Previously…

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
- Which is the best search strategy?

Comparing Tree Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

1. BFS and IDS are complete if b is finite.
2. UCS is complete if b is finite and step cost $\geq \varepsilon$
3. BFS and IDS are optimal if step costs are identical.

Choosing a Search Strategy

- Depends on the problem:
  - Finite/infinite depth of search tree
  - Known/unknown solution depth
  - Repeated states
  - Identical/non-identical step costs
  - Completeness and optimality needed?
  - Resource constraints (e.g., time, space)

Faster Search?

- Yes! Represent problem-specific knowledge as heuristics to guide search
- Today:
  - Informed (heuristic) search
  - Order of node expansion still matters: Which nodes are “more promising”?

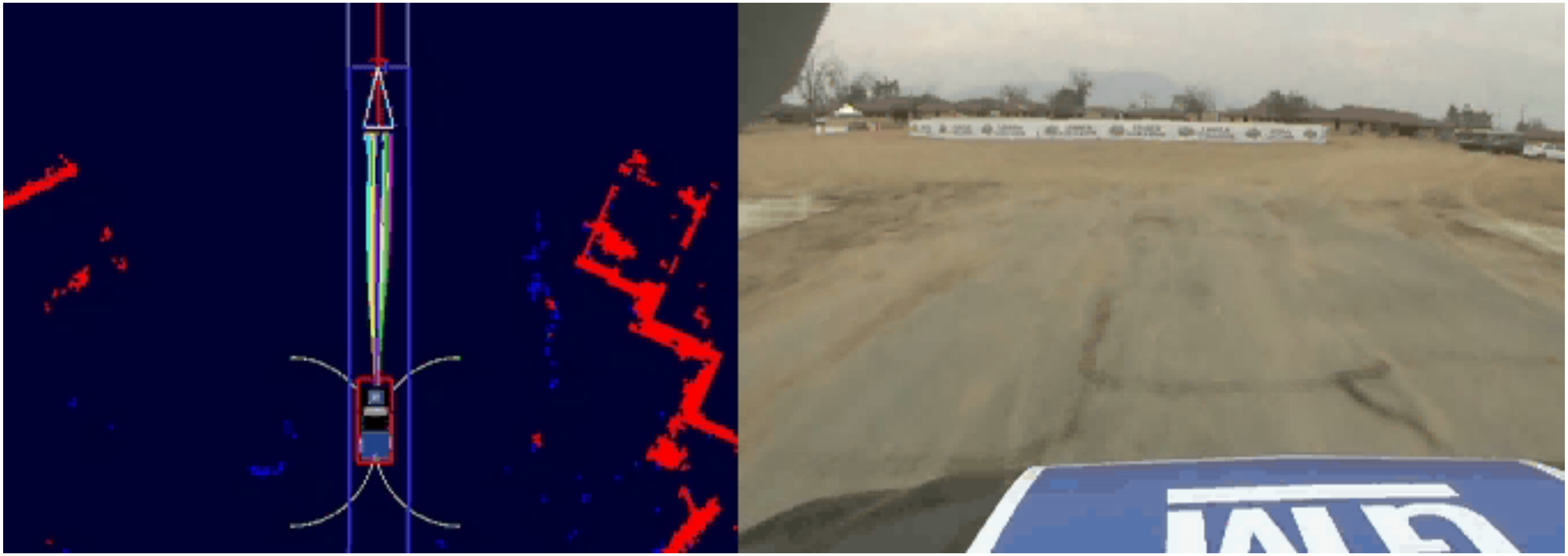
INFORMED SEARCH

AIMA Chapter 3.5.1 – 3.5.2, 3.6.1 – 3.6.2

Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics

CMU's BOSS: DARPA Urban Challenge 2007

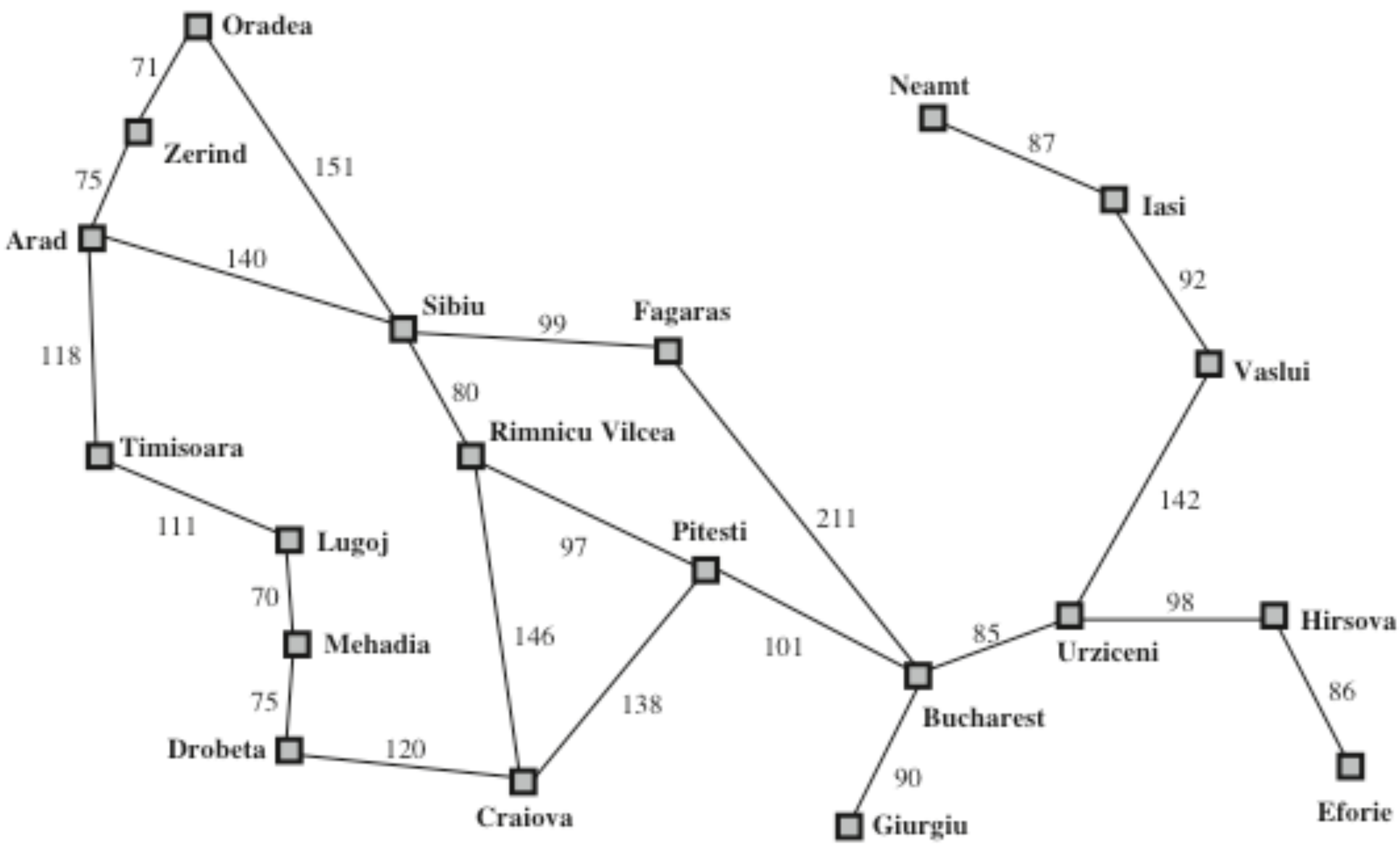


Best-First Search

- Idea: use an evaluation function $f(n)$ for each node n
  - Cost estimate
  - Expand node with lowest evaluation/cost first
- Implementation:

  Frontier = priority queue ordered by nondecreasing cost f
- Special cases (different choices of f):
  - Greedy best-first search
  - A* search

Romania with Step Costs (km)



Values of h_{SLD} - 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

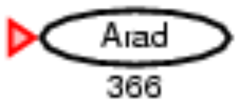
Greedy Best-First Search

- Evaluation function $f(n)$

  $= h(n)$ (heuristic function)

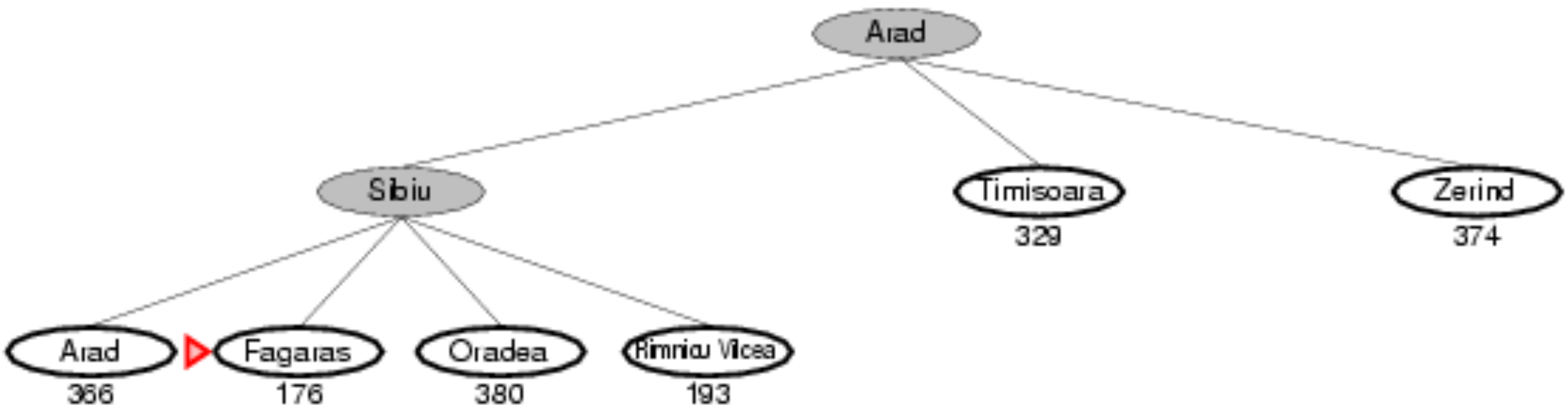
  = estimated cost of cheapest path from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Greedy Best-First Search Example

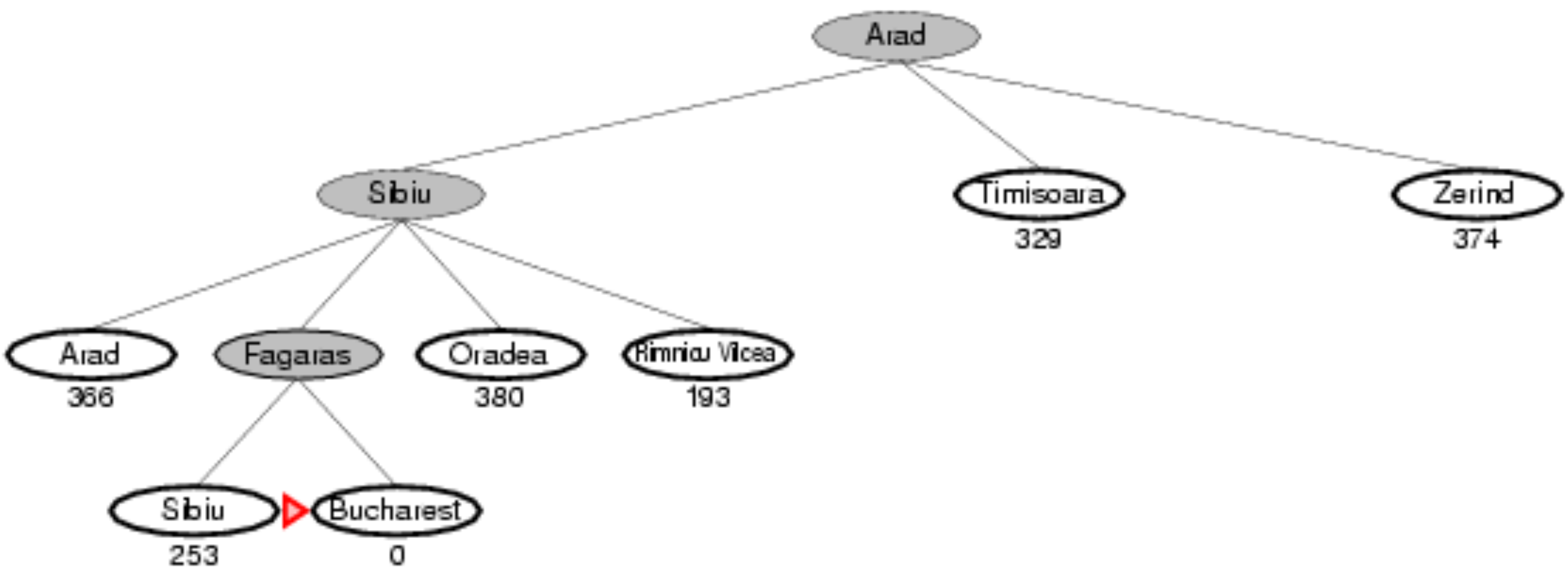


Greedy Best-First Search Example

Greedy Best-First Search Example



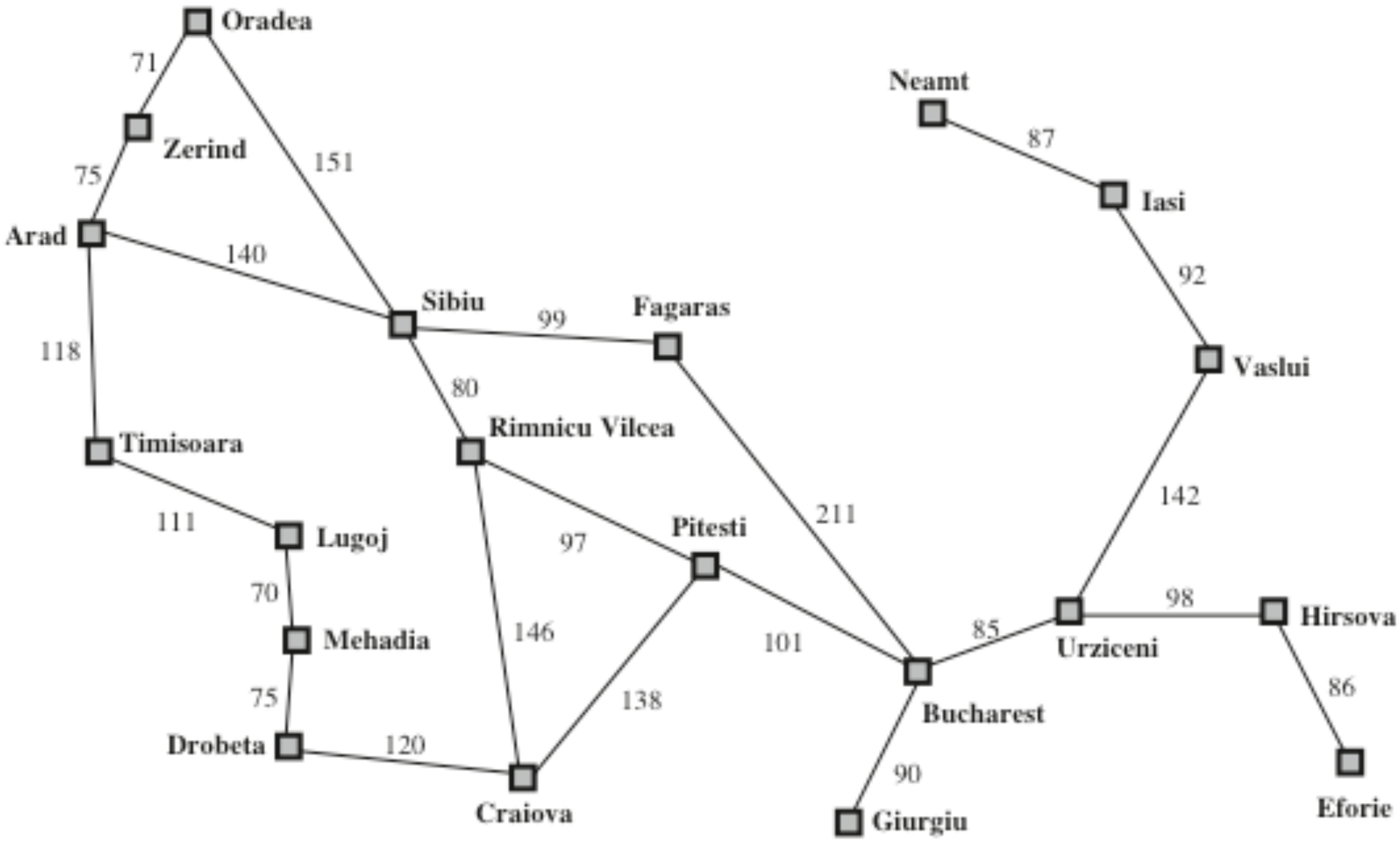
Greedy Best-First Search Example



Properties of Greedy Best-First Search

- Complete? No – can get stuck in infinite loop, e.g., to get from Iasi to Fagaras, Iasi → Neamt → Iasi → Neamt → …
- Optimal? No
- Time? $O(b^m)$, but a good heuristic can reduce complexity substantially
- Space? $O(b^m)$: keeps frontier nodes in memory

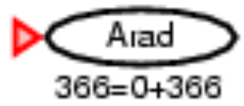
Romania with Step Costs (km)



A* Search

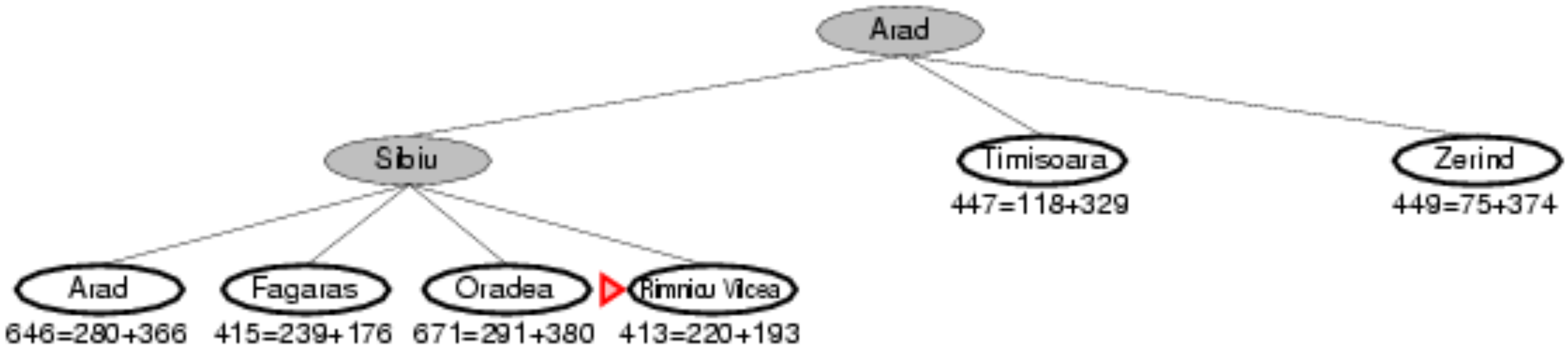
- Idea: Avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = path cost from start node to reach n
- $h(n)$ = estimated cost of cheapest path from n to goal
- $f(n)$ = estimated cost of cheapest path through n to goal

A* Search Example

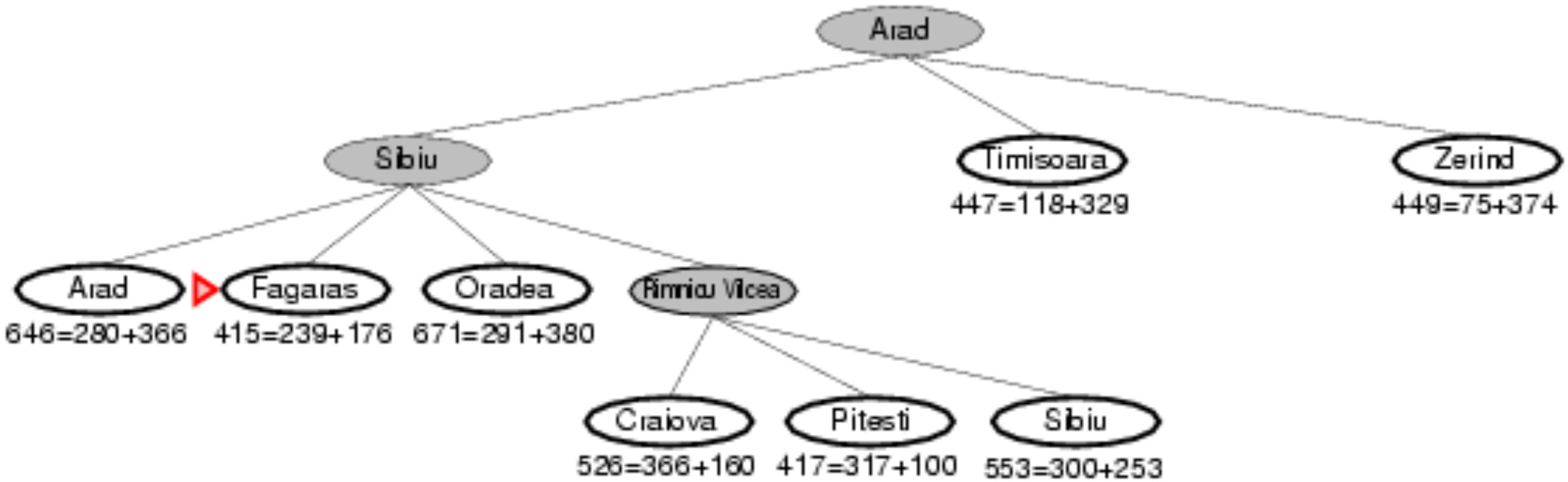


A* Search Example

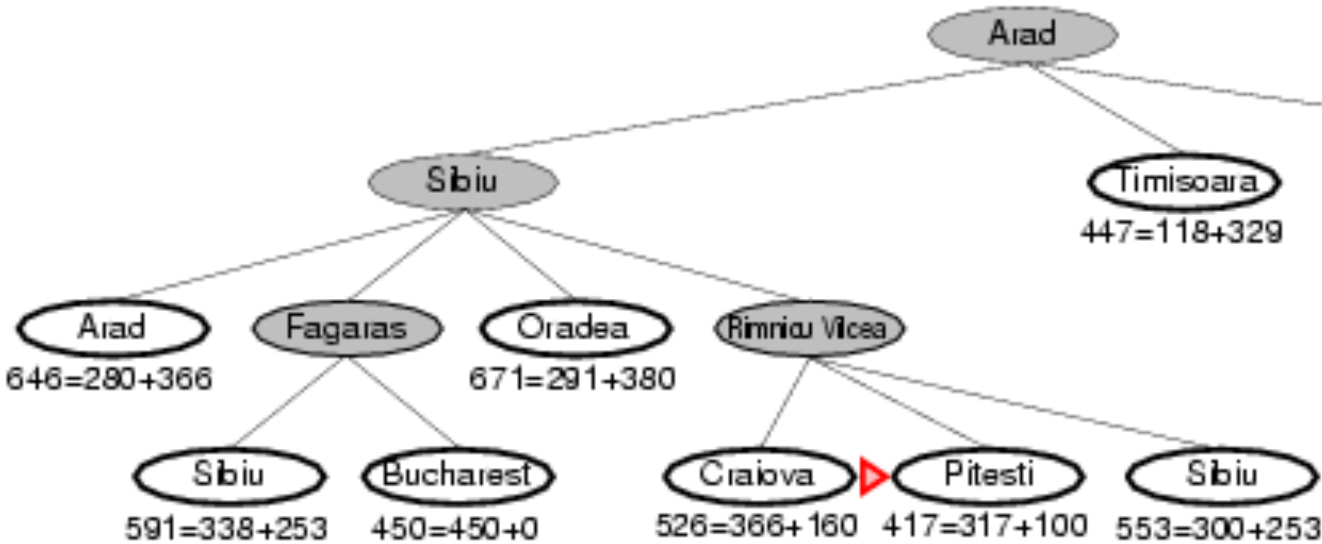
A^* Search Example



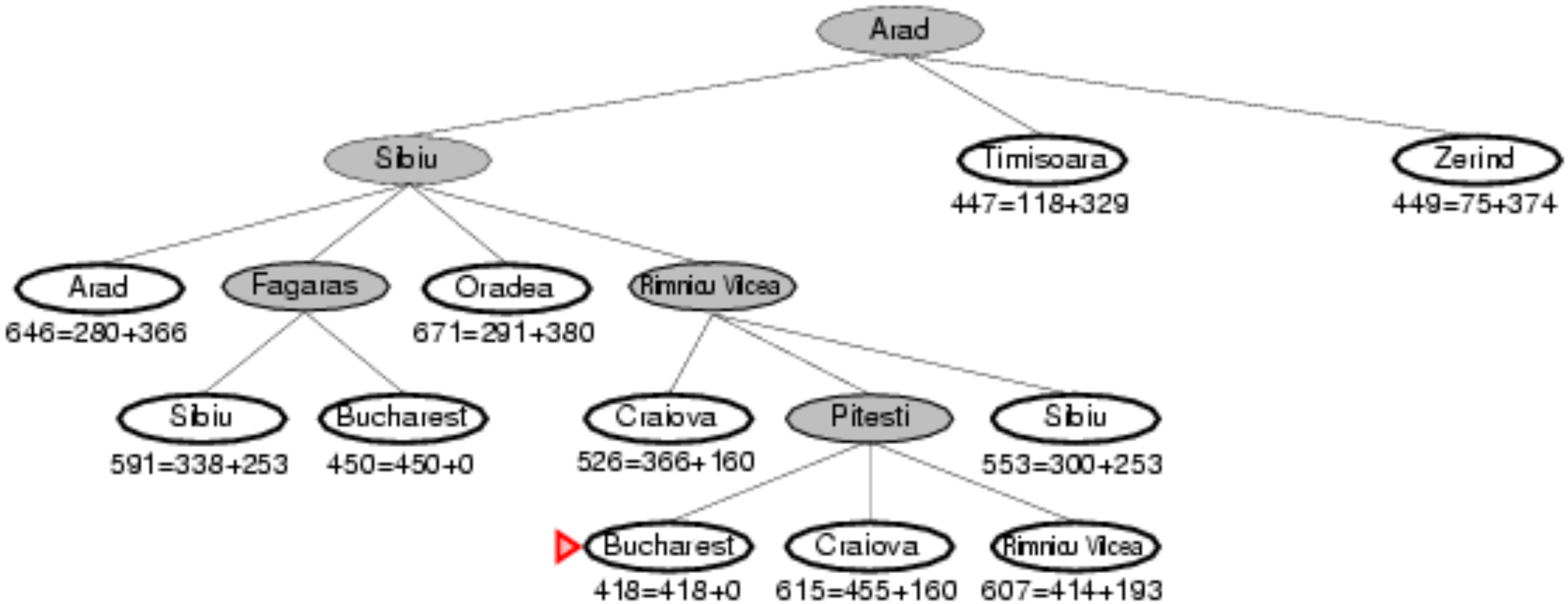
A* Search Example



A* Search Example



A* Search Example



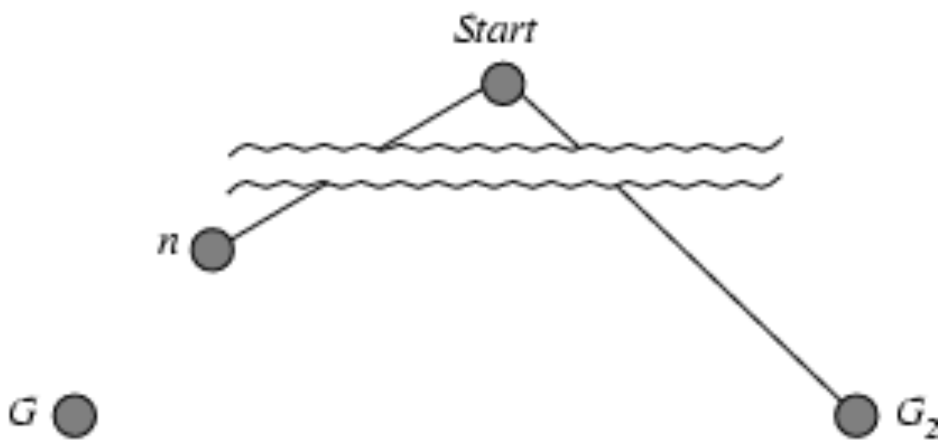
Admissible Heuristics

- 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
- Theorem: If $h(n)$ is admissible, then A* using TREE-SEARCH is optimal

Optimality of A* using TREE-SEARCH

Proof Sketch:

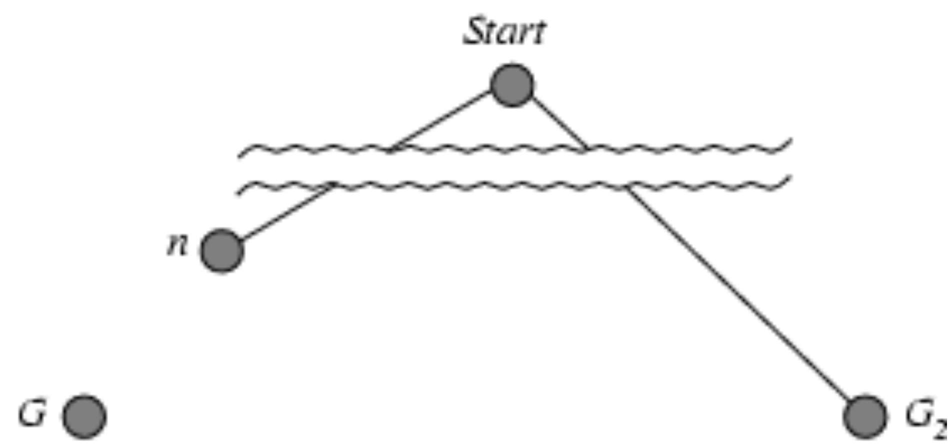
- Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G_2) > f(G)$ from above

Optimality of A* using TREE-SEARCH

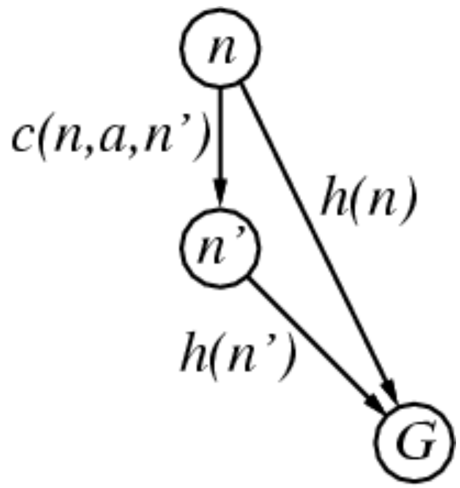
- Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



- $f(G_2) > f(G)$ from previous slide
- $h(n) \leq h^*(n)$ 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

Consistent Heuristics

- A heuristic is consistent if, for every node n and 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

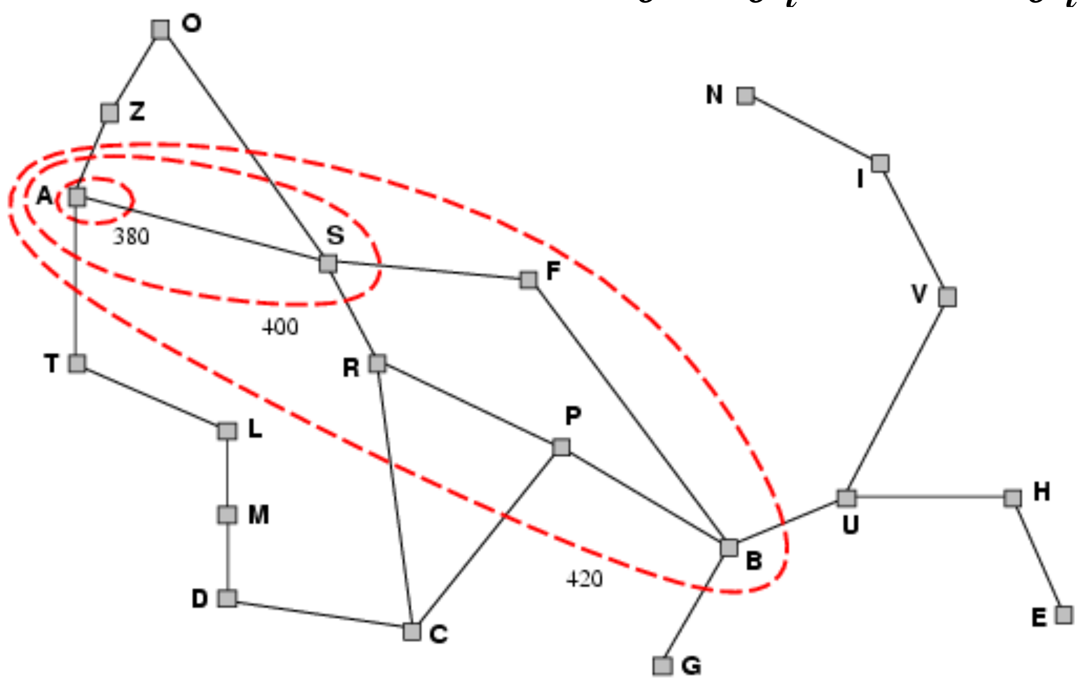


$$f(n') = g(n') + h(n')$$
$$= g(n) + c(n, a, n') + h(n')$$
$$\ge g(n) + h(n) = f(n)$$

- i.e., $f(n)$ is non-decreasing along any path.
- Theorem: If $h(n)$ is consistent, then A* using GRAPH-SEARCH is optimal

Optimality of A* using GRAPH-SEARCH

- $f(n)$ is non-decreasing along any path
- A* expands nodes in nondecreasing order of f value
- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f = f_i$ where $f_i < f_{i+1}$



Optimality of A* using GRAPH-SEARCH

Proof Sketch:

1. If $h(n)$ is consistent, then $f(n)$ is non-decreasing along any path
2. Whenever A* selects a node n (e.g., repeated state) for expansion, the optimal path to n has been found
3. A* expands nodes in nondecreasing order of $f(n)$
4. First expanded goal node is optimal solution:
    a. f is true cost for goal nodes ($h(G) = 0$)
    b. all later goal nodes are at least as expensive

Properties of A*

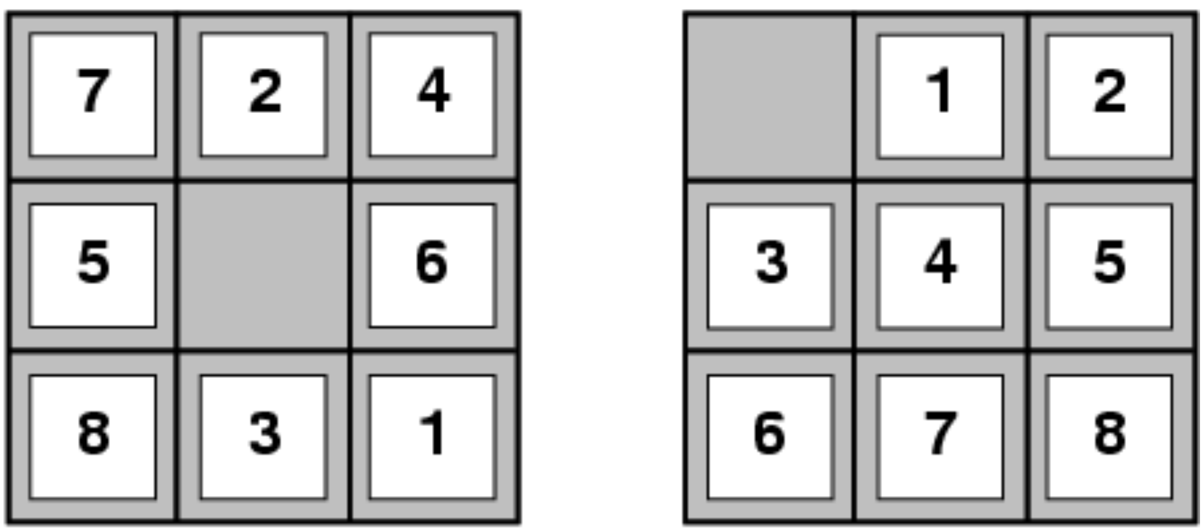
- Complete? Yes (if there are finitely many nodes with $f(n) \leq f(G)$)
- Optimal? Yes
- Time? $O(b^{h^*-h})$ where h^* is actual cost of getting from root to goal
- Space? Keeps all generated nodes in memory

Admissible vs. Consistent Heuristics

- Why is consistency a stronger sufficient condition than admissibility? How are they related?
  - A consistent heuristic is admissible
  - An admissible heuristic MAY be inconsistent
- An admissible but inconsistent heuristic cannot guarantee optimality of A* using GRAPH-SEARCH
  - Problem: GRAPH-SEARCH discards new paths to a repeated state. So, it may discard the optimal path. Previous proof breaks down.
  - Solution: Ensure that optimal path to any repeated state is always followed first.

Admissible Heuristics

- Let's revisit the 8-puzzle
  - Branching factor is about 3
  - Average solution depth is about 22 steps
  - Exhaustive tree search examines 3^{22} states
- How do we come up with good heuristics?

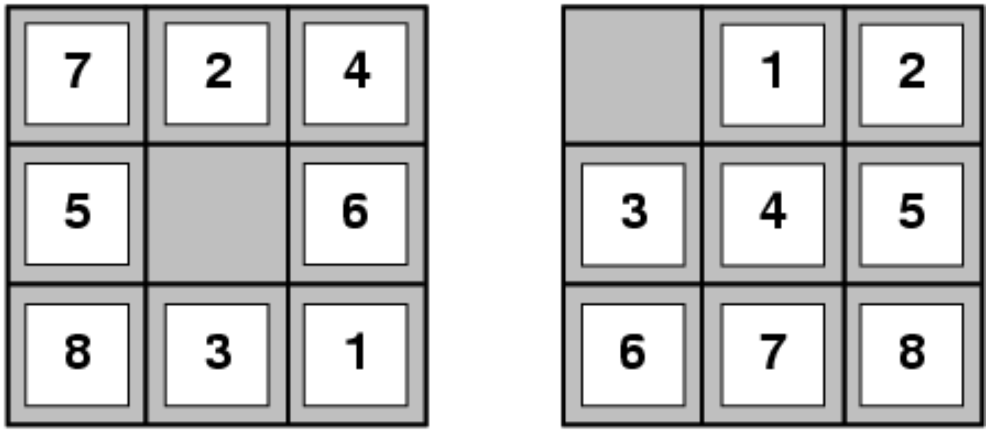


Start State   Goal State

Admissible Heuristics

E.g., 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

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 dominates h_1. It follows that h_2 incurs lower search cost than h_1.

Average search costs (nodes generated):

- $d = 12$ IDS = 3,644,035 nodes
  $A^*(h_1) = 227$ nodes
  $A^*(h_2) = 73$ nodes

- $d = 24$ IDS = too many nodes
  $A^*(h_1) = 39{,}135$ nodes
  $A^*(h_2) = 1{,}641$ nodes

Deriving Admissible Heuristics

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

Deriving Admissible Heuristics

- Rules of 8-puzzle:

  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

  1. A tile can move from square A to square B if A is adjacent to B
  2. A tile can move from square A to square B if B is blank
  3. A tile can move from square A to square B

Deriving Admissible Heuristics

- 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

LOCAL SEARCH

AIMA Chapter 4.1

Outline

- Local search algorithms
  - Hill-climbing search
  - Simulated annealing
  - Local beam search
  - Genetic algorithms

Local Search Algorithms

- In many optimization problems, the path to goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find final configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms to keep a single "current" state and try to improve it
- Advantages: (1) very little/constant memory, and (2) find reasonable solutions in large state space

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 Search

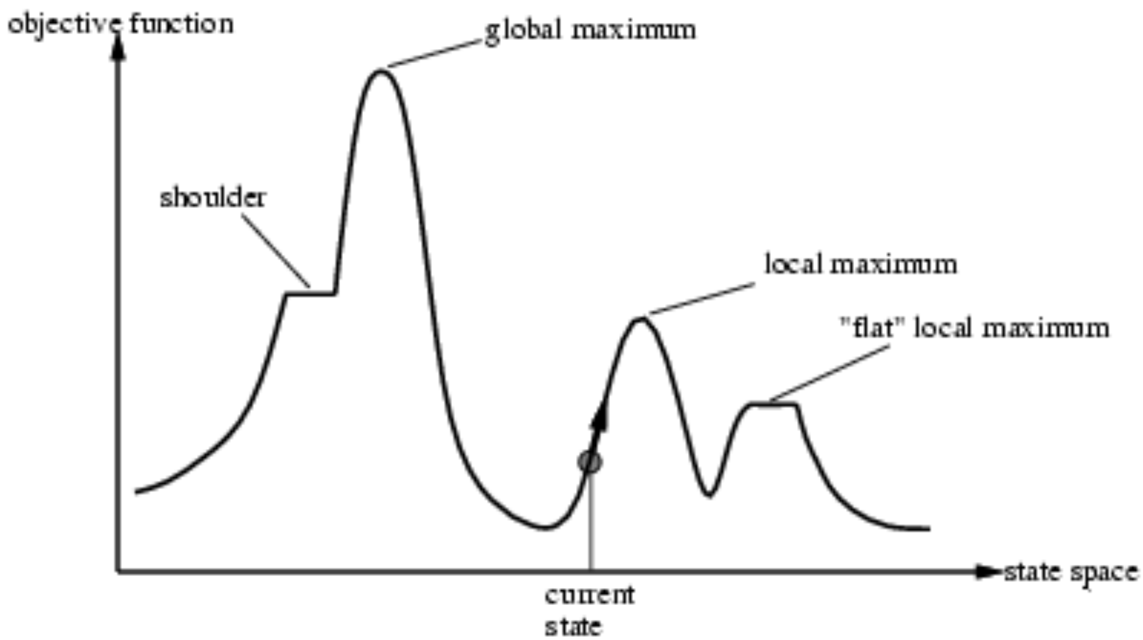
- "Like climbing Mt. Everest in thick fog with amnesia"

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

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

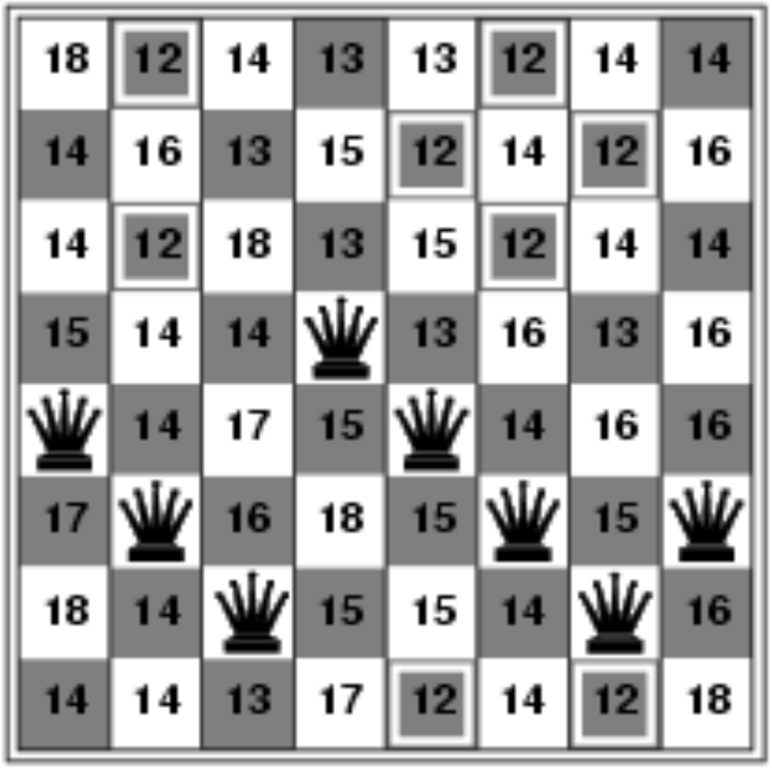
Hill-Climbing Search

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



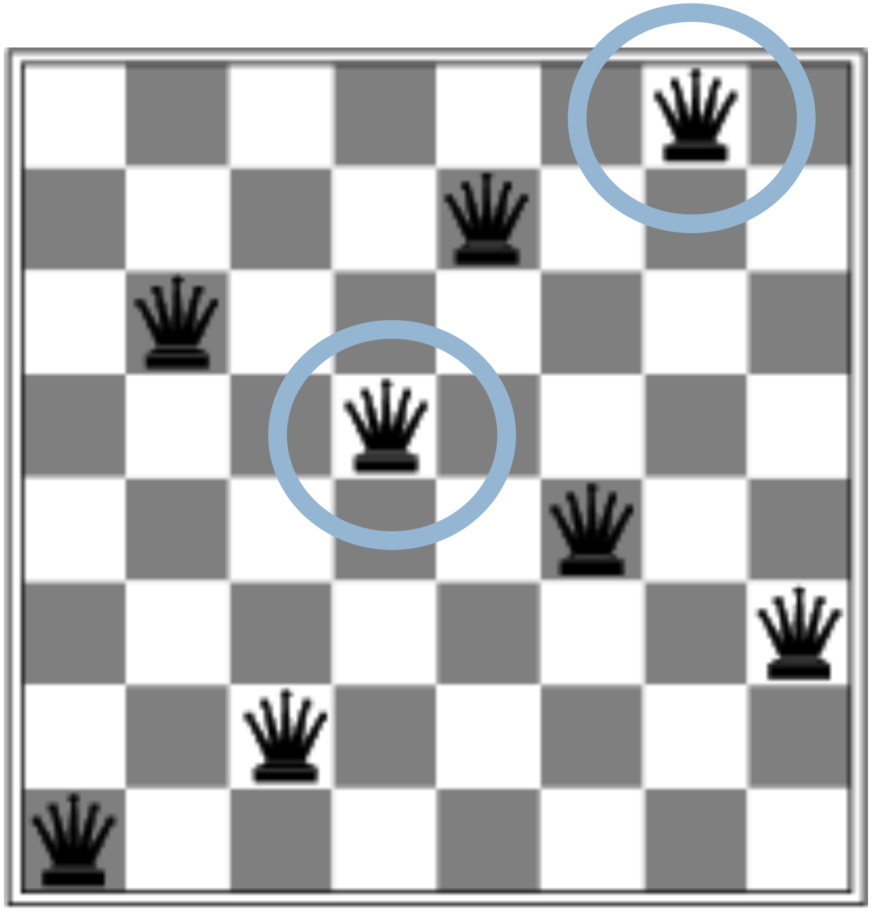
- Non-guaranteed fixes: sideway moves, random restarts

Hill-Climbing Search: 8-Queens



- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-Climbing Search: 8-Queens



A local minimum with $h = 1$

Simulated Annealing

- Idea: escape local maxima by allowing some “downhill” moves but gradually decrease their frequency

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to “temperature”

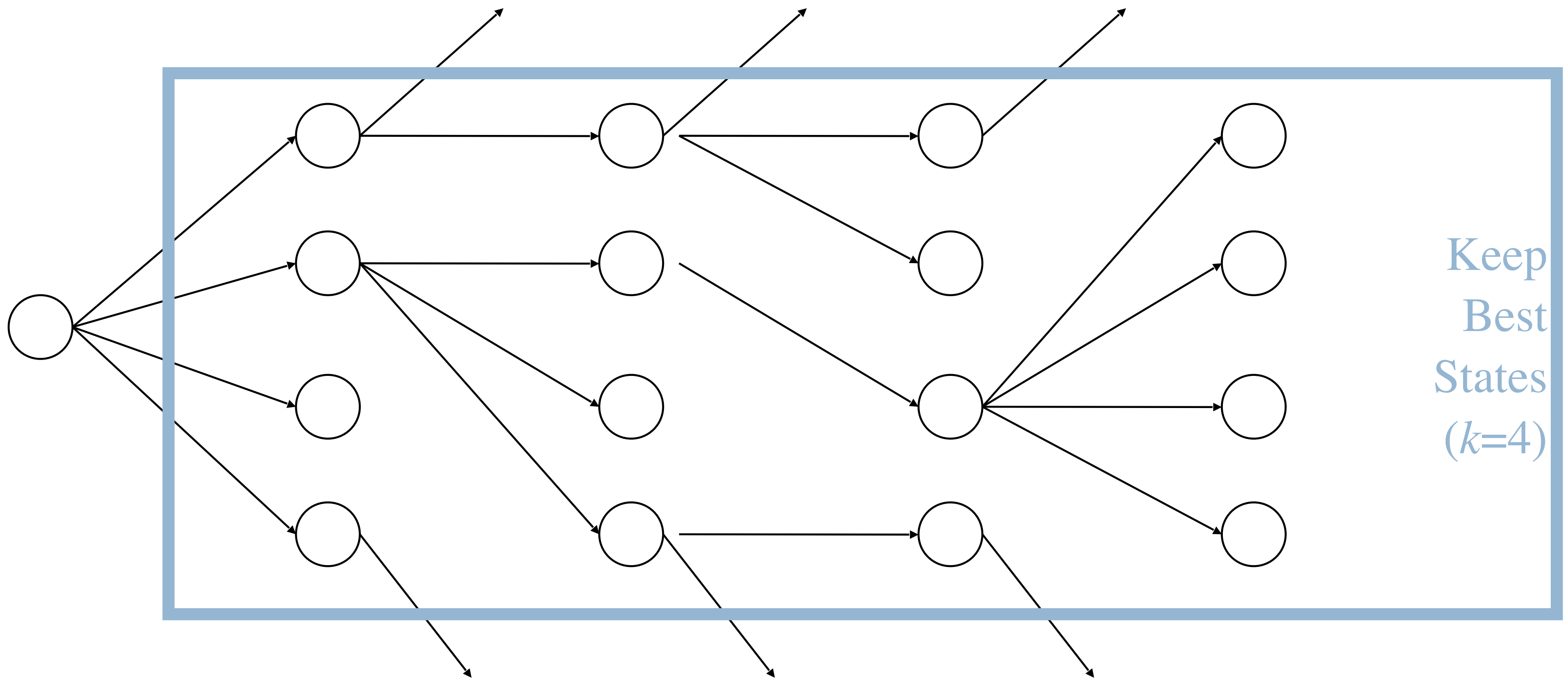
  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Properties of Simulated Annealing

- Can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, factory scheduling, and other large-scale optimization tasks

Local Beam Search

- Why keep just one best state?



- Problem: May quickly concentrate in small region of state space → Solution: stochastic beam search

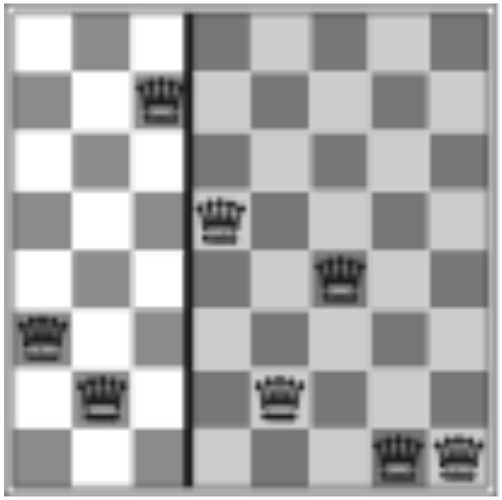
Genetic Algorithms

- Idea: a successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function): higher values for better states
- Produce the next generation of states by selection, crossover, and mutation

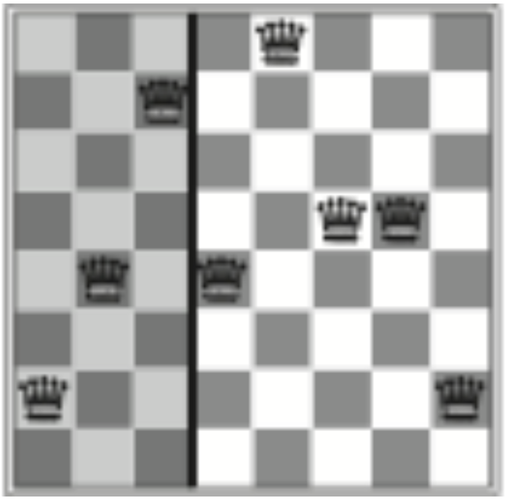
Genetic Algorithms Example

- Fitness function: number of non-attacking pairs of queens (min = 0, max = $(8 \times 7)/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc.

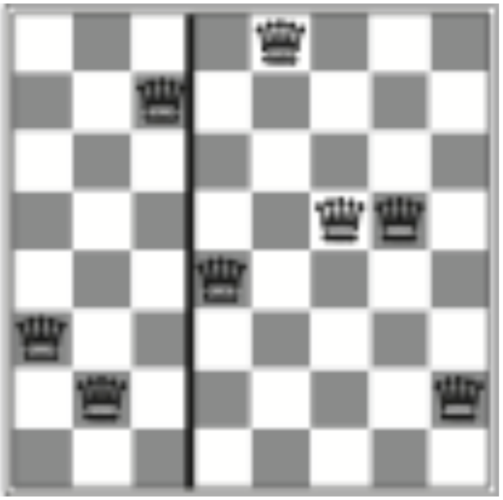
Genetic Algorithms: Crossover



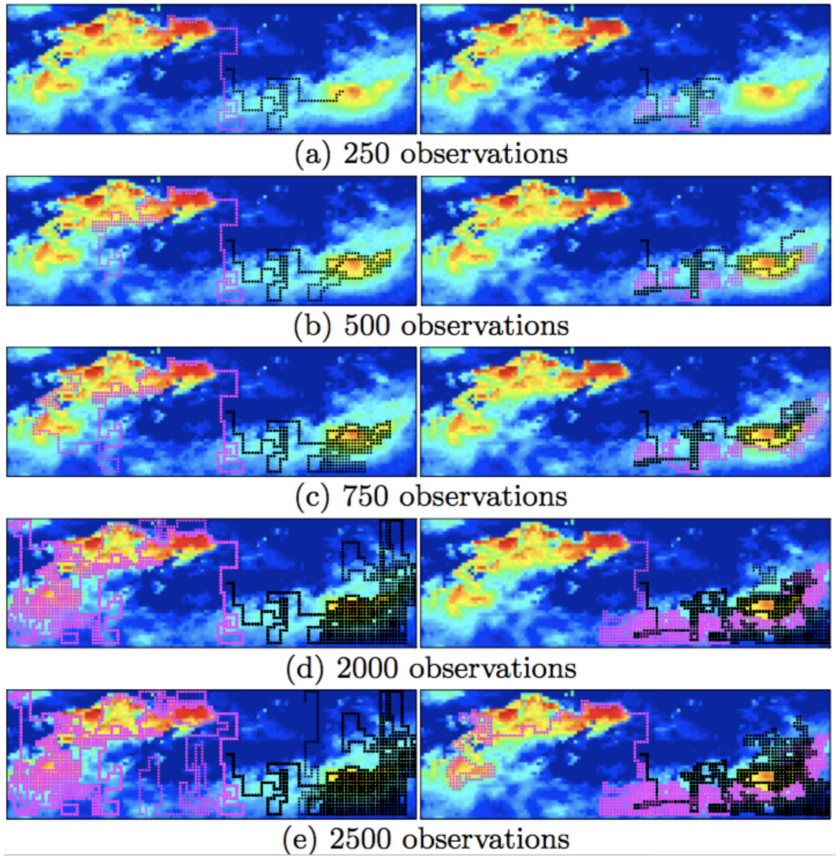
+



=



Exploration Problems



Exploration problems: agent(s) physically in some part of the state space

- e.g. find hotspot using an agent or agents with local temperature sensors
- Sensible to expand states easily accessible to agent (i.e. local states)
  - Local search algorithms apply (e.g., hill-climbing)