Heuristic Search[1]

Lecture 4

[1]The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello

Summary

Russell & Norvig Chapter 3, Sec. 5–6

- Best-first search
- A*
- Heuristics

Best-first search

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

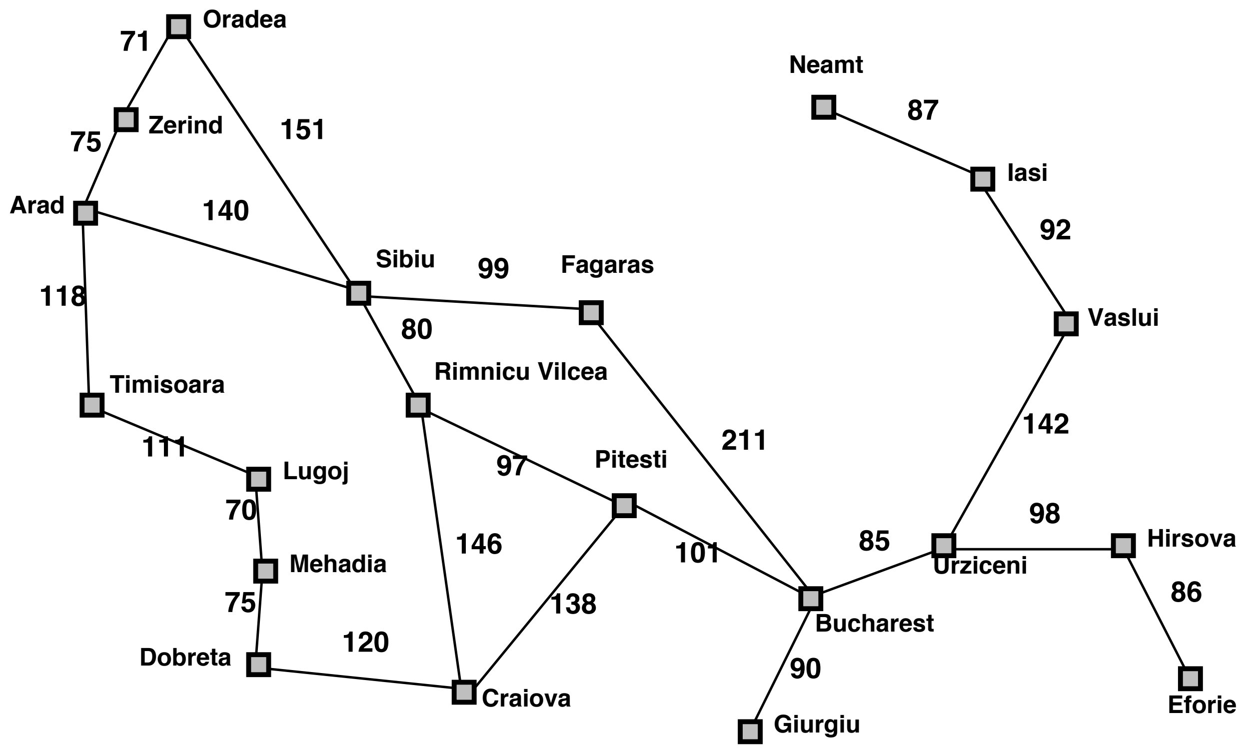
$\Rightarrow$ Expand most desirable unexpanded node

Implementation:
$frontier$ is a queue sorted in decreasing order of desirability

Special cases:
"best-first" search
A* search

best? "Best-first" vs greedy

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 "Best-first" search

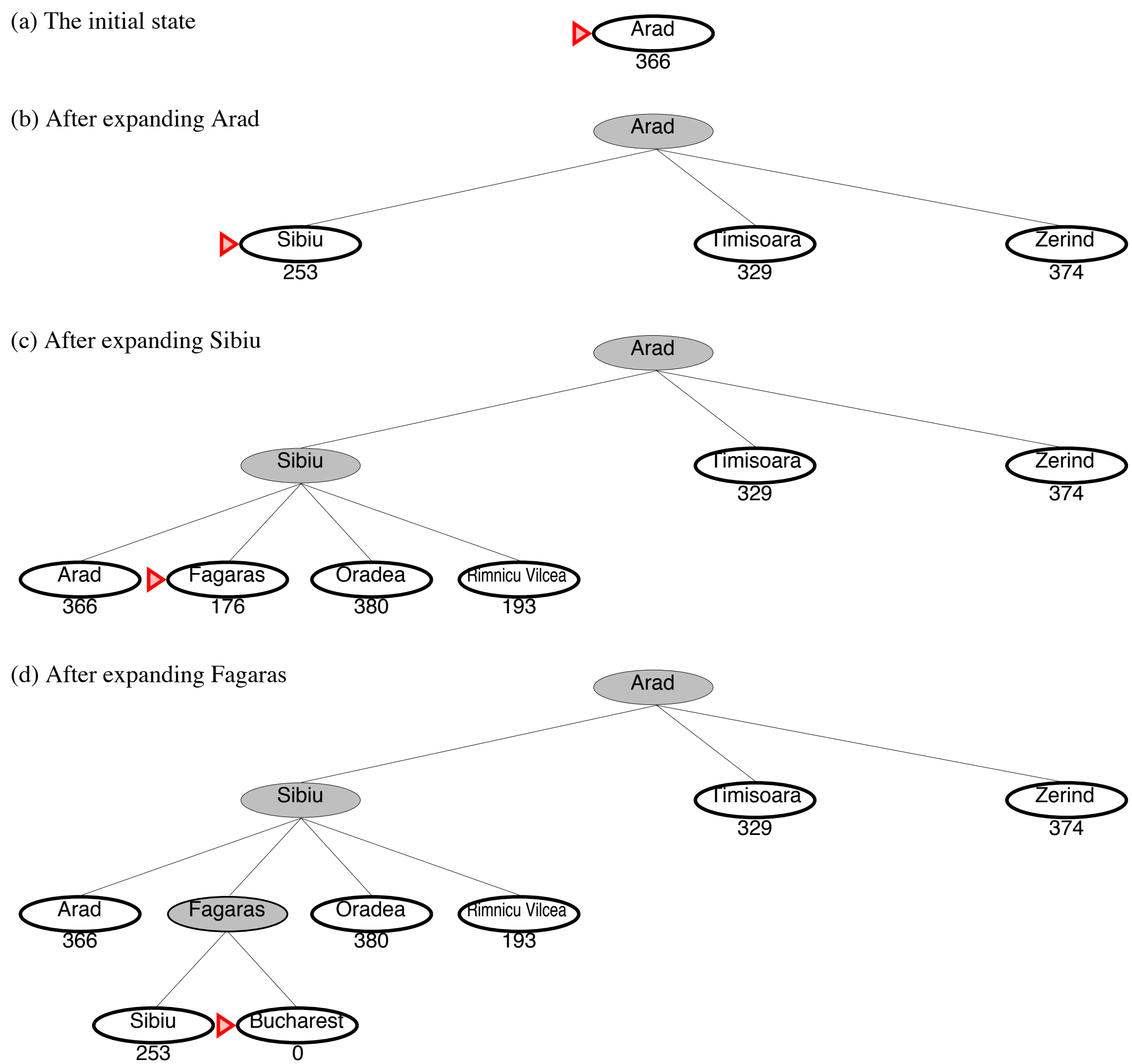
Evaluation function $h(n)$ (heuristic)
$= $ estimate of cost from n to the closest goal

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

"Best-first" search expands the node that *appears* to be closest to goal

A remark about heuristics: Usually a good h greatly improves the search (by expanding less nodes), but knowing ideal h amounts to knowing the solution ...

Greedy search example



Properties of “best-first” 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 $f(n) = g(n) + h(n)$

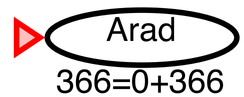
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

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.)

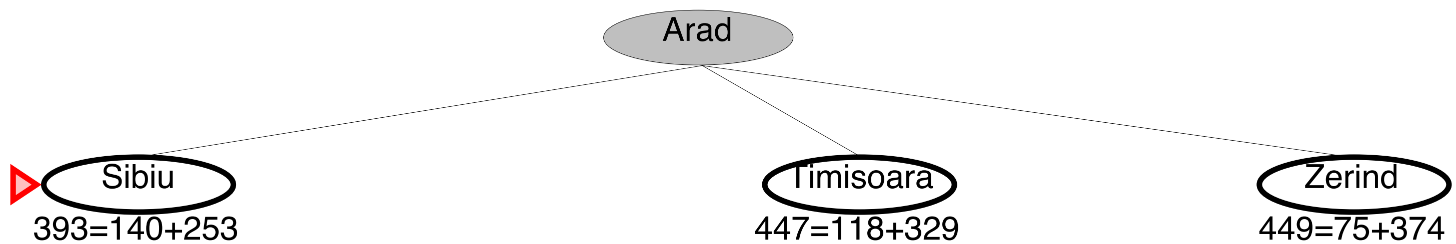
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Key property: A* search is optimal

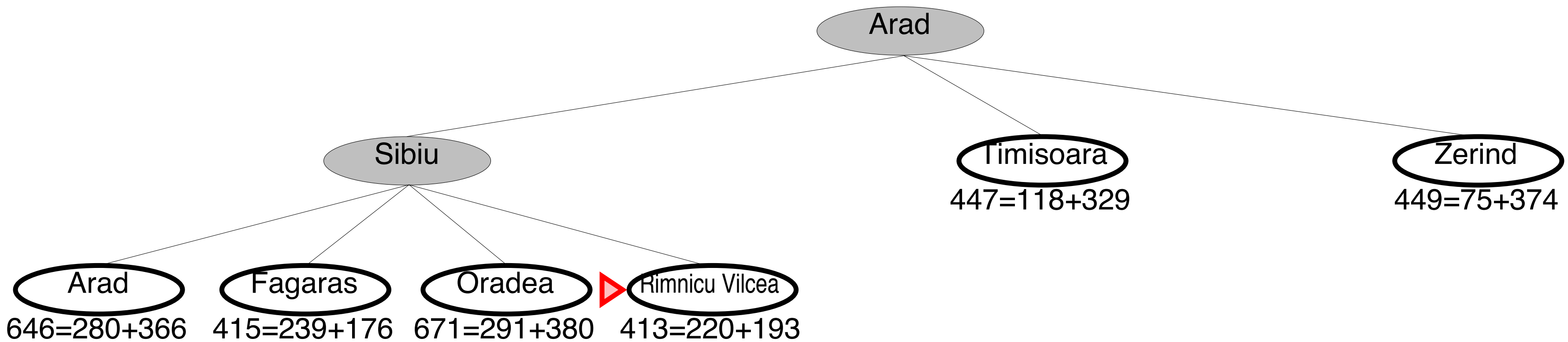
A* search example



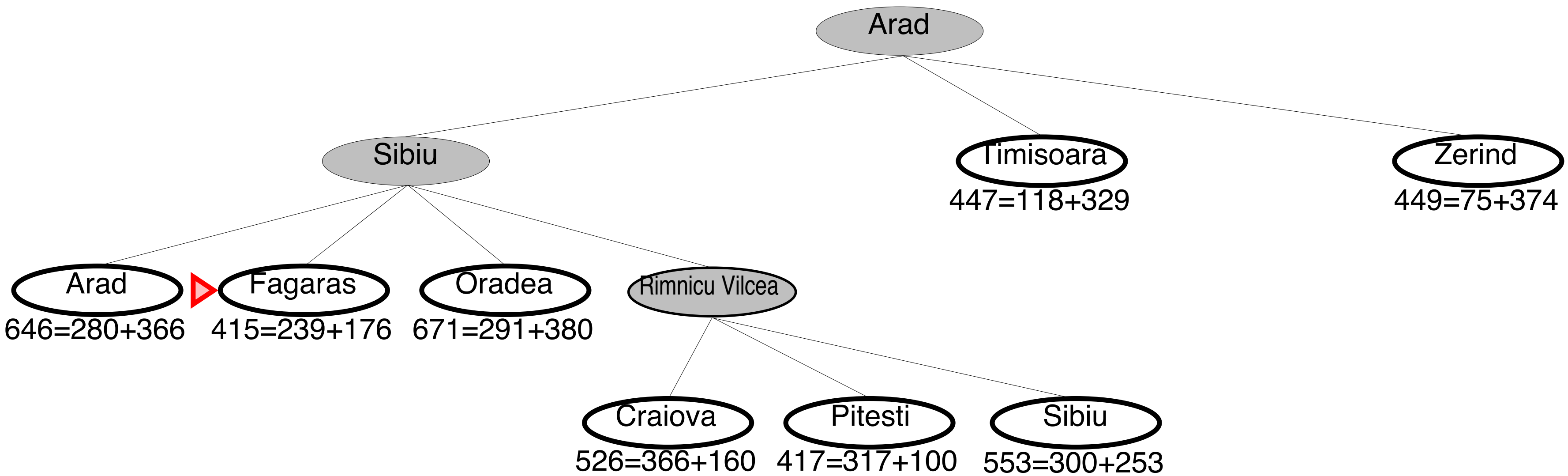
A* search example



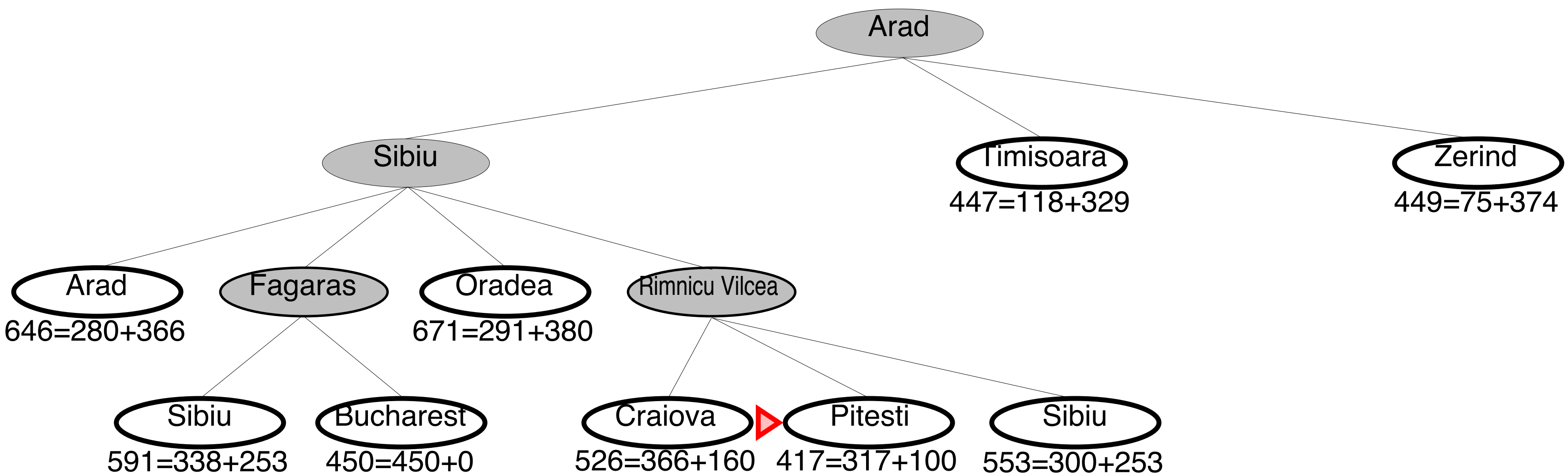
A^* search example



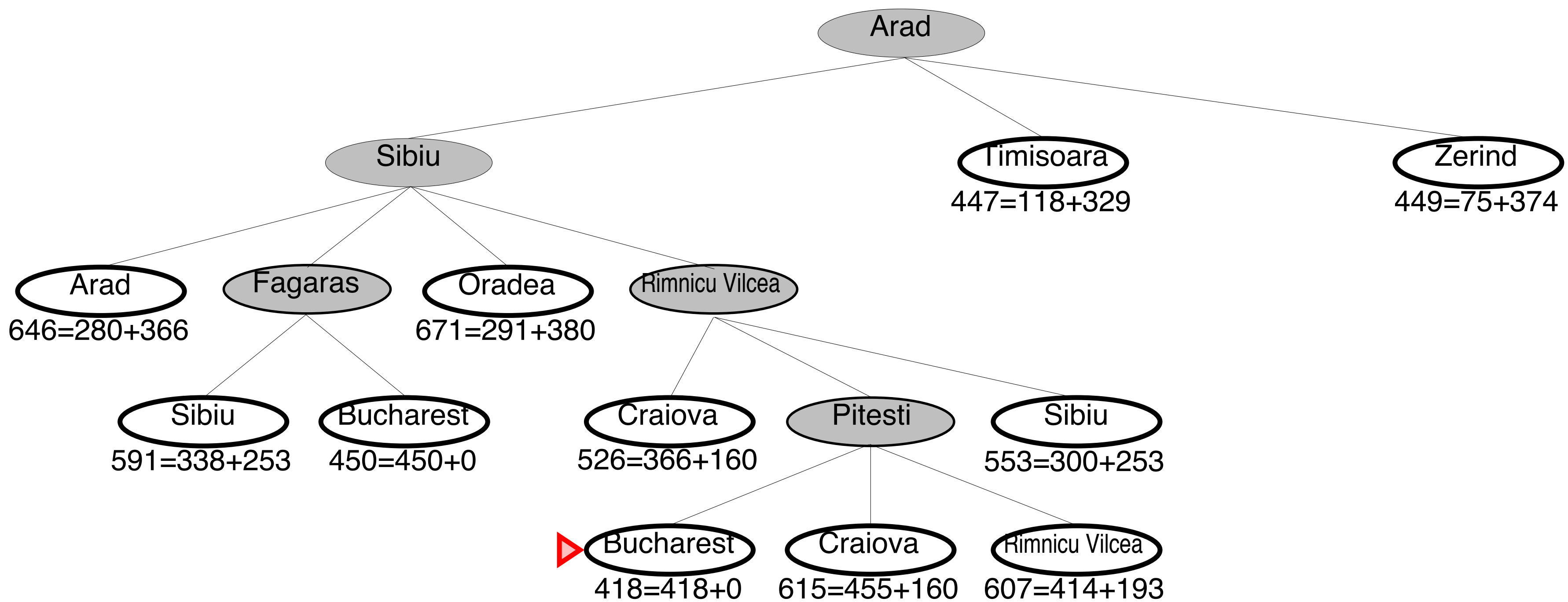
A* search example



A* search example



A* search example



Optimality of A*

Let:

C^* be the optimal cost and C be the cost returned

n be a node in the optimal path, which is unexpanded

$g^*(n)$ be the cost to reach n

$h^*(n)$ be the cost of the optimal path to reach the goal.

$f(n) > C^*$

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

$f(n) = g^*(n) + h(n)$ n is on an optimal path

$f(n) \leq g^*(n) + h^*(n)$ admissibility

$f(n) \leq C^*$ contradiction!

Consistent Heuristics

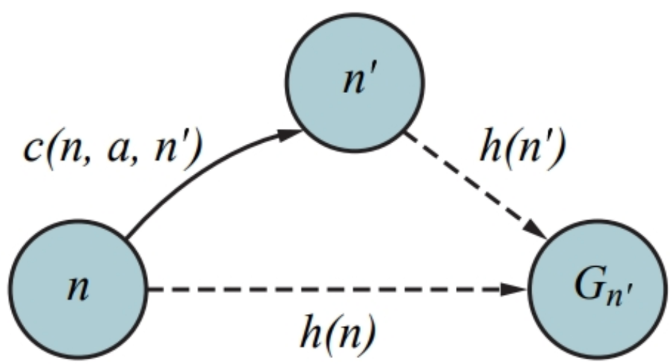
For some admissible heuristics, f can *decrease* in a path. This is a problem when the search space is a **graph**.

A heuristics is **consistent** if

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

This guarantees that f is not decreasing (monotonic)

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



Pathmax

Usually, admissible heuristics are consistent.

In alternative, pathmax can be build for a heuristics:
Instead of $f(n') = g(n') + h(n')$, take

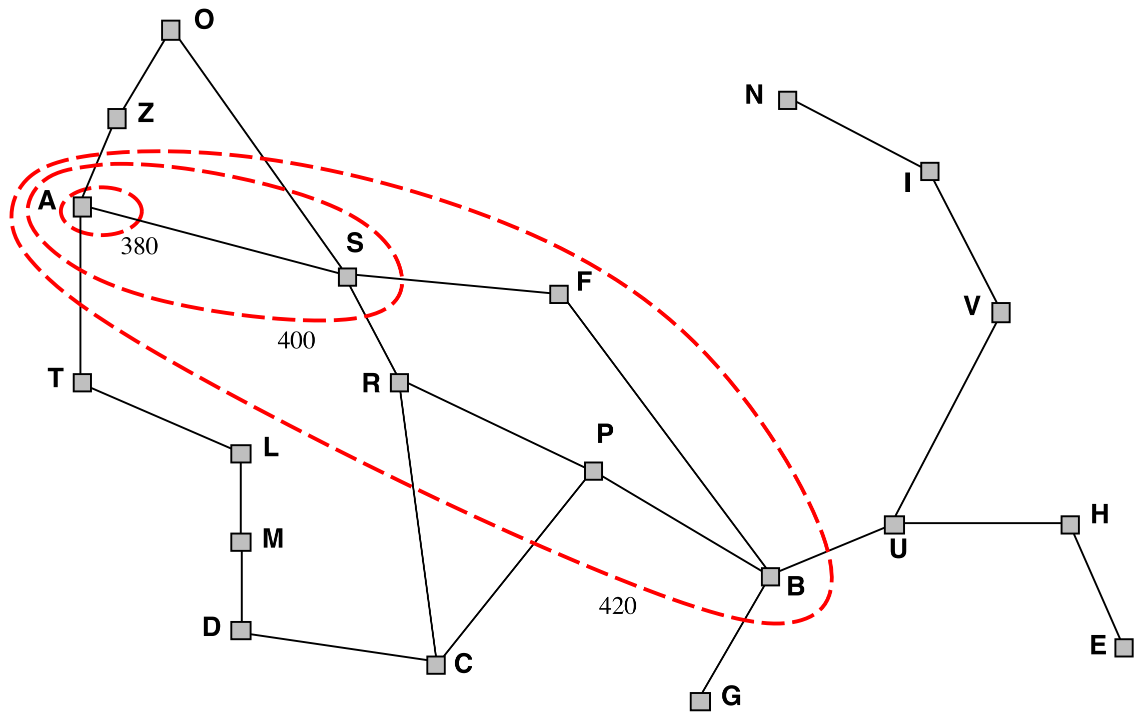
$$f(n') = max(g(n') + h(n'), f(n))$$

Using **pathmax**, f is always non decreasing

Optimality of A* (more useful)

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

Gradually adds "f-contours" of nodes (cf. uniform search)
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 $f \leq f(G)$

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 $f(n) > C^*$

Limitations of: A^*

◇ the memory to store the $frontier$ may be exponential.

Countermeasures:

- Use good, but not admissible heuristics
- Live with sub-optimal solutions

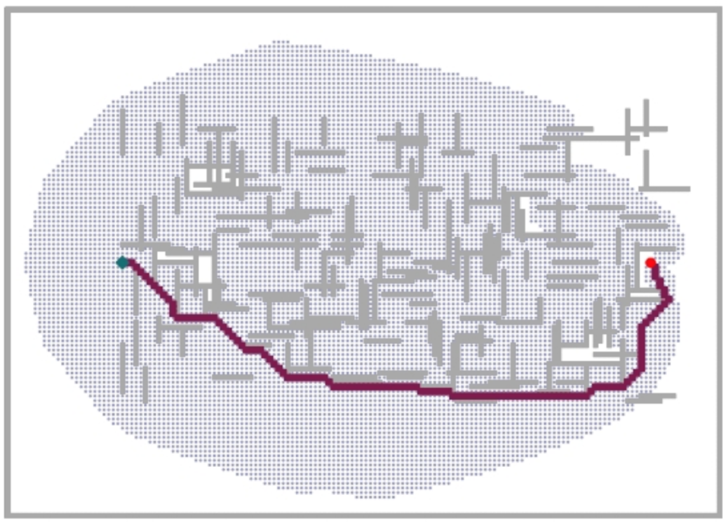
◇ space problem remains (as in breadth first).

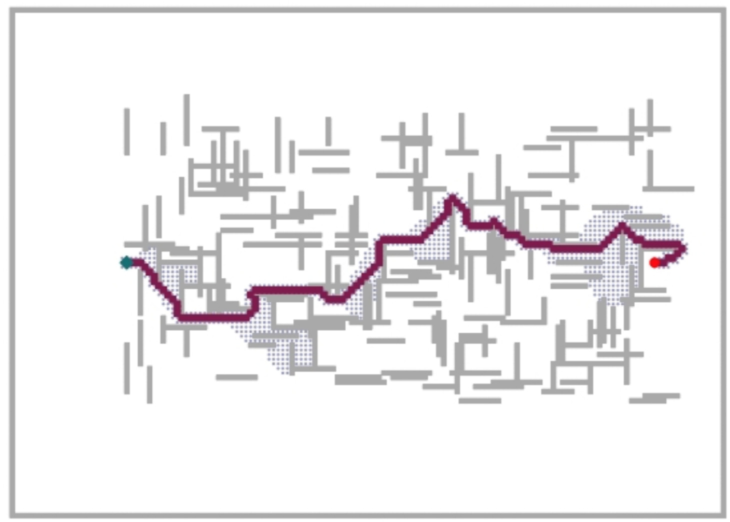
Generalizing the use of heuristics

Weigthed A^*: trade optimality for speed

$f(n) = g(n) + W * h(n)$ with $W > 1$

cost is bound by a factor W wrt optimal cost





detour index: $W = 1.3$

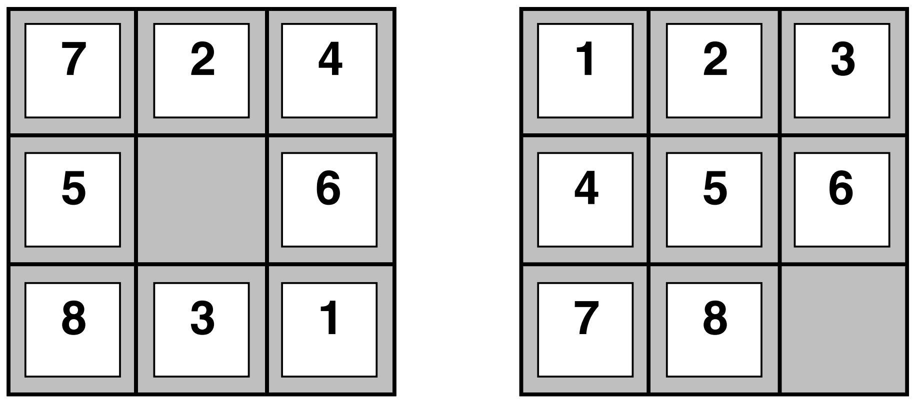
Variants of A*

- ♢ Iterative Deepening A* (IDA*)
- ♢ Recursive Best First Search (RBFS)
- ♢ Memory Bounded A*

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles (not including the blank)
$h_2(n)$ = 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:

$d = 14$ BFS $=$ 6783 nodes
A$^*(h_1) =$ 678 nodes
A$^*(h_2) =$ 174 nodes
$d = 28$ BFS $=$ 463234 nodes
A$^*(h_1) =$ 202565 nodes
A$^*(h_2) =$ 22055 nodes

$\#$ nodes expanded by A* with a dominant h is always less.

Effective branching factor

$N =$ total number of nodes expanded by A*
$d =$ the depth of the solution
b^* is the branching factor of a uniform tree of depth d with $N + 1$ nodes.

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d.$$

Example: if A* finds a solution at depth 5 with 52 nodes, then the effective branching factor is 1,92.

Effective branching factor: 8-puzzle

$d = 14$ IDS = 1,77 ebf
A$^*(h_1)$ = 1,47 ebf
A$^*(h_2)$ = 1,31 ebf
$d = 28$ IDS = 1,53
A$^*(h_1)$ = 1,49ebf
A$^*(h_2)$ = 1,36 ebf

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

Combination of heuristics

What if we have several heuristics not dominating each other?

let $h_1 \dots h_m$ a collection of such heuristics, define

$$h(n) = max(h_1(n), \dots, h_m(n)).$$

h is admissible and dominates $h_1 \dots h_m$

Summary

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Summary

- ♢ FIFO
- ♢ LIFO
- ♢ priority cue

- $W = 0$ (i.e. $h(n) = 0$) Uniform search
- $W = \infty$ (i.e. $g(n) = 0$) Best-first search
- $W = 1$ A*

Looking for heuristics

- Finding relaxed problems (via a formal problem specification)
- Pattern DataBases (use solutions to subproblems)
- Introducing landmarks
- Learning by looking at the "meta-level" computation tree.
- Learning the heuristics by successfully solving several instances of the problem

Summary

♢ Heuristics make the difference

♢ Finding good heuristics is key to success, but ideal heuristics means no search!

♢ A* is optimal even though it uses too much memory