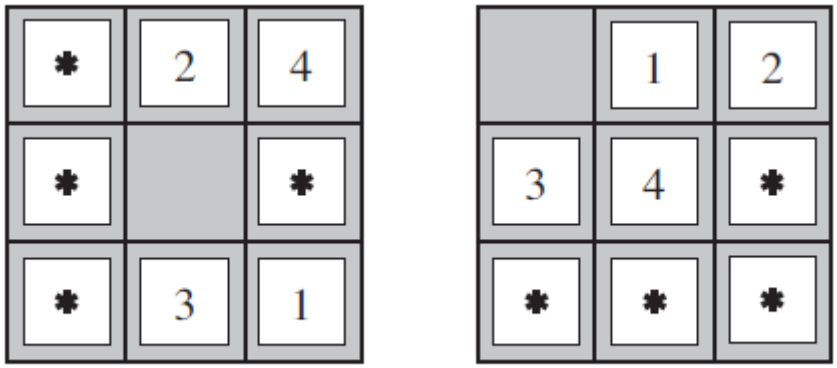
Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

eg. 8-puzzle: other admissible heuristics?
subproblem? howto combine?

Pattern Databases



- Store exact solution of every possible subproblem instance
- Lookup table
- Do this for different subproblems – take max()
- How about sum?
  - Share common moves
  - Disjoint pattern databases:
    - Random 15-puzzles in a few seconds
    - Reduced by a factor of 10,000 compared to Manhattan distance

Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

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

A* search expands lowest $g + h$
- complete and optimal
- also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems

Previous search algorithms explored state space systematically. Kept paths in the memory. Path to the goal was part of solution.

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., TSP other examples?
or, find configuration satisfying constraints, e.g., timetable

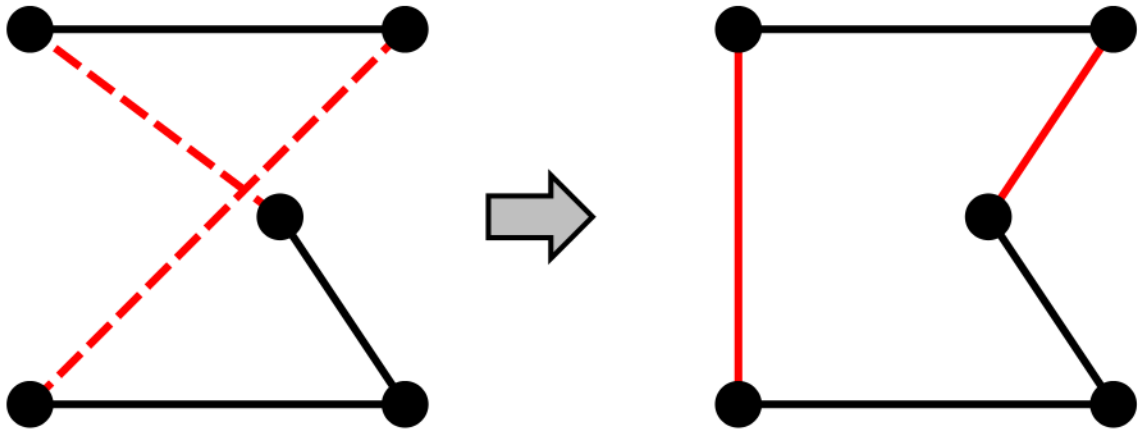
In such cases, can use iterative improvement algorithms;
keep a single "current" state, try to improve it -- local search in the neighbourhood

1 Constant space, suitable for online as well as offline search

2 Often find solutions in large or infinite (continuous) state spaces

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

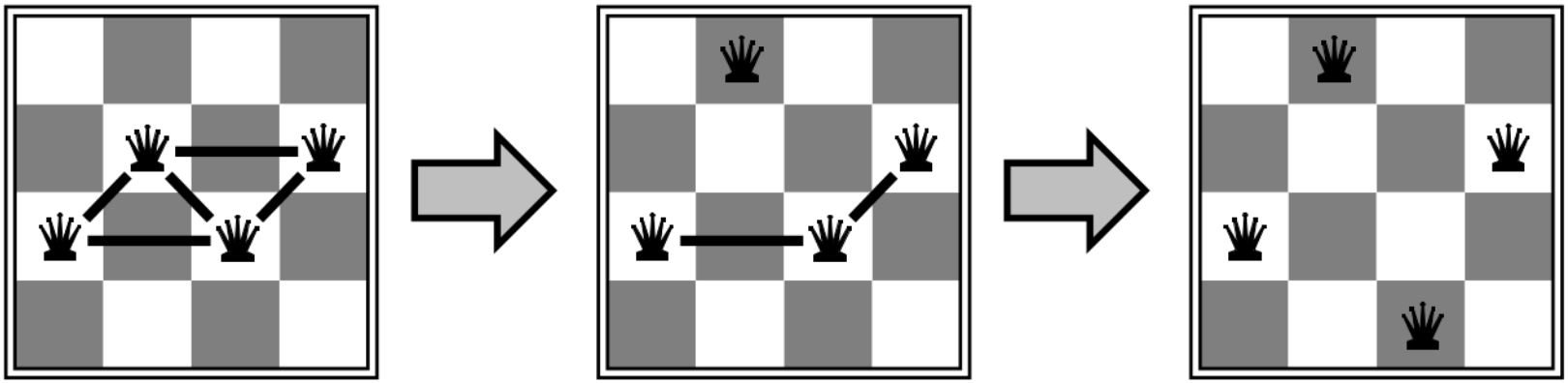


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

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

Move a queen to reduce number of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1million$

Hill-climbing (or gradient ascent/descent)

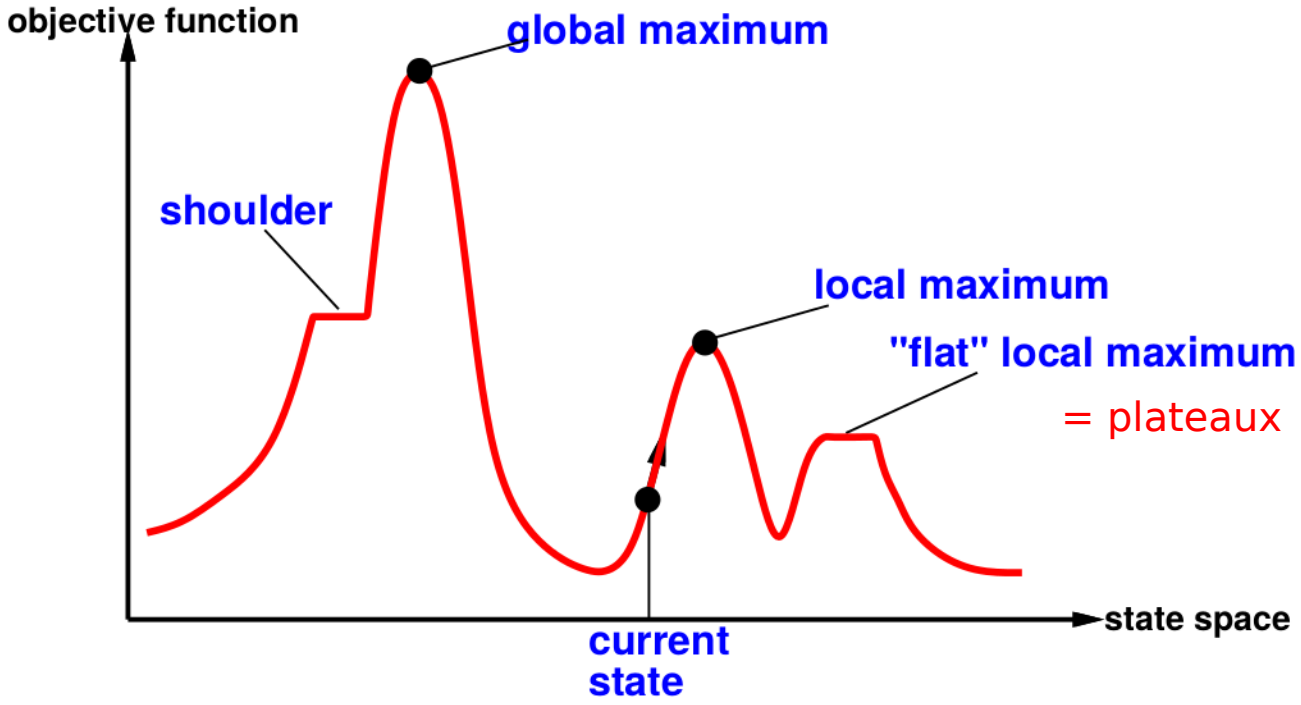
"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

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

Hill-climbing contd.

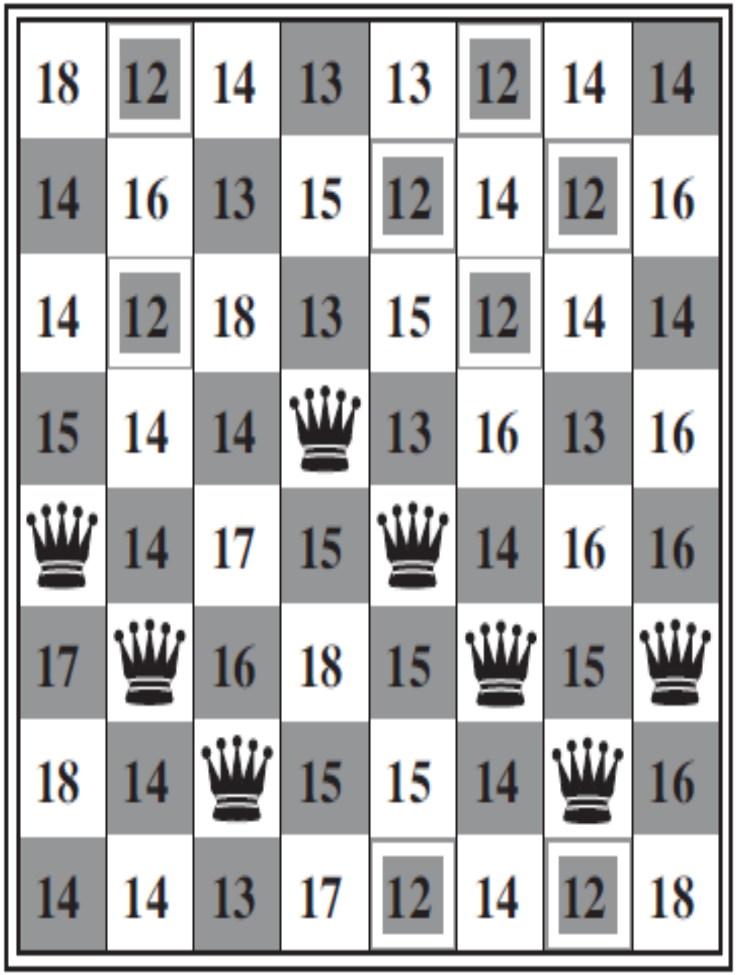
Useful to consider state space landscape



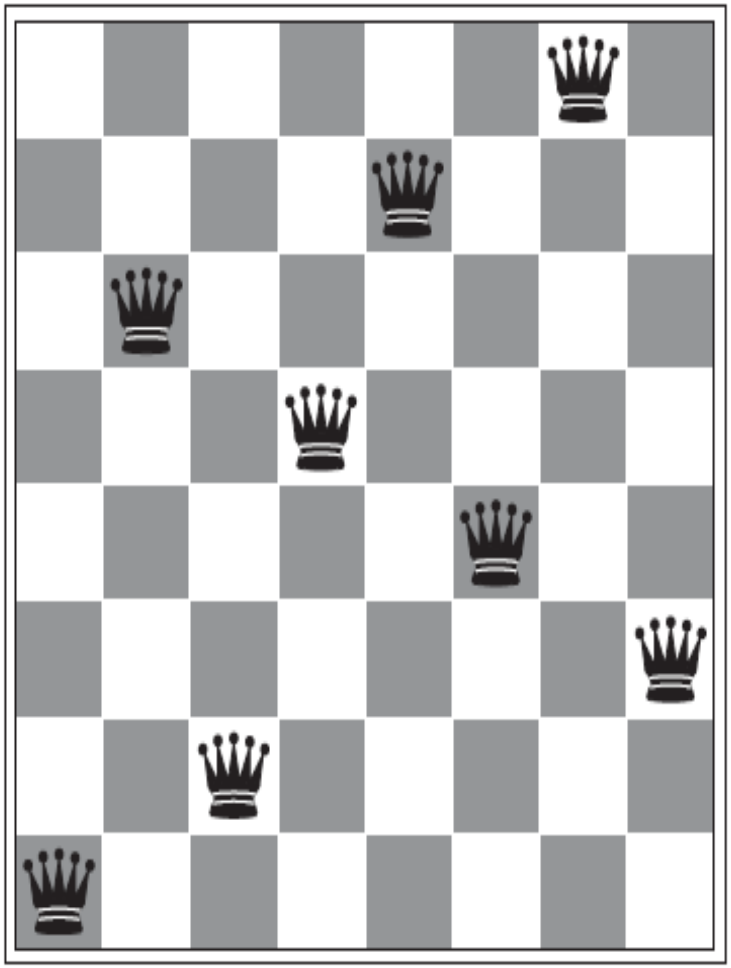
Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves escape from shoulders loop on flat maxima

Hill-climbing contd.



greedy:
5 steps
---------->



h= # of pairs attacking (direct or indirect)

local minimum

steepest-ascent solves only 14%. If limited sideway moves: 94%

Stochastic hill-climbing: select with probability steepness

Random-restart hill-climing: start-over. Expected number of restarts for 8-queens?

--> good for few local minima

Hill-climbing: incomplete: might always stuck in local optima (even stochastic)
Random-walk: complete: inefficient

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next                          .. accept if it improves
        else current ← next only with probability e^(Δ E/T)    .. accept with small prob.
                                                                  higher chance in the beginning
```

shake hard enough to bounce the ball at the local minimum
but not hard enough to dislodge it from global minimum.