

# HEURISTIC SEARCH<sup>1</sup>

## LECTURE 4

---

<sup>1</sup>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
- $A^*$
- Variants of  $A^*$
- Heuristics

## Best-first search

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

⇒ Expand most desirable unexpanded node

Implementation:

*frontier* is a queue sorted in decreasing order of desirability

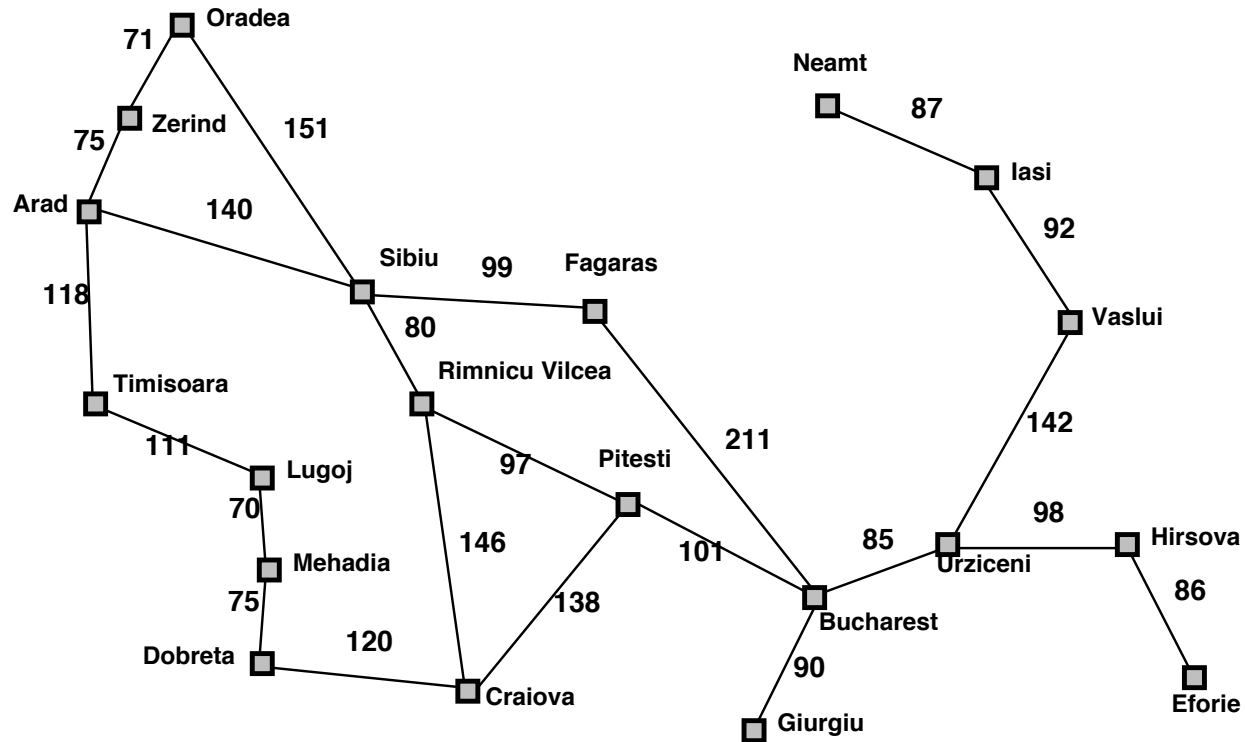
Special cases:

“best-first” search

A\* search

**best?** “Best-first” vs greedy

# Romania with step costs in km



Straight-line distance  
to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

## Greedy “Best-first” search

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

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

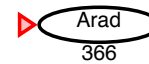
E.g.,  $h_{\text{SLD}}(n)$  = straight-line distance from  $n$  to Bucharest

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

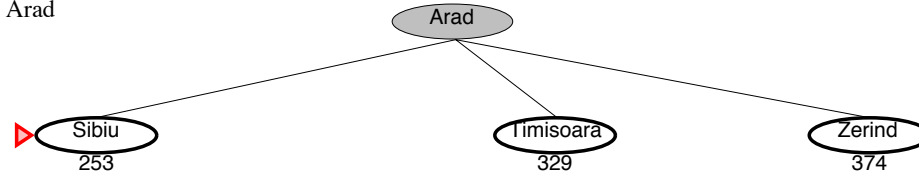
**A remark about heuristics:** Usually a good  $h$  greatly improves the search (by expanding less nodes), but knowing ideal  $h$  amounts to knowing the solution ...

# Greedy search example

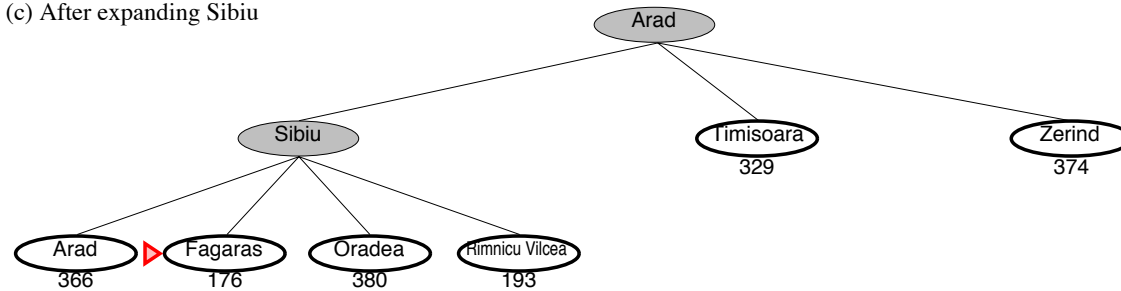
(a) The initial state



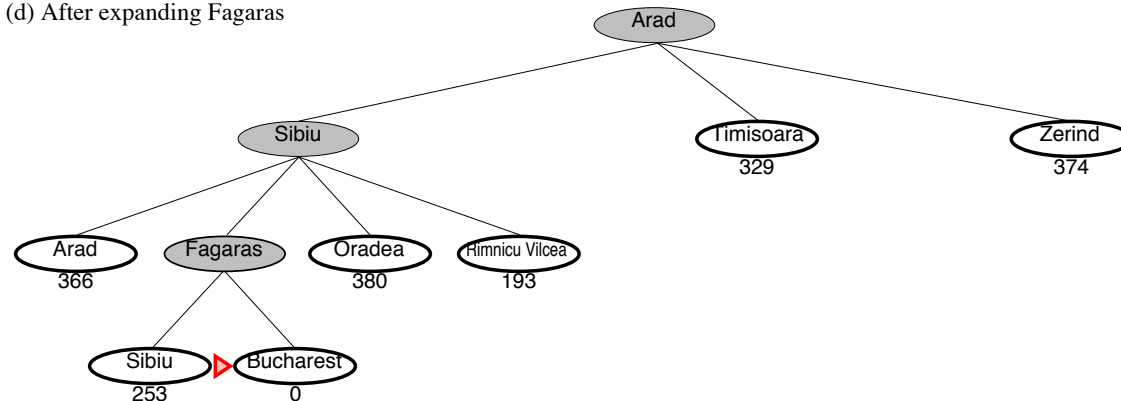
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



## Properties of “best-first” search

**Complete** No—can get stuck in loops, e.g.,

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

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

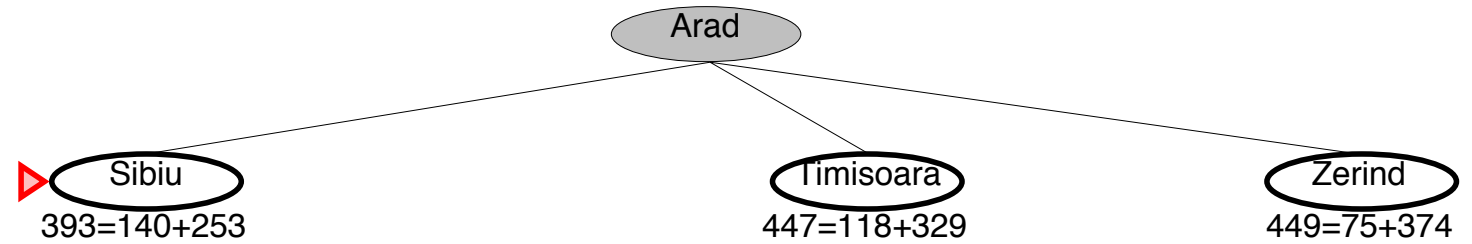
**Key property:** A\* search is optimal



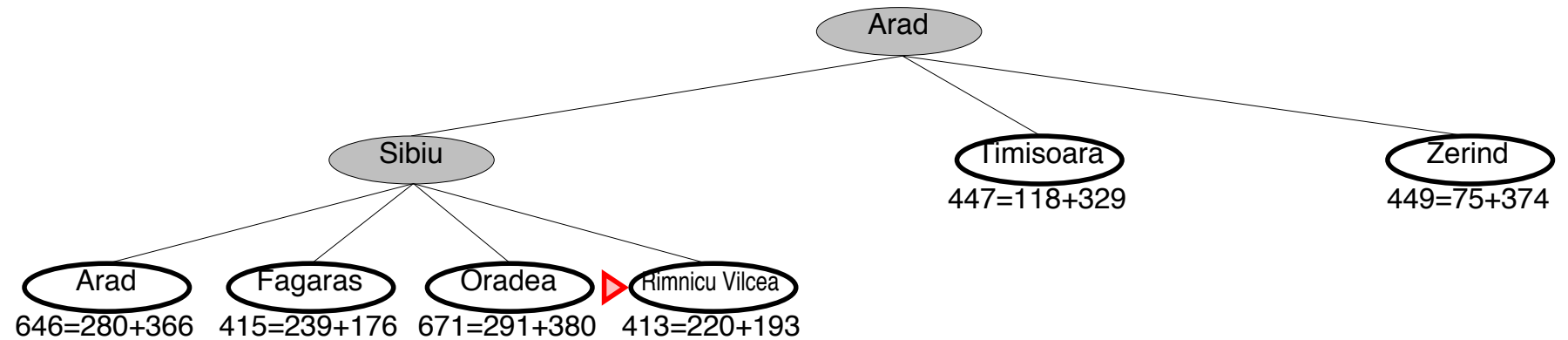
# A\* search example

▶ Arad  
366=0+366

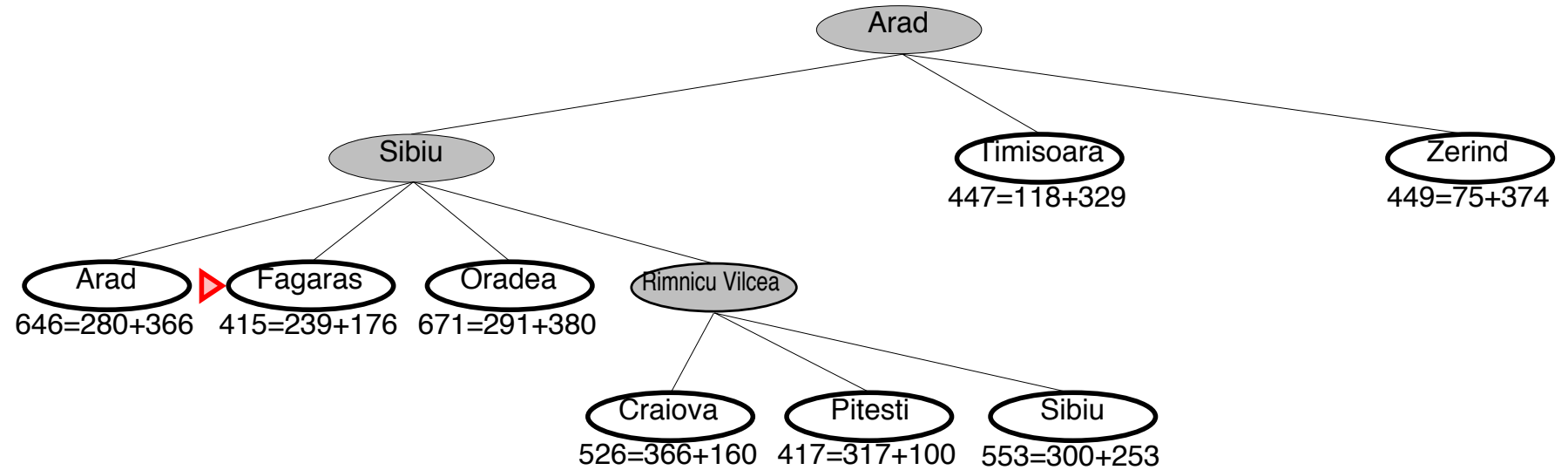
# A\* search example



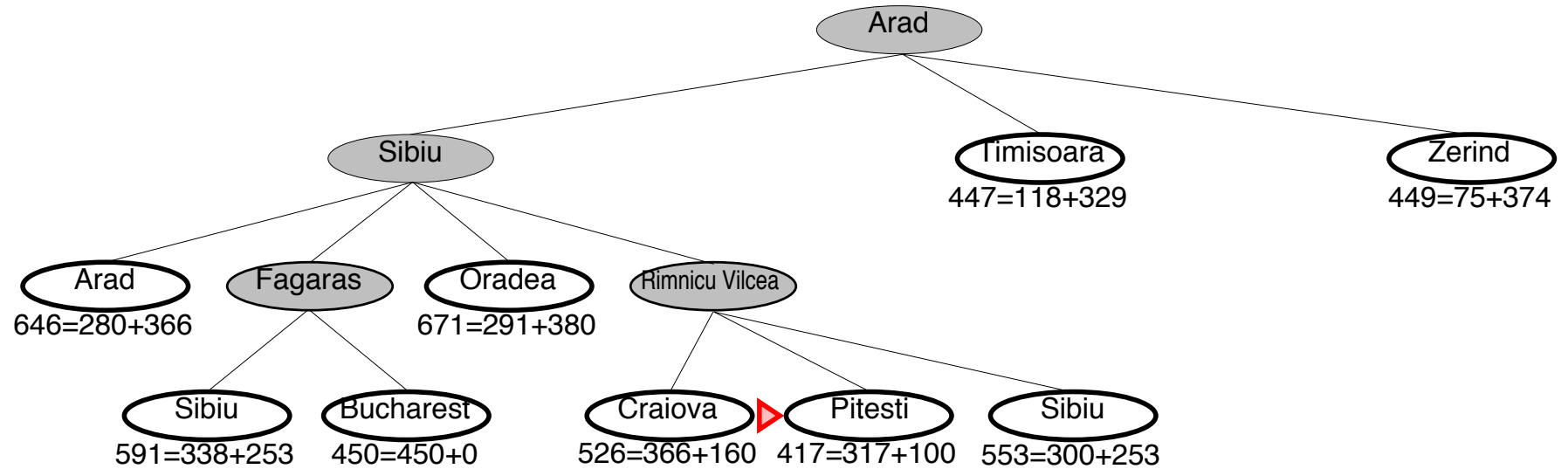
# A\* search example



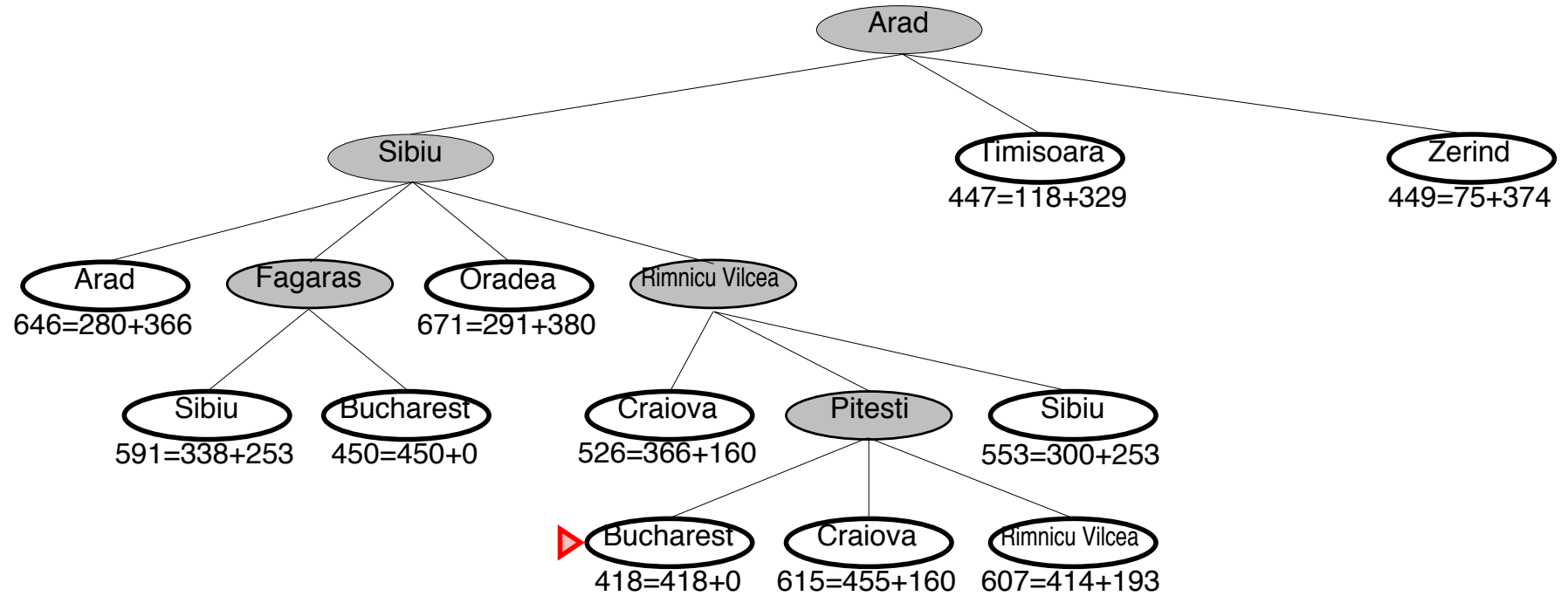
# A\* search example



# 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.

$$f(n) > C^*$$

$$f(n) = g(n) + h(n) \text{ by definition}$$

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

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

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

## Consistent Heuristics

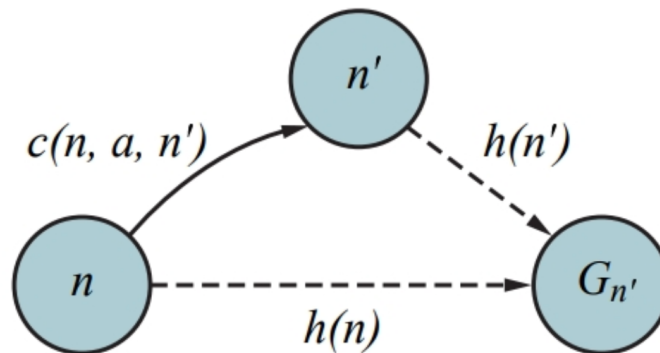
For some admissible heuristics,  $f$  can *decrease* in a path. This is a problem when the search space is a **graph**.

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

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

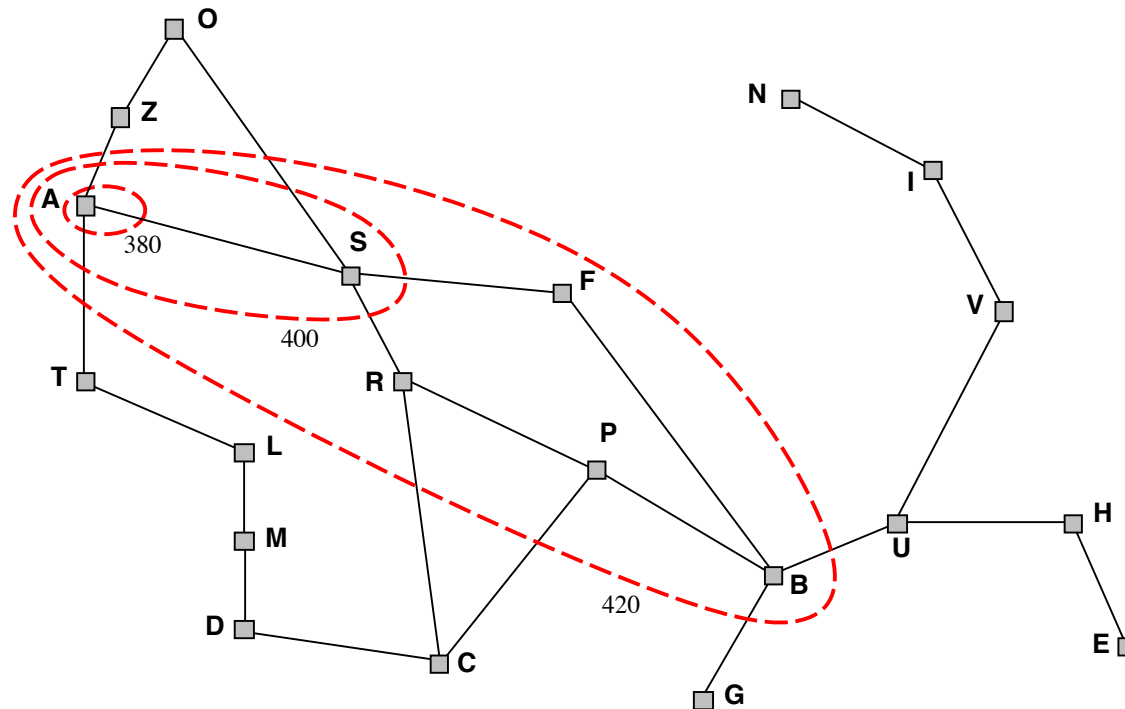
Using **pathmax**,  $f$  is always non decreasing

## 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

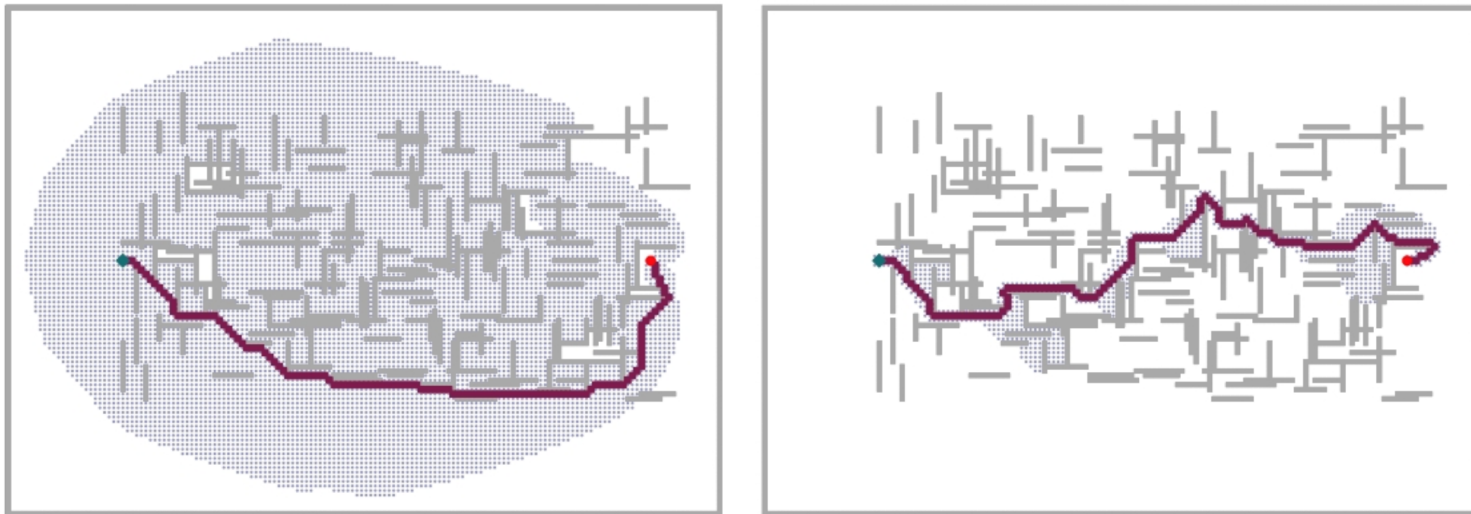
◇ space problem remains (as in breadth first).

## Generalizing the use of heuristics

**Weighted  $A^*$ :** trade optimality for speed

$$f(n) = g(n) + W * h(n) \quad \text{with } W > 1$$

cost is bound by a factor  $W$  wrt optimal cost



detour index:  $W = 1.3$

## Limiting the memory requirement

- ◇ **frontier** + **reached states**
- ◇ Beam search (puts a limit on the states in the frontier)
- ◇ Remove states from **reached states** when not reachable

## Limiting the memory requirement: $IDA^*$

- ◇ Iterative Deepening search where  $f$  ( $g + h$ ) is fixed
- ◇  $IDA^*$  is complete and optimal as  $A^*$  using linear memory
- ◇ the next limit can be chosen by increasing  $f$  of a fixed amount or looking at the values of the successors

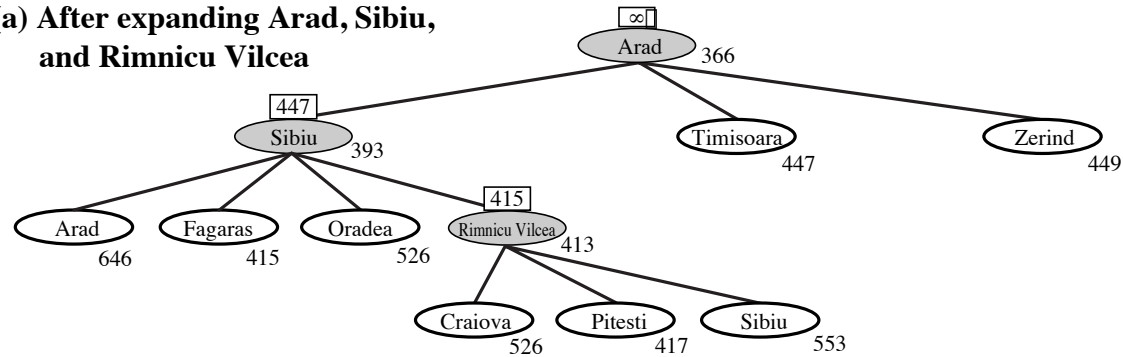
**Weakness:** same sources of inefficiencies as uniform-cost search

## Recursive Best First Search: RBFS

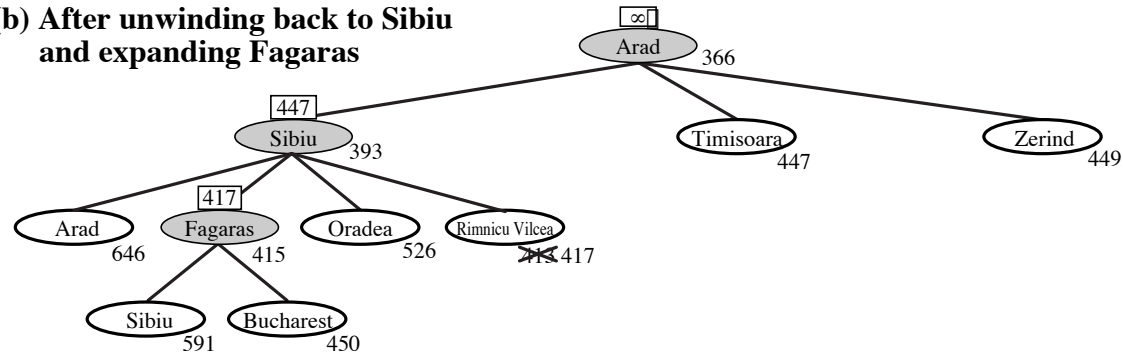
- ◇ Stores the  $f$  of the unexpanded nodes and goes depth-first till  $f$  of the current node becomes worse of one of the previous alternatives
- ◇ Backs up recording the best  $f$  of the nodes discarded by backtracking
- ◇ more efficient than  $IDA^*$  but sometimes it wastes time in re-generating discarded nodes.
- ◇ space complexity is linear  $O(bd)$



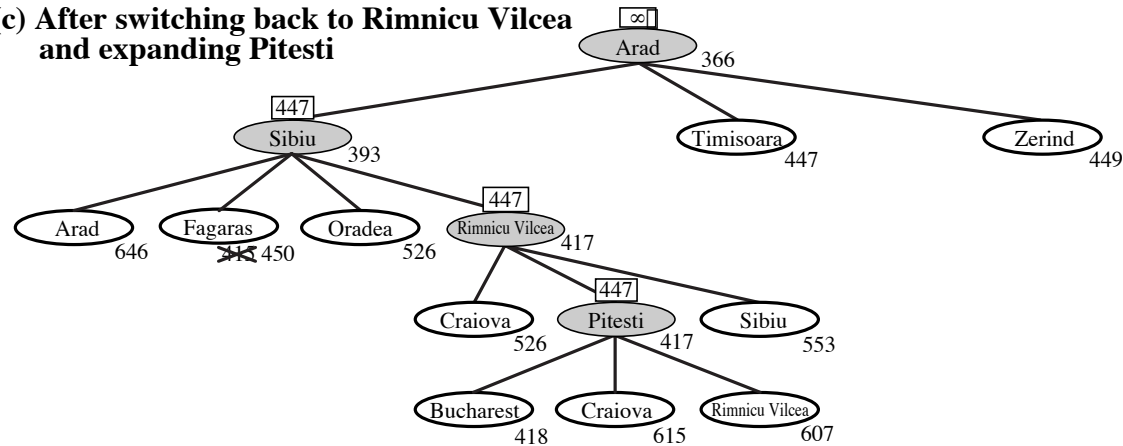
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



## SMA\*

The problem with IDA\* and RBFS is that they use *too little* memory.

- in IDA\* at each iteration only the current cost limit for  $f$  is kept,
- in RBFS the cost of the nodes in the depth first search are recorded

**SMA\*** (Simplified Memory-Bounded A\*) can use all the available memory for the search.

Idea: better remember a node that re-generate it when needed

## Basic intuition of SMA\*

Like A\* till there is memory available.

When a new node must be generated the node with the highest  $f$  is discarded (**forgotten node**), while keeping its cost  $f$  in the parent node.

A discarded node will be re-generated *only* when *all other paths* are worse than the forgotten node.

## Properties of SMA\*

- It uses all the available memory.
- It avoids to repeat states until the available memory is exhausted.
- It is complete if the available memory allows the *shallowest* solution path to be recorded.
- It is optimal if the available memory allows the *shallowest* optimal solution path to be recorded. Otherwise it returns the best solution attainable with the available memory.
- The search is optimally efficient, when the available memory can store the whole search

## Usage of SMA\*

SMA\* is typically the best heuristic search method

In difficult problems, SMA\* **flounders** when it keeps re-generating states that have already been discarded.

Memory limitations can lead to unacceptable computing time (as for **trashing** as in pagination).

## Bidirectional heuristic search

In the case of two frontier priority queues:

$$f_2(n) = \max(2g(n), g(n) + h(n))$$

- ◇ can estimate the cost **start-end** or the cost to frontier
- ◇ works well for average heuristics

## 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)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

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.

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

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

$$A^*(h_1) = 1,47 \text{ ebf}$$

$$A^*(h_2) = 1,31 \text{ ebf}$$

$d = 28$  IDS = 1,53

$$A^*(h_1) = 1,49 \text{ ebf}$$

$$A^*(h_2) = 1,36 \text{ ebf}$$

## 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  $h_1 \dots h_m$

## Looking for heuristics

- Learning by looking at the “meta-level computation tree.
- Finding relaxed problems (via a formal problem specification)
- Pattern DataBases (use solutions to subproblems)
- Introducing landmarks
- 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 but uses too much memory
- ◇  $SMA^*$  Best HS (compromise between memory and time)