Informed/Heuristic search and Exploration

BBM 405 – Fundamentals of Artificial Intelligence

Pinar Duygulu

Hacettepe University

Slides are mostly adapted from AIMA, MIT Open Courseware and Svetlana Lazebnik (UIUC)

Review: Tree search

- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node to expand according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the frontier
- To handle repeated states:
  - Keep an **explored set**; add each node to the explored set every time you expand it
  - Every time you add a node to the frontier, check whether it already exists in the frontier with a higher path cost, and if yes, replace that node with the new one

Review: Uninformed search strategies

- A **search strategy** is defined by picking the order of node expansion
- **Uninformed** search strategies use only the information available in the problem definition
  - Breadth-first search
  - Depth-first search
  - Iterative deepening search
  - Uniform-cost search

Informed search strategies

- Idea: give the algorithm “hints” about the desirability of different states
  – Use an ***evaluation function*** to rank nodes and select the most promising one for expansion

- Greedy best-first search
- A* search

Outline

Informed search strategies use problem specific knowledge beyond the definition of the problem itself

- Best-first search
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Best-first search

- Idea: use an evaluation function *f(n)* to select the node for expansion
  - estimate of "desirability"

  → Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- A key component in best-first algorithms is a heuristic function, *h(n)*, which is the estimated cost of the cheapest path from n to a goal node

Best-first search

Best-first:

Pick "best" (measured by heuristic value of state) element of Q

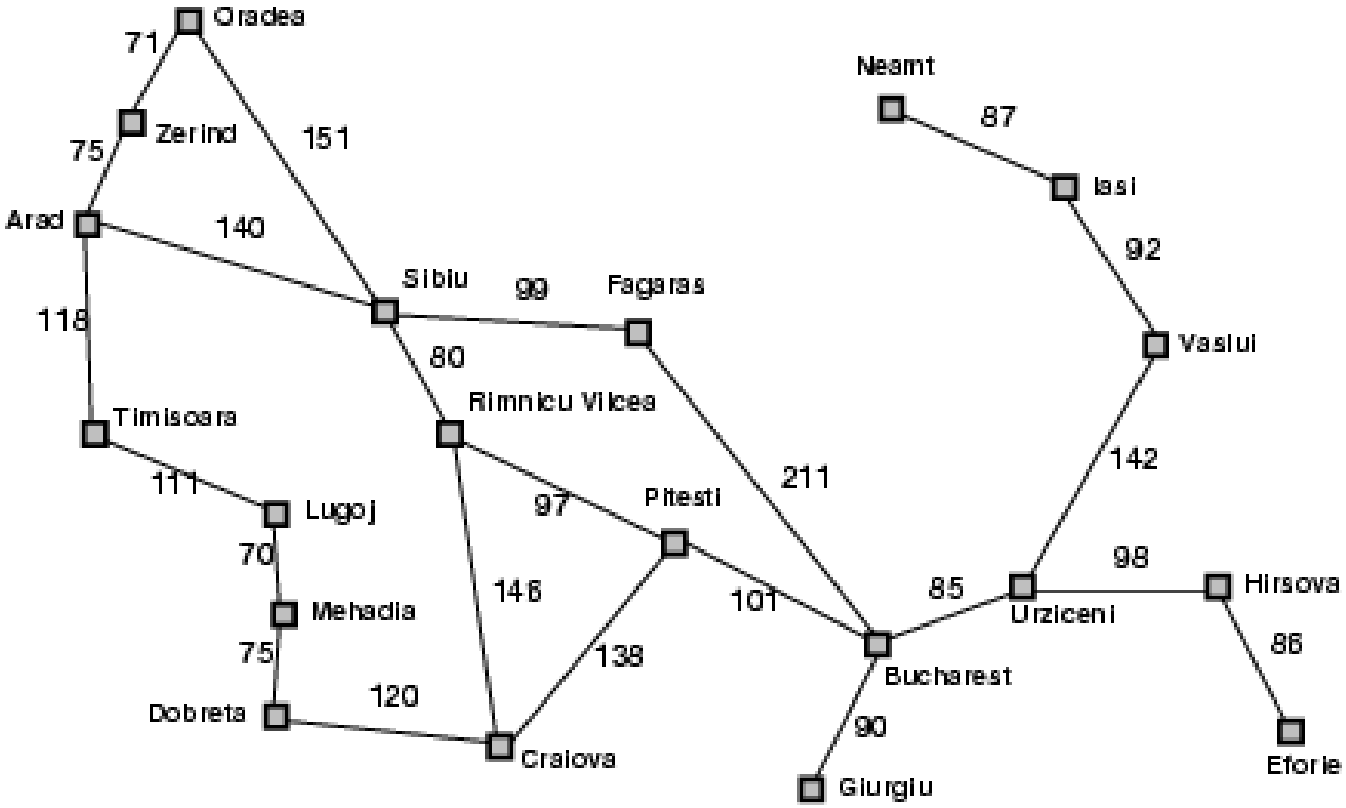
Add path extensions anywhere in Q (it may be more efficient tc keep the Q ordered in some way so as to make it easier to find the "best" element).

There are many possible approaches to finding the best node in Q.

- **Scanning Q to find lowest value**
- **Sorting Q and picking the first element**
- **Keeping the Q sorted by doing "sorted" insertions**
- **Keeping Q as a priority queue**

Romania with step costs in km

e.g. For Romania, cost of the cheapest path from Arad to Bucharest can be estimated via the straight line distance



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

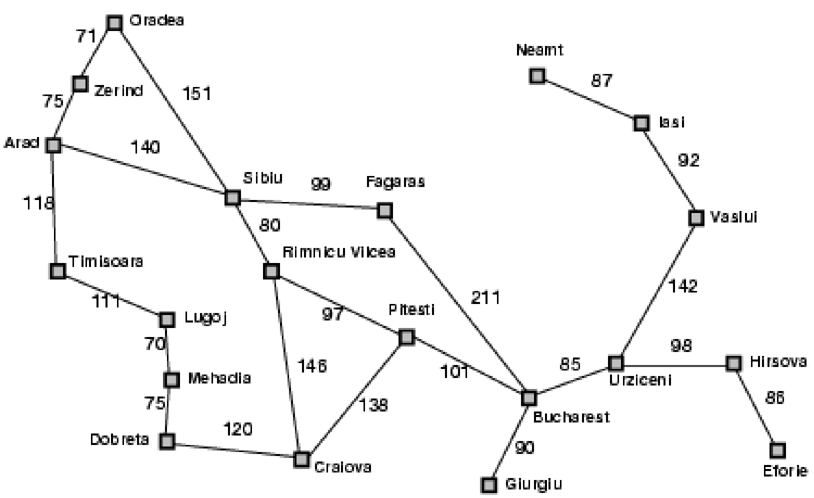
Greedy best-first search

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

- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- Note that, h_{SLD} cannot be computed from the problem description itself. It takes a certain amount of experience to know that it is correlated with actual road distances, and therefore it is a useful heuristic

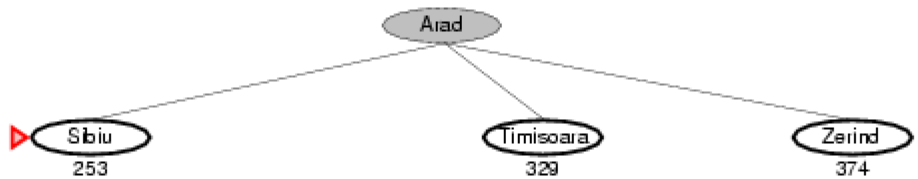
Greedy best-first search example

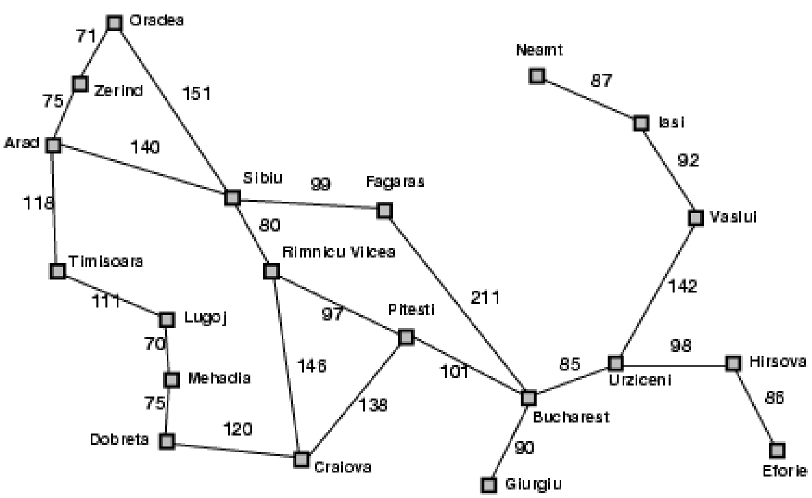


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

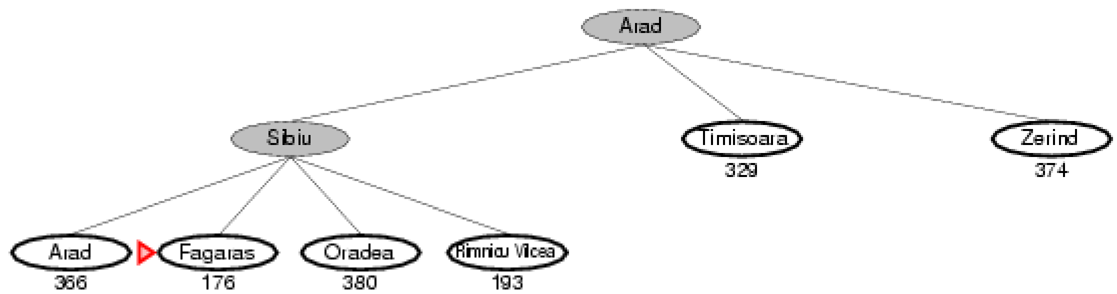


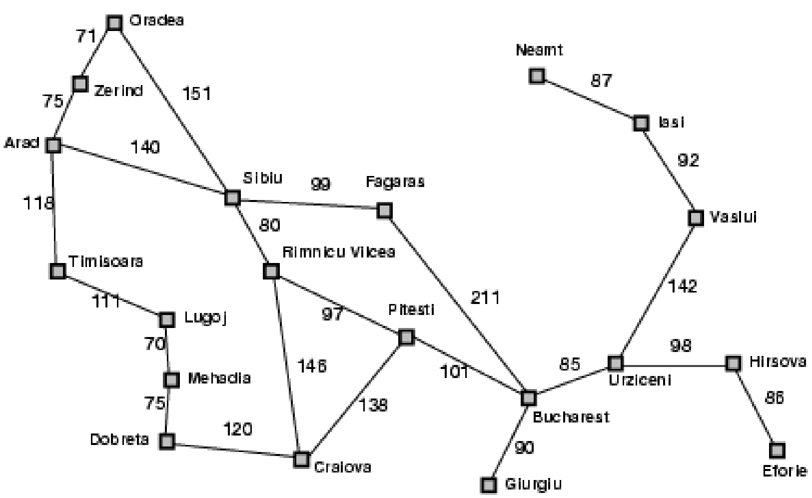


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

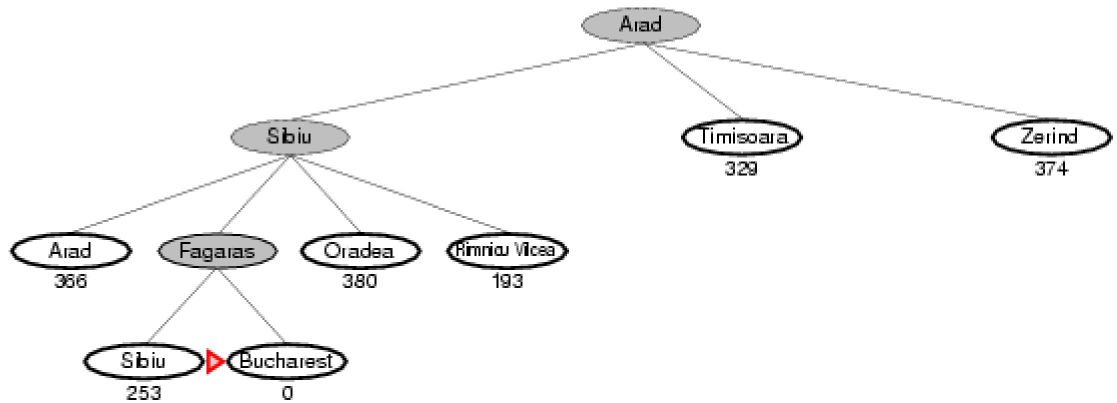
Greedy best-first search example

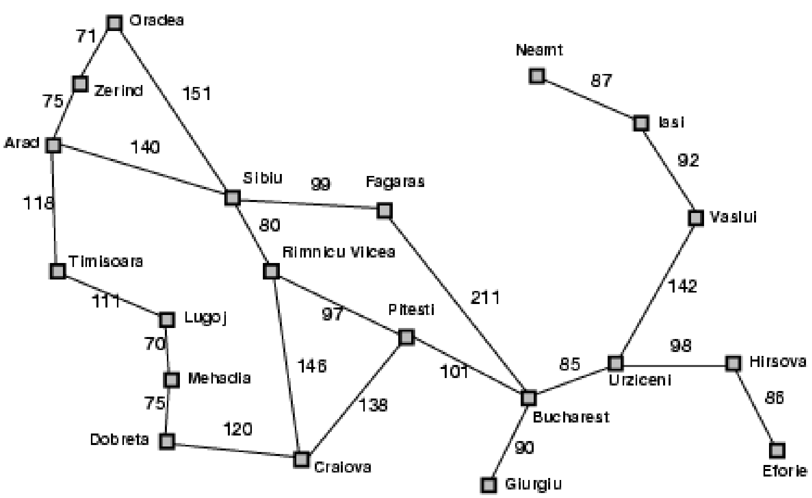




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example





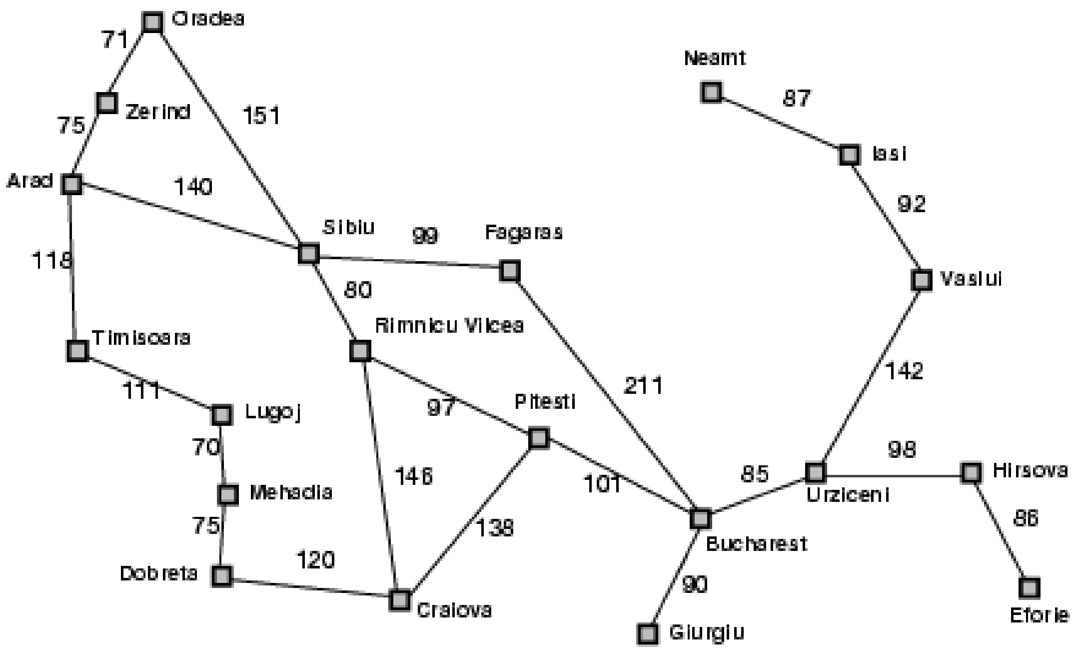
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

Problems:

Path through Faragas is not the optimal

In getting Iasi to Faragas, it will expand Neamt first but it is a dead end



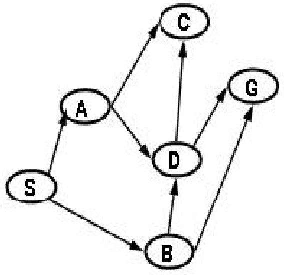
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search – Another example

Pick "best" (by heuristic value) element of Q; Add path extensions anywhere in Q

	Q	Visited
1	(10 S)	S
2		
3		
4		
5		



Heuristic Values

A=2 C=1 S=10

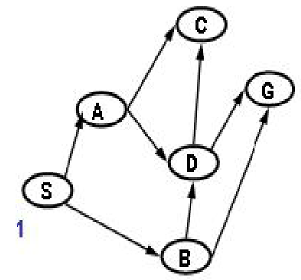
B=3 D=4 G=0

Added paths in blue; heuristic value of node's state is in front.

We show the paths in reversed order; the node's state is the first entry.

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3		
4		
5		



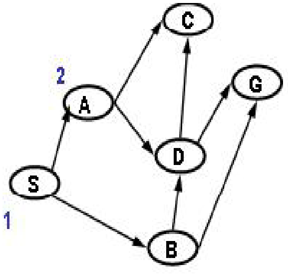
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4		
5		



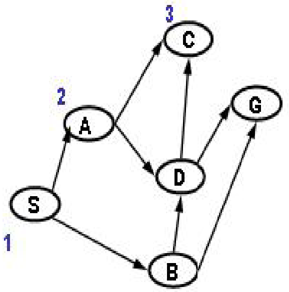
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5		



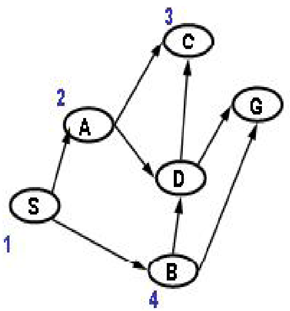
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S)	G,C,D,B,A,S



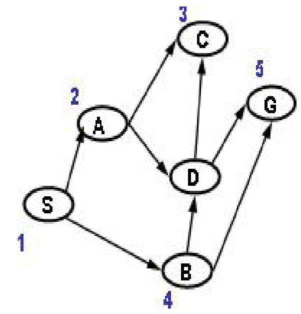
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S)	G,C,D,B,A,S



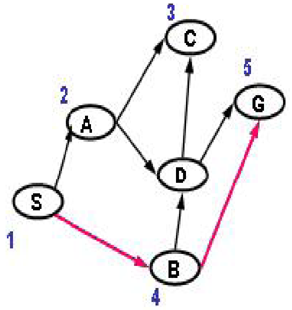
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Greedy best-first search – Another example

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	$\boxed{\text{(0 G B S)}}$ (4 D A S)	G,C,D,B,A,S



Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

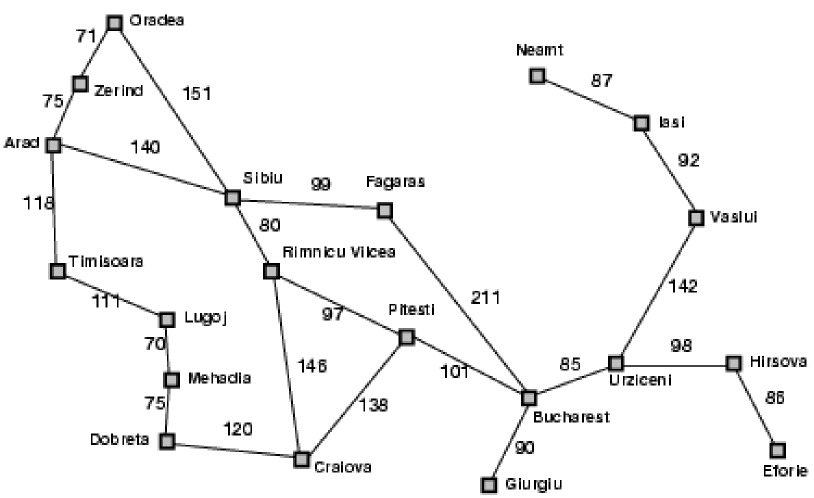
Properties of greedy best-first search

- Complete? No – can get stuck in loops,
  - e.g., Iasi → Neamt → Iasi → Neamt →
- 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
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

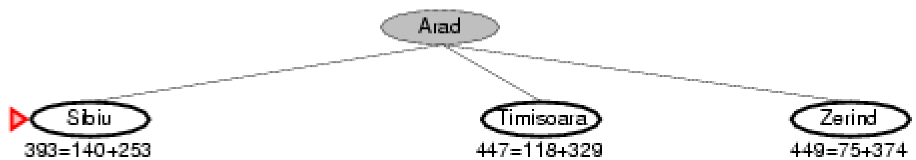
A* search example

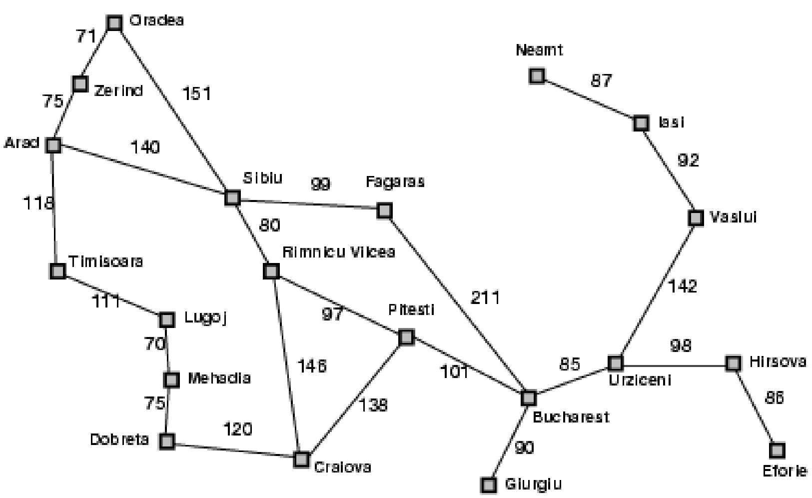


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

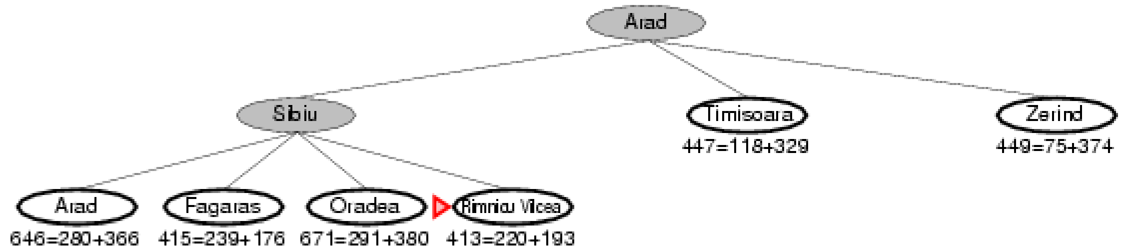
A* search example

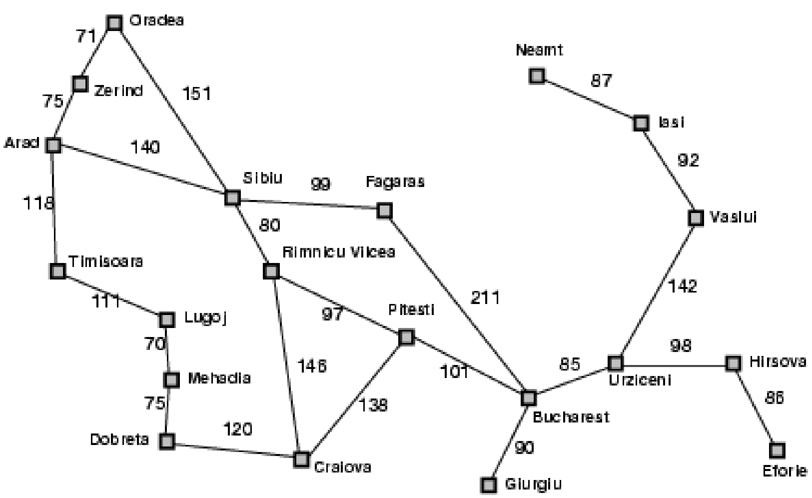




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

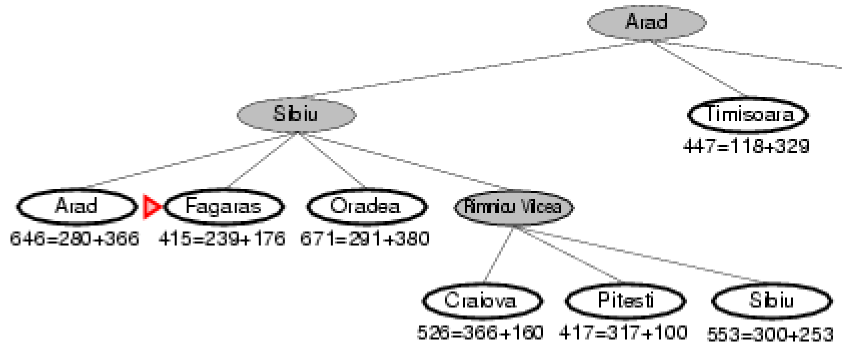


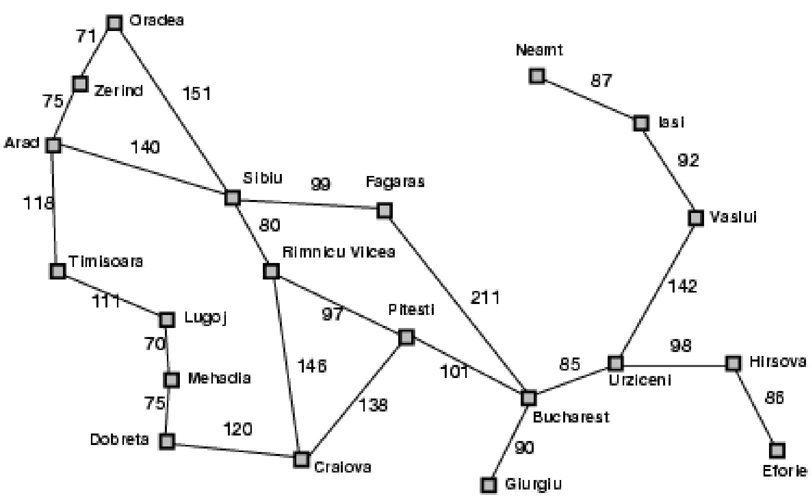


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

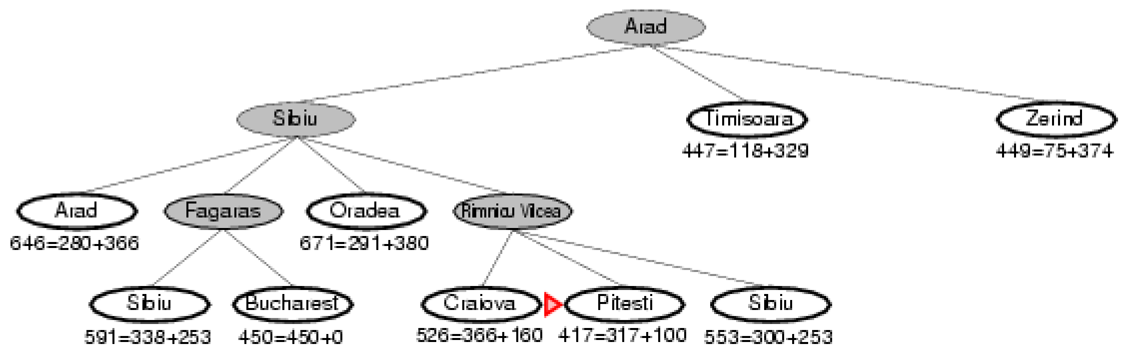


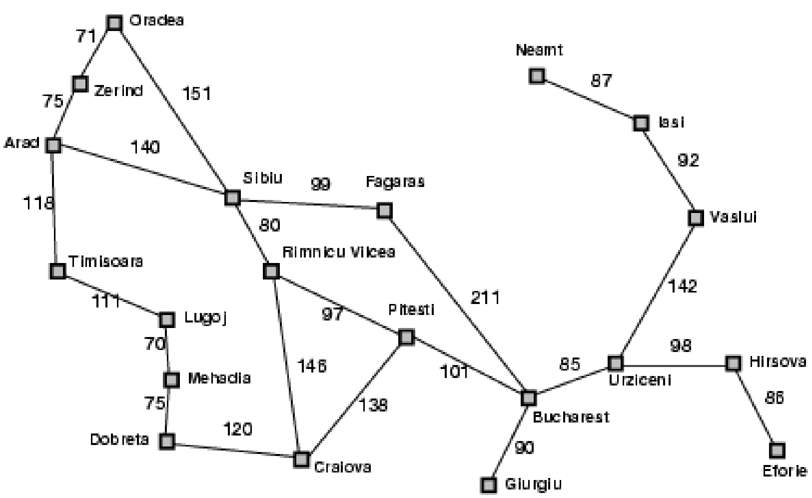


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

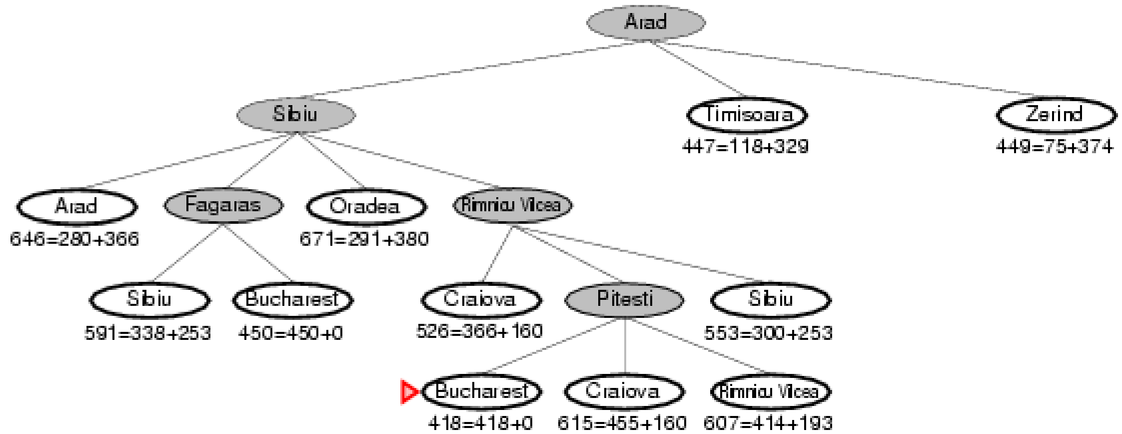
A* search example

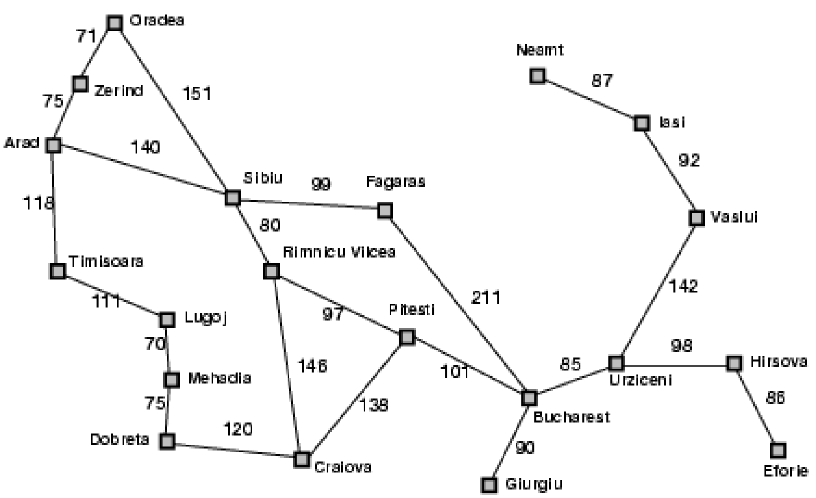




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example





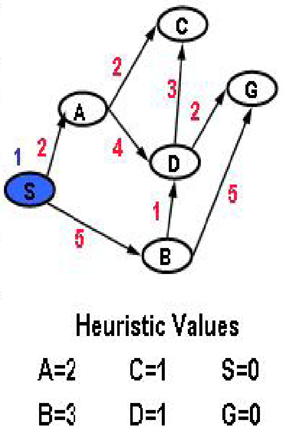
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search – Another example

Pick best (by path length+heuristic) element of Q; Add path extensions anywhere in Q

	Q
1	(0 S)

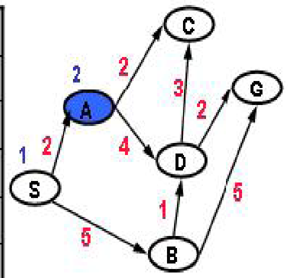


Added paths in blue; underlined paths are chosen for extension.

We show the paths in reversed order; the node's state is the first entry.

A* search – Another example

	Q
1	(0 S)
2	(4 A S) (8 B S)



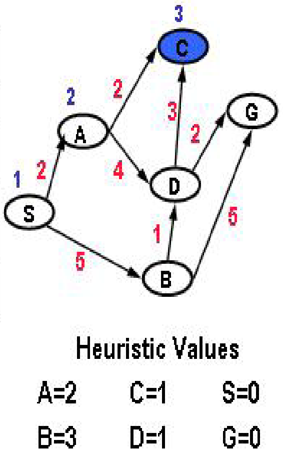
Heuristic Values

A=2 C=1 S=0

B=3 D=1 G=0

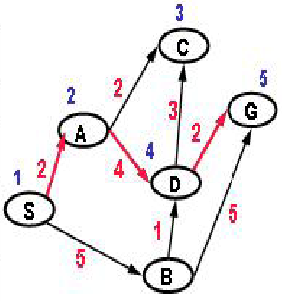
A* search – Another example

	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)



A* search – Another example

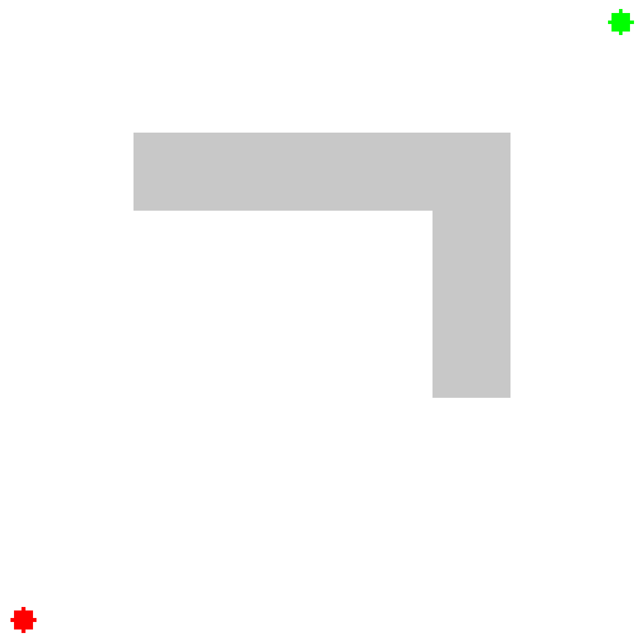
	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)
4	(7 D A S) (8 B S)
5	(8 G D A S) (10 C D A S) (8 B S)



Heuristic Values

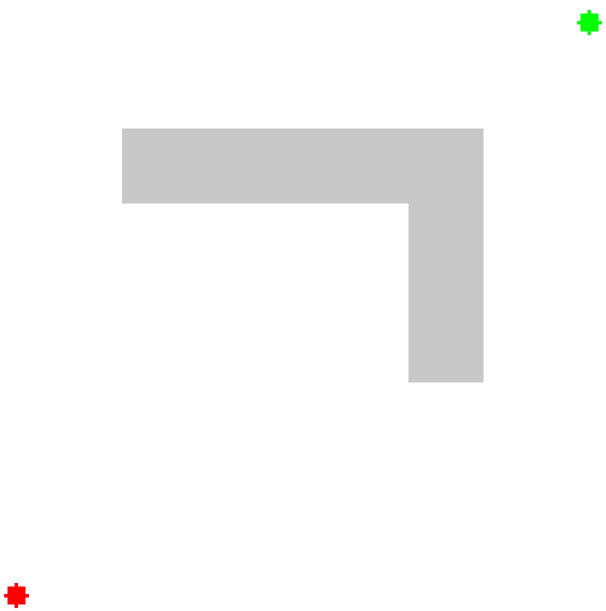
A=2	C=1	S=0
B=3	D=1	G=0

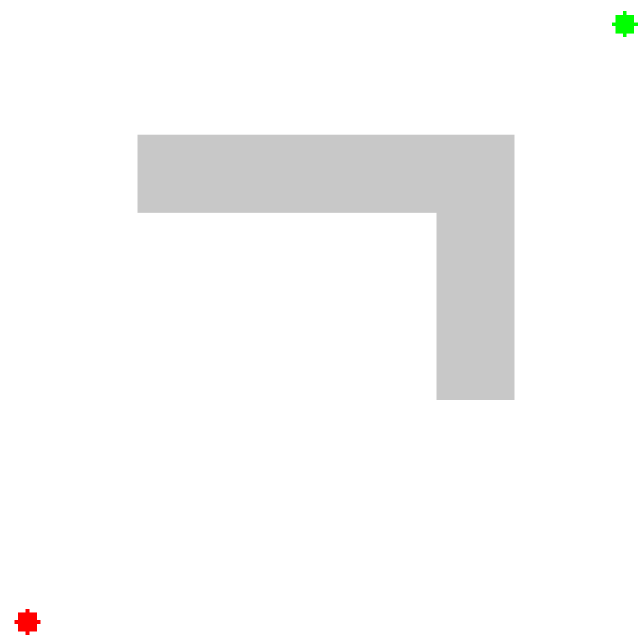
Another example



Wikipedia

Uniform cost search vs. A* search





Wikipedia

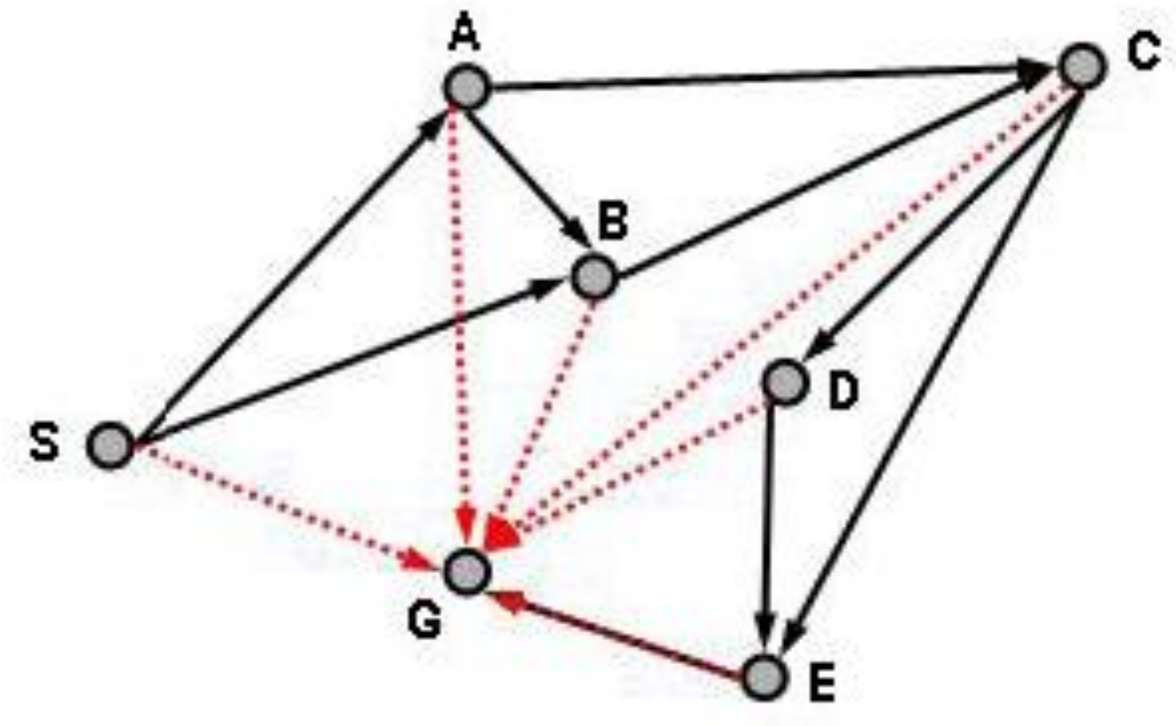
Classes of search

Class	Name	Operation
Any Path Uninformed	Depth-First Breadth-First	Systematic exploration of whole tree until a goal node is found.
Any Path Informed	Best-First	Uses heuristic measure of goodness of a node, e.g. estimated distance to goal.
Optimal Uninformed	Uniform-Cost	Uses path "length" measure. Finds "shortest" path.
Optimal Informed	A*	Uses path "length" measure and heuristic Finds "shortest" path

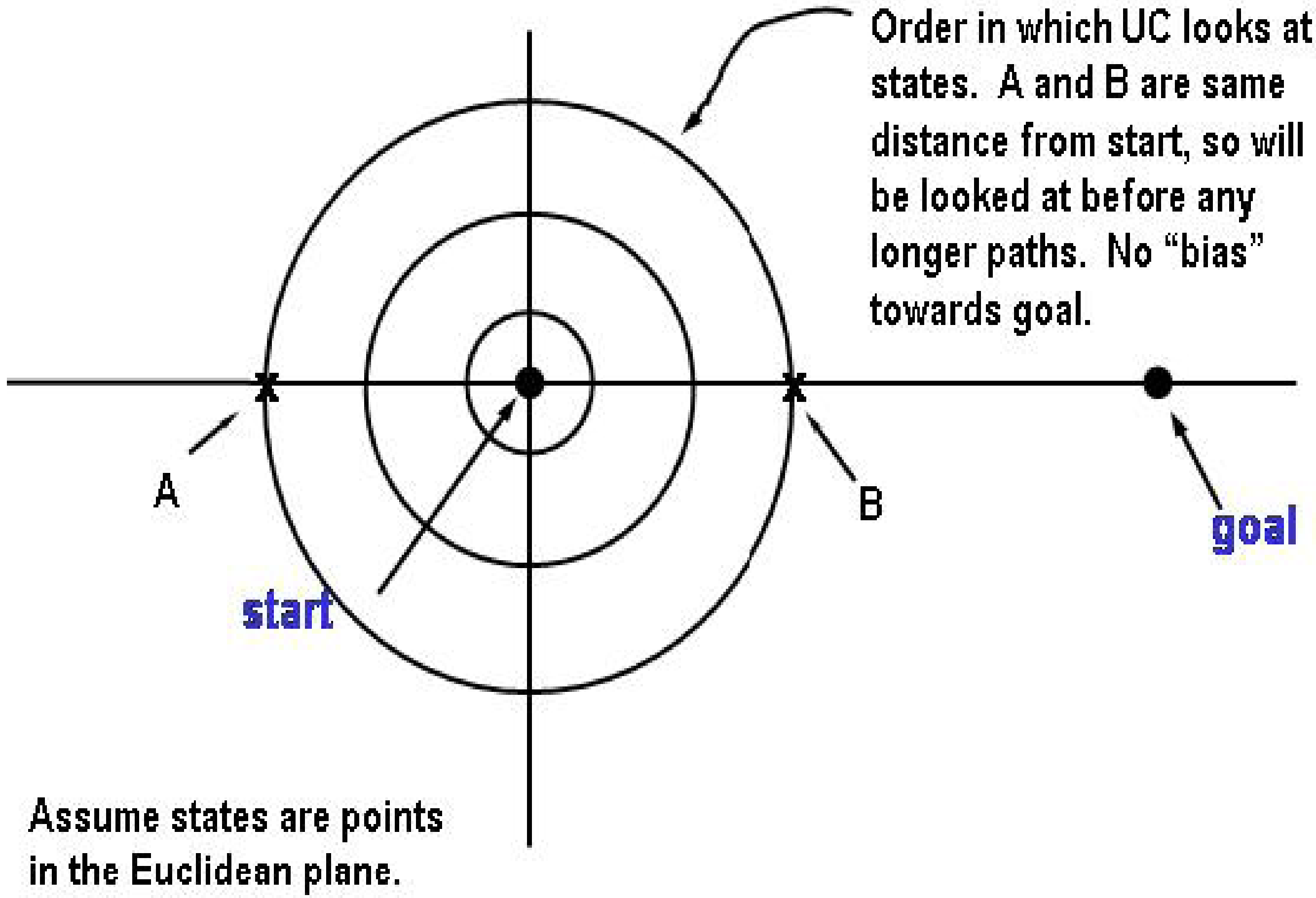
Uniform Cost (UC) versus A*

- UC is really trying to identify the shortest path to every state in the graph in order. It has no particular bias to finding a path to a goal early in the search.
- We can introduce such a bias by means of heuristic function h(N), which is an estimate (h) of the distance from a state n to a goal.
- Instead of enumerating paths in order of just length (g), enumerate paths in terms of f = estimated total path length = g + h.
- An estimate that always underestimates the real path length to the goal is called admissible. For example, an estimate of 0 is admissible (but useless). Straight line distance is admissible estimate for path length in Euclidean space.
- Use of an admissible estimate guarantees that UC will still find the shortest path.
- UC with an admissible estimate is known as A* (pronounced "A star") search.

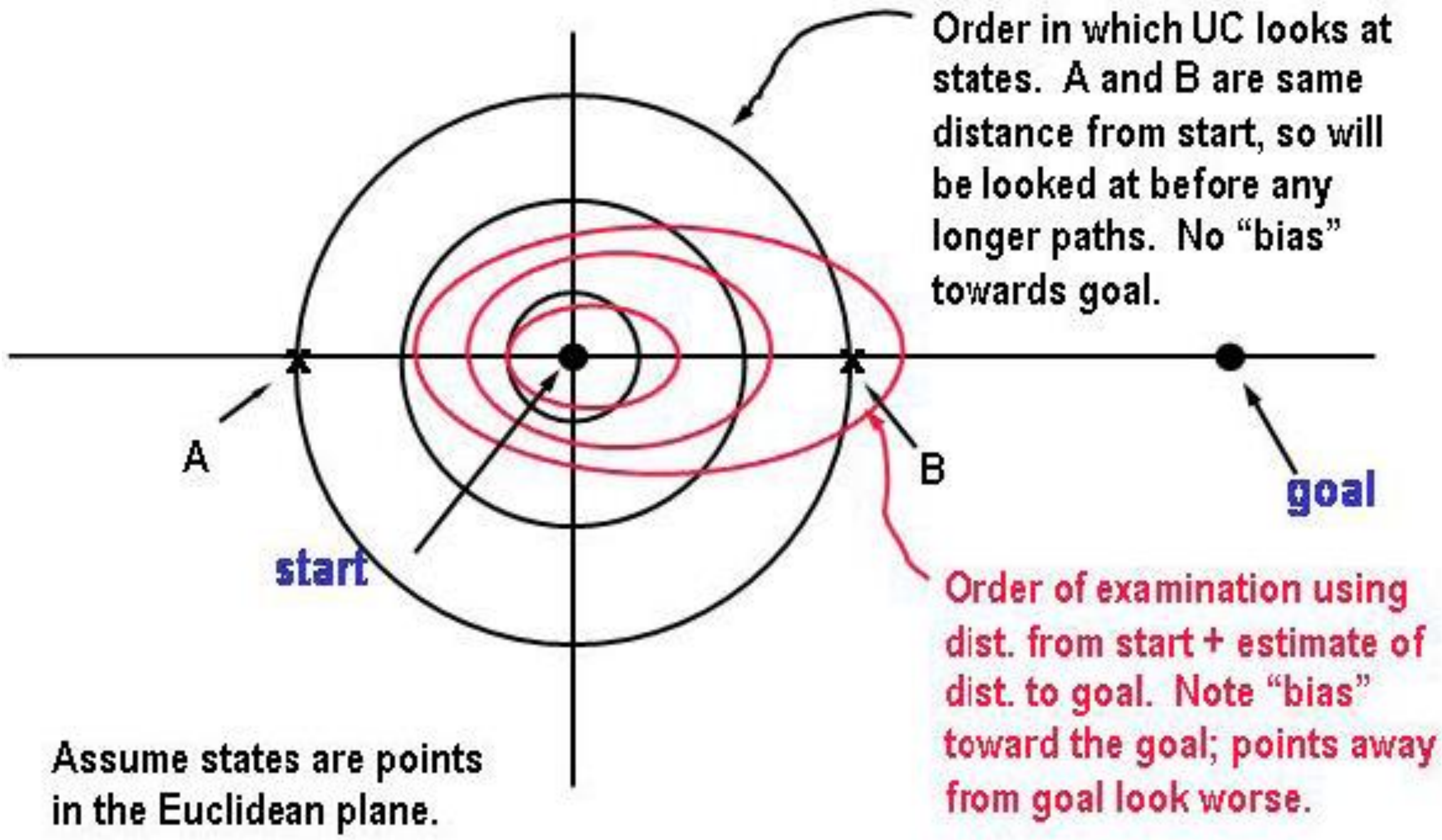
Straight line estimate



Why use estimate of goal distance

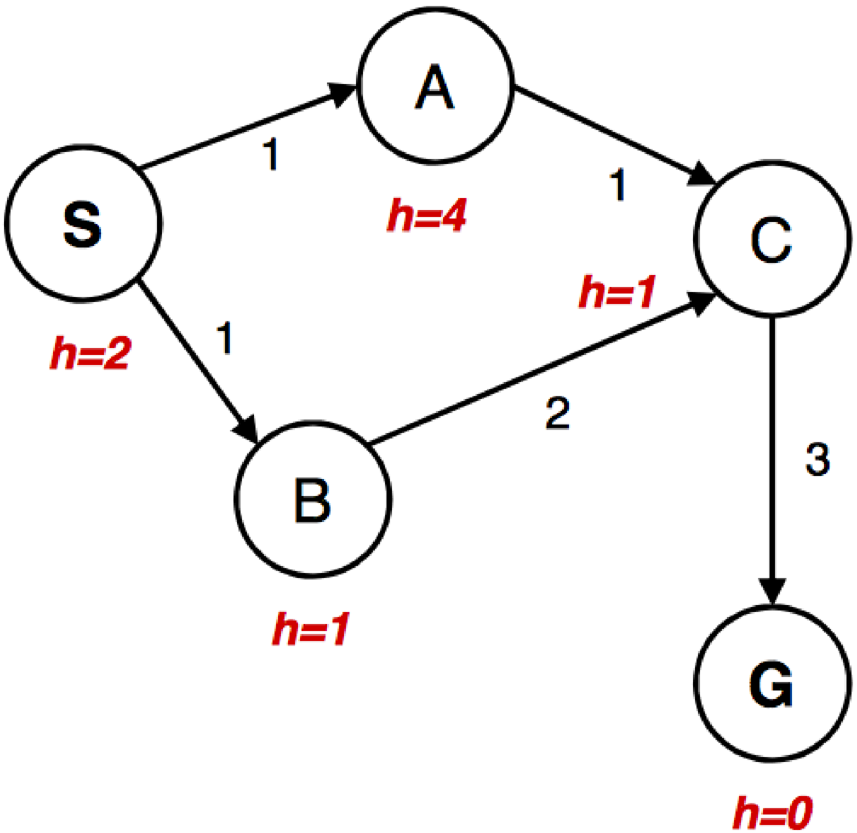


Why use estimate of goal distance

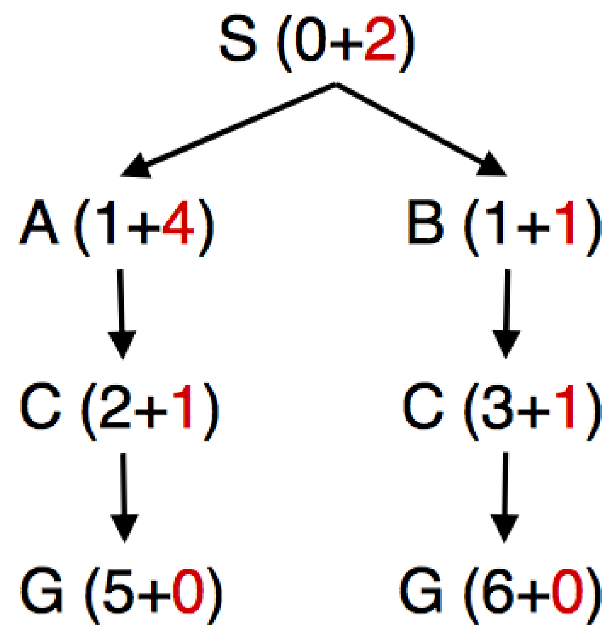


A* gone wrong?

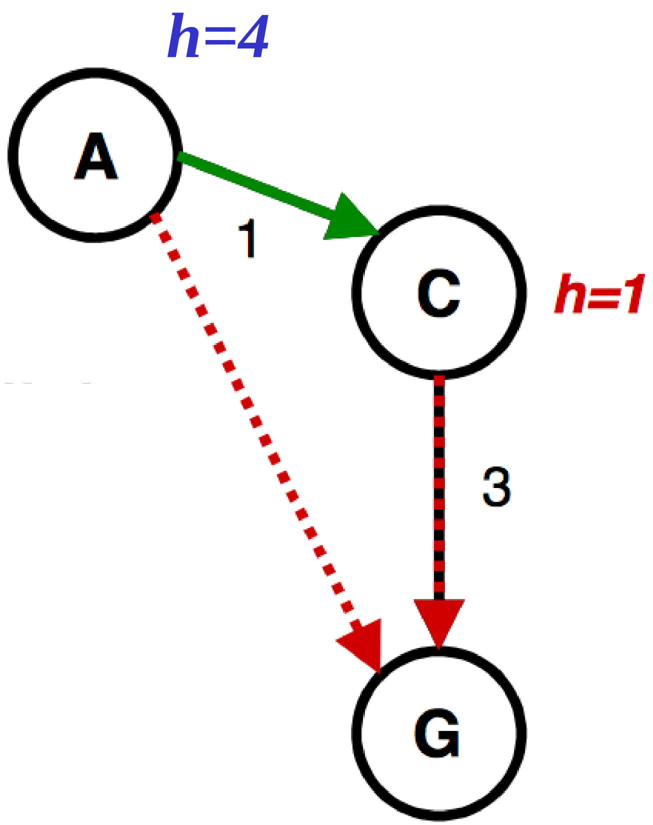
State space graph



Search tree



Consistency of heuristics



- Consistency: Stronger than admissibility
- Definition:

  cost(A to C) + h(C) ≥ h(A)

  cost(A to C) ≥ h(A) - h(C)

  real cost ≥ cost implied by heuristic

- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal

Not all heuristics are addmissible

Given the link lengths in the figure, is the table of heuristic values that we used in our earlier best-first example an admissible heuristic?

No!

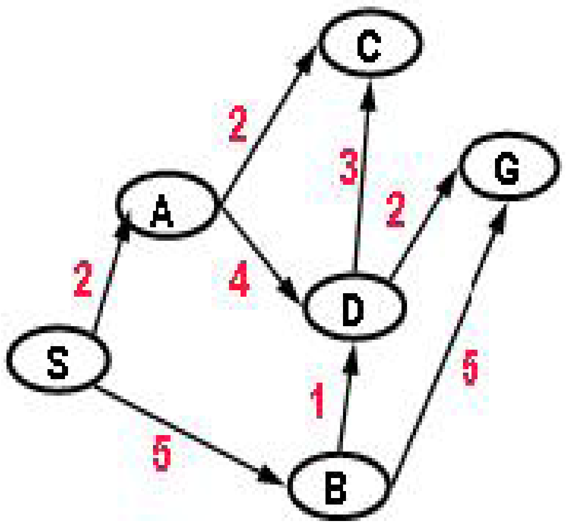
A is ok

B is ok

C is ok

D is too big, needs to be <= 2

S is too big, can always use 0 for start



Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

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 – thinks that the cost of solving the problem is less than it actually is
- Consequence: f(n) never over estimates the the true cost of a solution through n since g(n) is the exact cost to reach n
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance) since the shortest path between any two points is a straight line

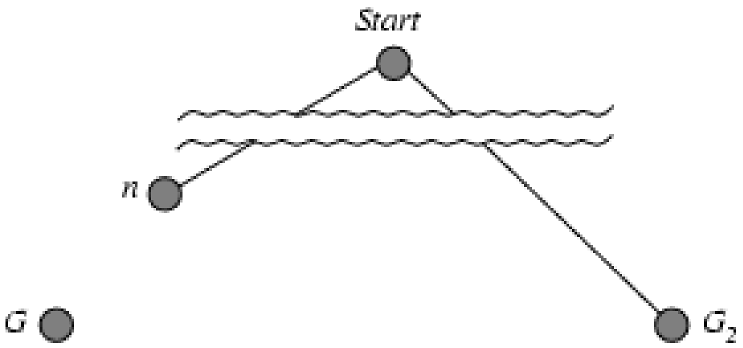
- Theorem: If *h(n)* is admissible, A^* using `TREE-SEARCH` is optimal

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let the cost of the optimal solution to goal *G is C**

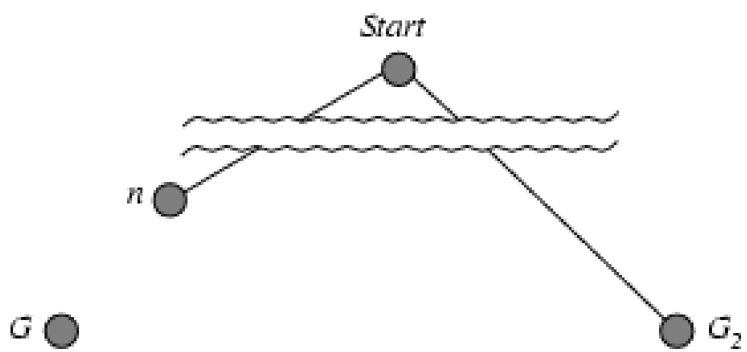
  f = g + h

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



Optimality of A* (proof)

- Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G (e.g. Pitesti)*.

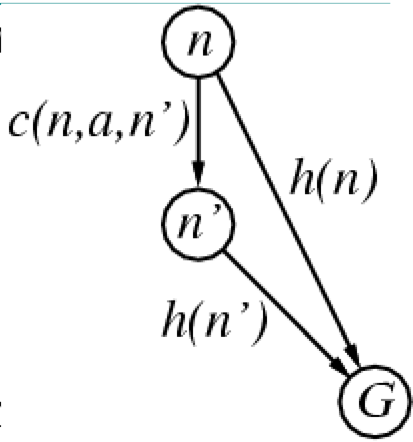


- If h(n) does not overestimate the cost of completing the solution path, then
- $f(n) = g(n) + h(n) \leq C^*$
- $f(n) \leq f(G)$
- $f(G_2) > f(G)$ from above
- Hence $f(G_2) > f(G) >= f(n)$, and A* will never select G_2 for expansion

Consistent heuristics

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

  $h(n) \leq c(n,a,n') + h(n')$
  n' = successor of n generated by action a



- The estimated cost of reaching the goal from n is no greater than th cost of getting to n' plus the estimated cost of reaching the goal from n'

- If *h* is consistent, we have

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

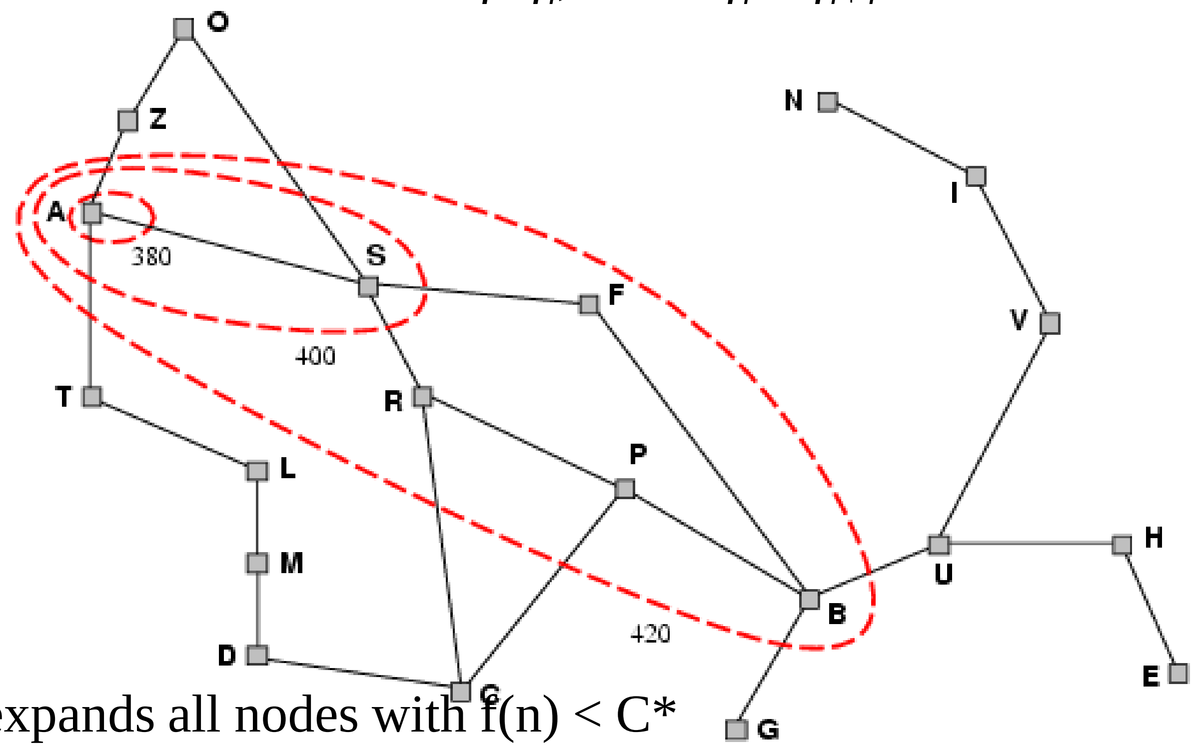
- if *h(n) is consistent then the values of f(n)* along any path are non-decreasing

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

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 $f_i < f_{i+1}$



- A* expands all nodes with f(n) < C*
- A* might then expand some of the nodes right on the goal contour (where f(n) = C*) before selecting a goal state
- A* expands no nodes with f(n) > C* (e.g. the subtree under Timisoara)

Properties of A*

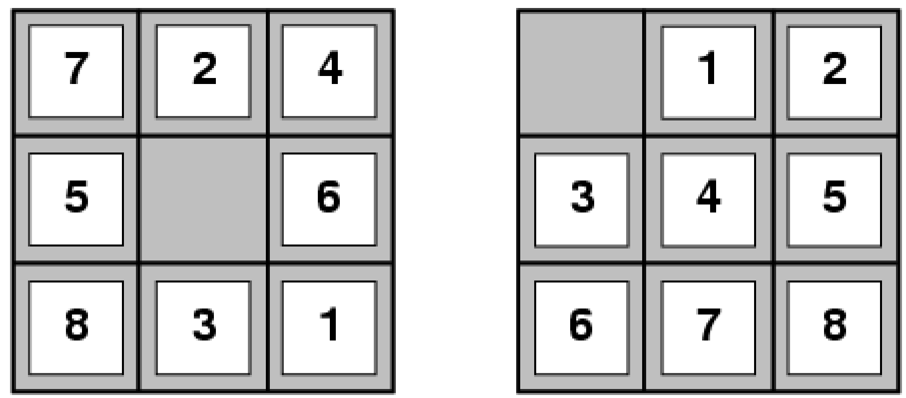
- Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)*
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

- Alternative:
  - Memory bounded heuristic search :
    - IDA*: adapt the idea of iterative deepening search, use cut-off as f-cost rather than the depth.
    - Recursive best-first search

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance – the sum of the distances of the tiles from their goal positions



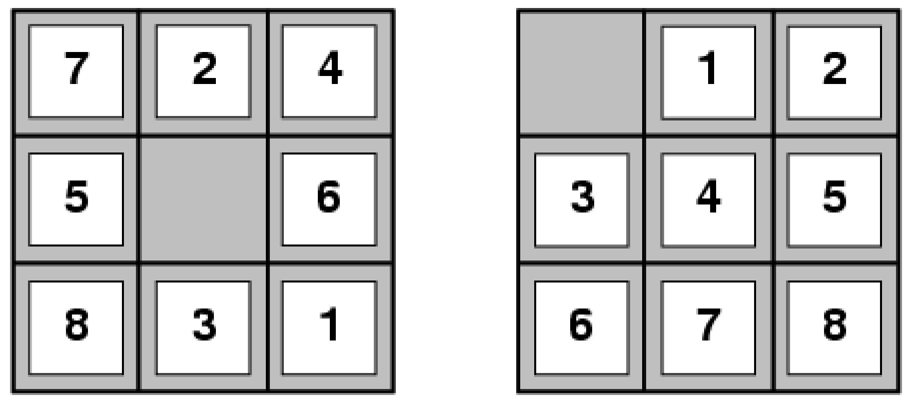
Start State   Goal State

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

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance



Start State Goal State

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

Quality of a heuristic

- Effective branching factor b*
- If N is the number of nodes generated by A*, and the solution depth is d, then
  - N+1 = 1 + b* + (b*)^2 + … + (b*)^d

- E.g. If A* finds a solution at depth 5 using 52 nodes, then b* = 1.92

- The average solution cost for randomly generated 8-puzzle instance is about 22 steps. The branching factor is 3 (when the tile is in middle it is 4, when in the corner it is 2, when it is along the edge it is 3)

- Typical search costs (average number of nodes expanded):
- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes, b* =1,42
  $A^*(h_2)$ = 73 nodes, b* = 1.24
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes, b* = 1.48
  $A^*(h_2)$ = 1,641 nodes, b* = 1.26

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search
- It is always better to use a heuristic function with higher values, provided it does not overestimate and that the computation time for the heuristic is not too large

Why?

Every node with f(n) < C* will be expanded

i.e. every node with h(n) < C* - g(n) will be expanded

Since h2 is at least as big as h1 for all nodes, every node that is expanded by h2, will be also expanded by h1, and h1 may also cause other nodes to be expanded

Inventing admissible heuristic functions

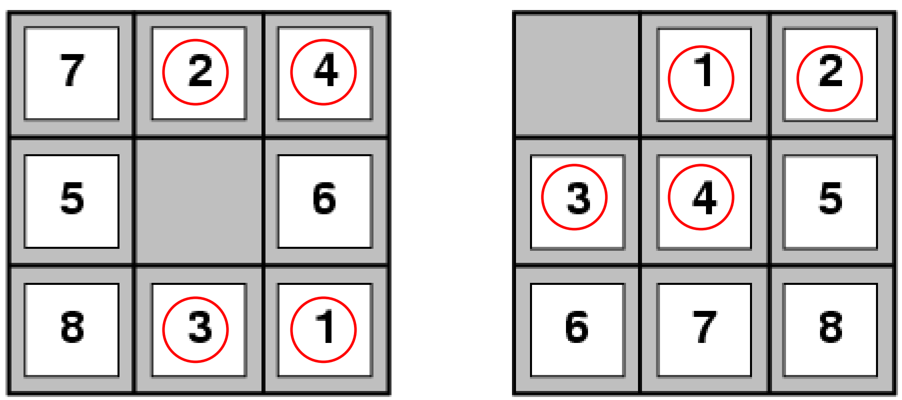
- h1 and h2 estimates perfectly accurate path length for simplified versions of 8-puzzle
- If a tile could move anywhere, then h1 would give the exact number of steps in the shortest solution.
- If a tile could move one square in any direction, even onto an occupied square, then h2 would give the exact number of steps in the shortest solution.

Relaxed problems

- 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
- The heuristic is admissible because the optimal solution in the original problem is also a solution in the relaxed problem and therefore must be at least as expensive as the optimal solution in the relaxed 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

Heuristics from subproblems

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions
- Can precompute and save the exact solution cost for every possible subproblem instance – *pattern database*



Start State Goal State

Inventing admissible heuristic functions

- If a problem definition is written down in a formal language, it is possible to construct relaxed problems automatically (ABSOLVER)
  - If 8-puzzle is described as
    - A tile can move from square A to square B if
      - A is horizontally or vertically adjacent to B and B is blank
  - A relaxed problem can be generated 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
  - h2 can be derived from (a) – h2 is the proper score if we move each tile into its destination
  - h1 can be derived from (c) – it is the proper score if tiles could move to their intended destination in one step
- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem

Combining heuristics

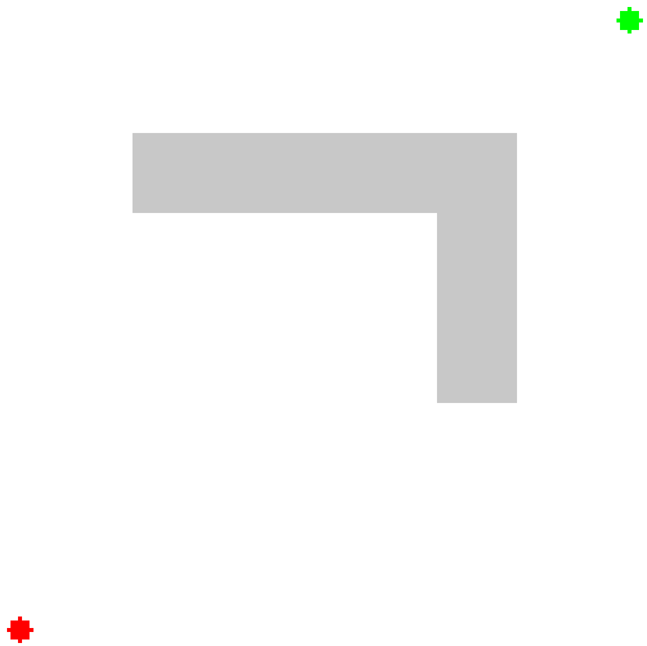
- Suppose we have a collection of admissible heuristics $h_1(n)$, $h_2(n)$, …, $h_m(n)$, but none of them dominates the others
- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

Weighted A* search

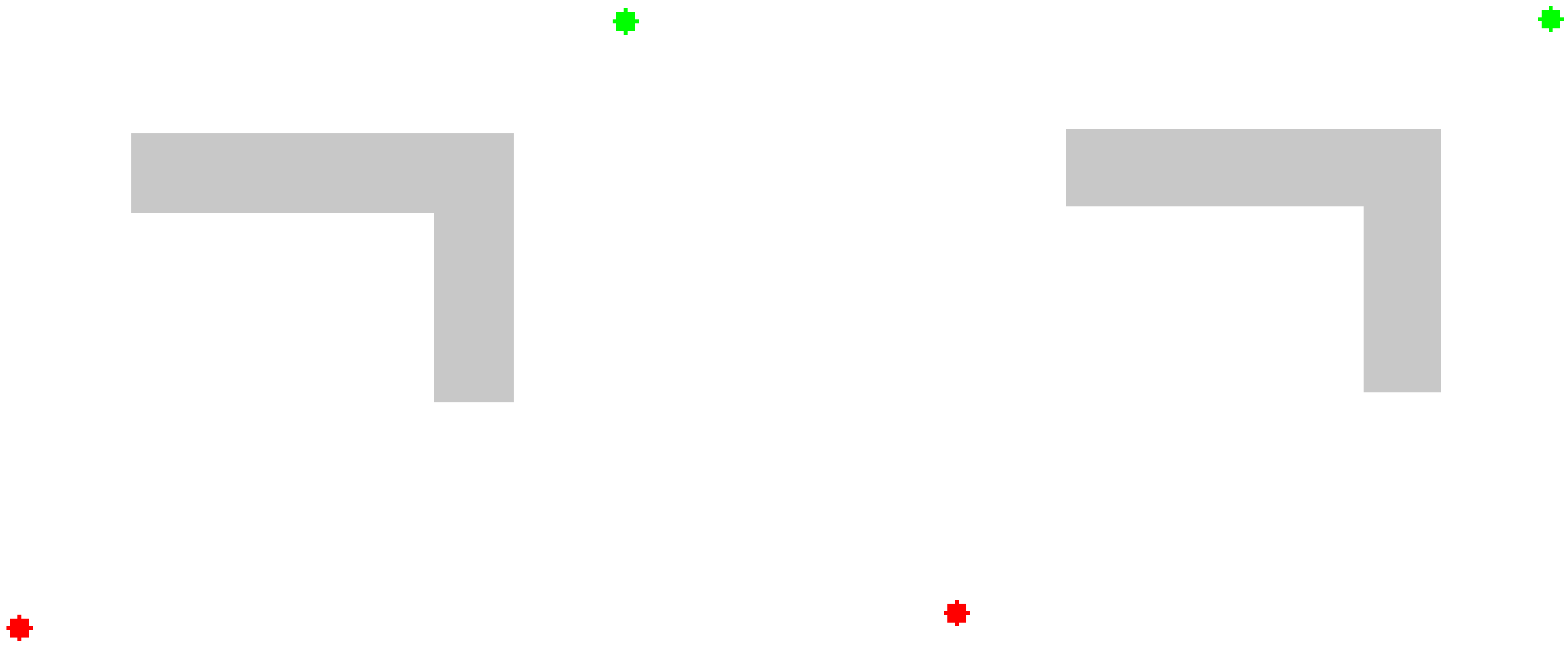
- **Idea:** speed up search at the expense of optimality
- Take an admissible heuristic, “inflate” it by a multiple $\alpha > 1$, and then perform A* search as usual
- Fewer nodes tend to get expanded, but the resulting solution may be suboptimal (its cost will be at most α times the cost of the optimal solution)

Example of weighted A* search



Wikipedia

Example of weighted A* search



Wikipedia

Local search algorithms

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

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

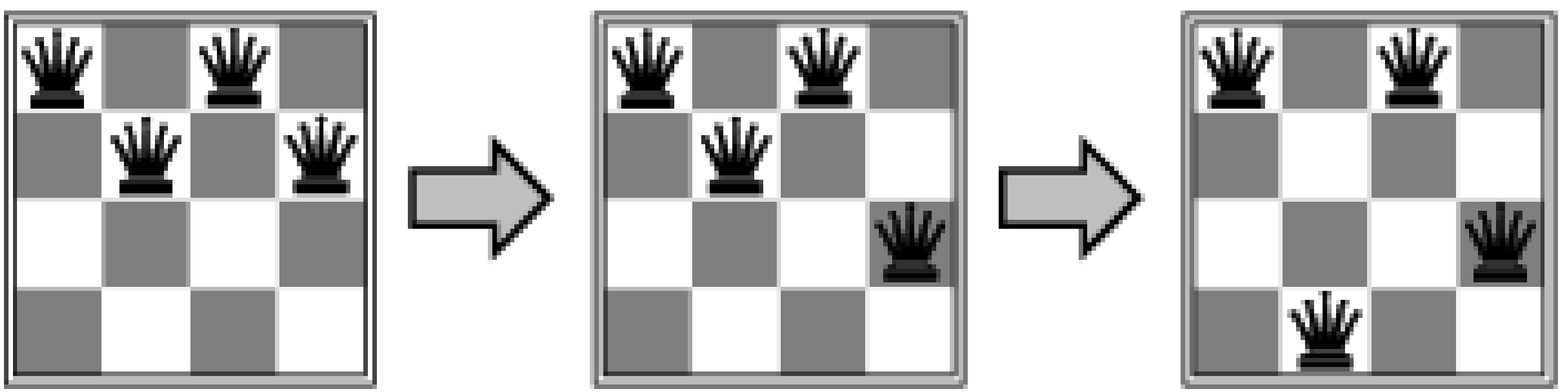
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

All search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

Example: *n*-queens

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



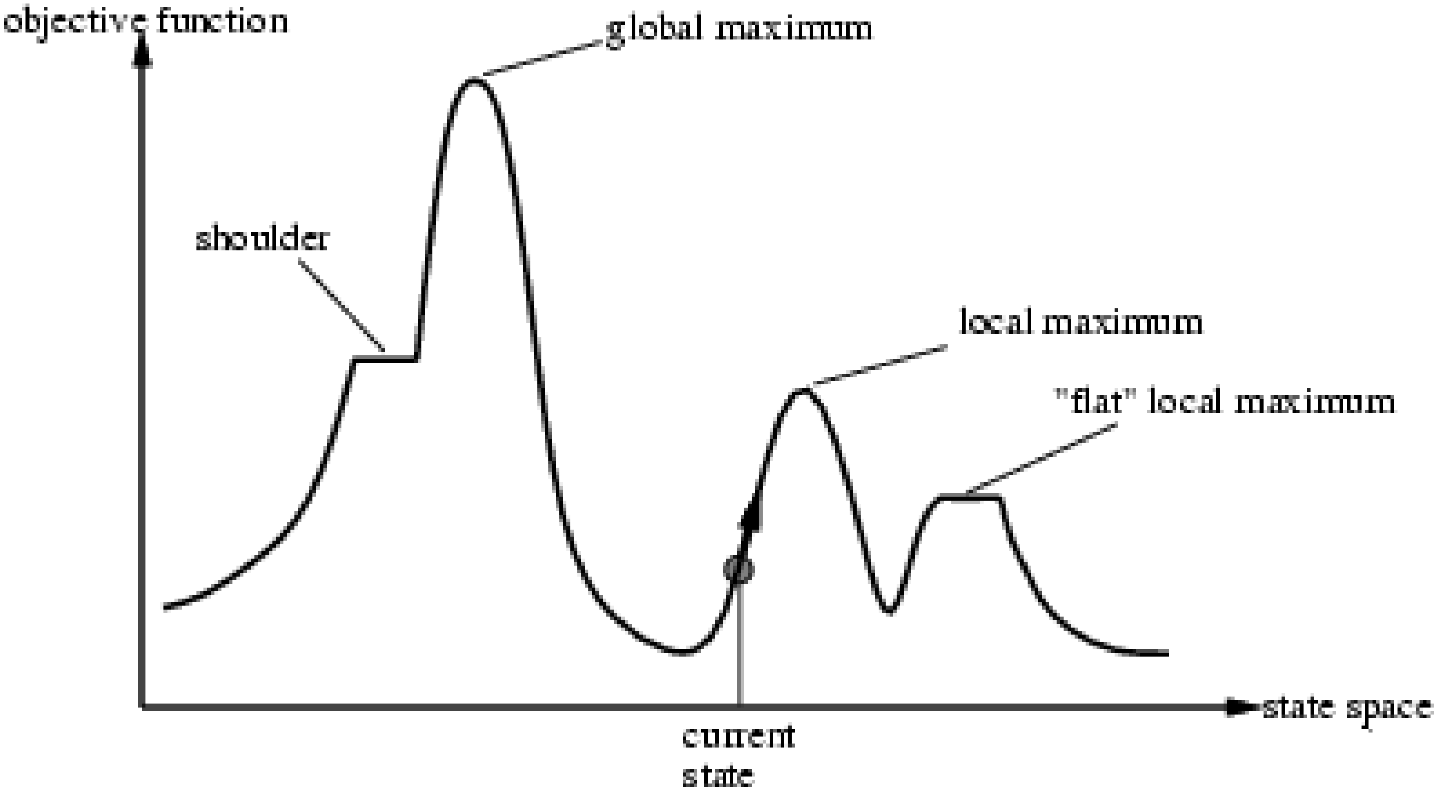
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

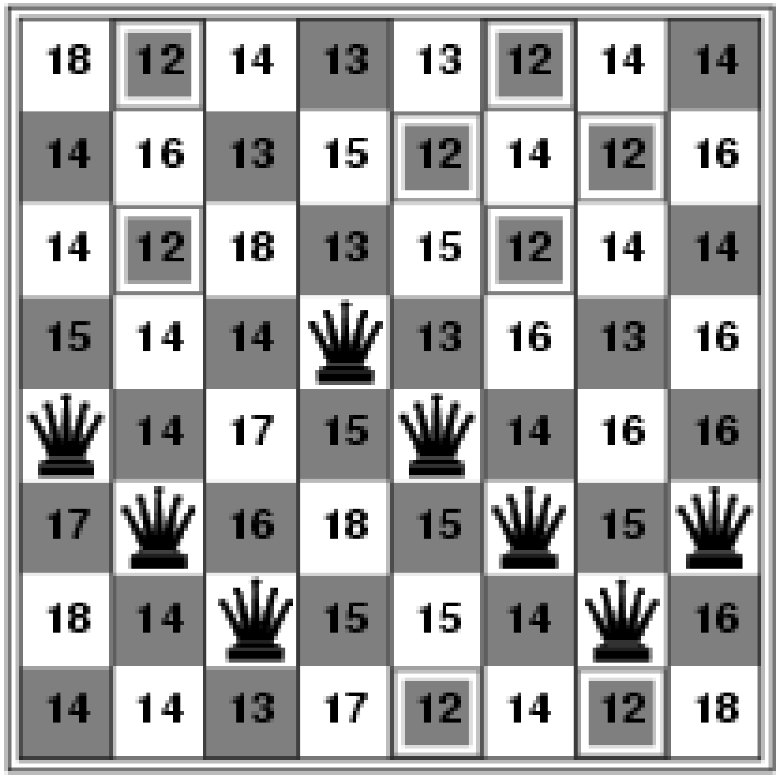
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    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
```

Hill-climbing search

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

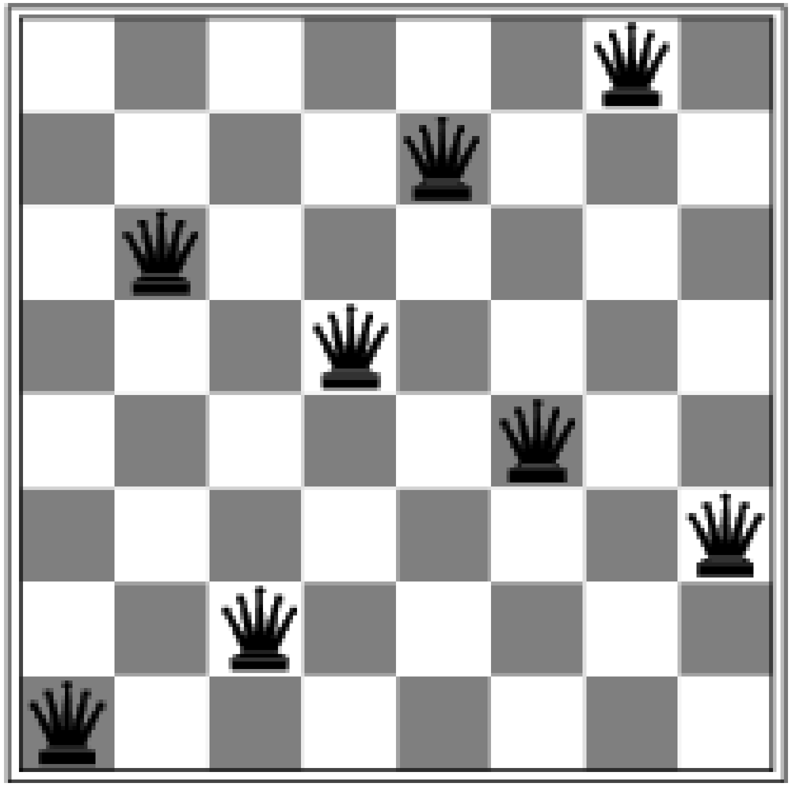


Hill-climbing search: 8-queens problem



- *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 problem



- A local minimum with $h = 1$

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" 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"
   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
       else current ← next only with probability e^(Δ E/T)
```

Properties of simulated annealing search

- One 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, airline scheduling, etc

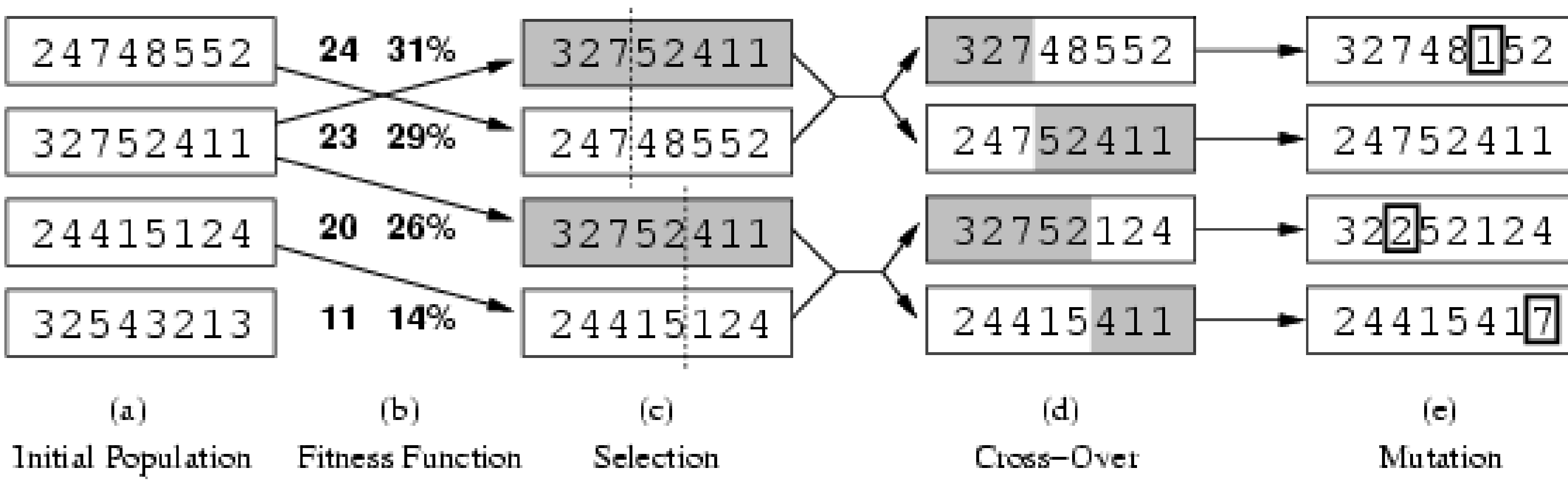
Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Genetic algorithms

- 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



Genetic algorithms