Heuristics and A* Search

Hal Daumé III

Computer Science
University of Maryland

me@hal3.name

CS 421: Introduction to Artificial Intelligence

2 Feb 2012

Many slides courtesy of
Dan Klein, Stuart Russell,
or Andrew Moore

Announcements

- Office hours:
  - Angjoo: Tuesday 1:00-2:00
  - Me: Thursday 1:30-3:15
  - We will usually schedule "overload" office hours before the week that projects are due
- Project 1 posted
  - Checkpoint: by next class, you should have pacman running without problems (see FAQ)
  - (Ideally, also do DFS by next class)
  - There is a lot to learn in this project
  - The entire infrastructure will be used in all projects
  - Start NOW!

Today

- Heuristics
- A* Search
- Heuristic Design
- Local Search

Recap: Search

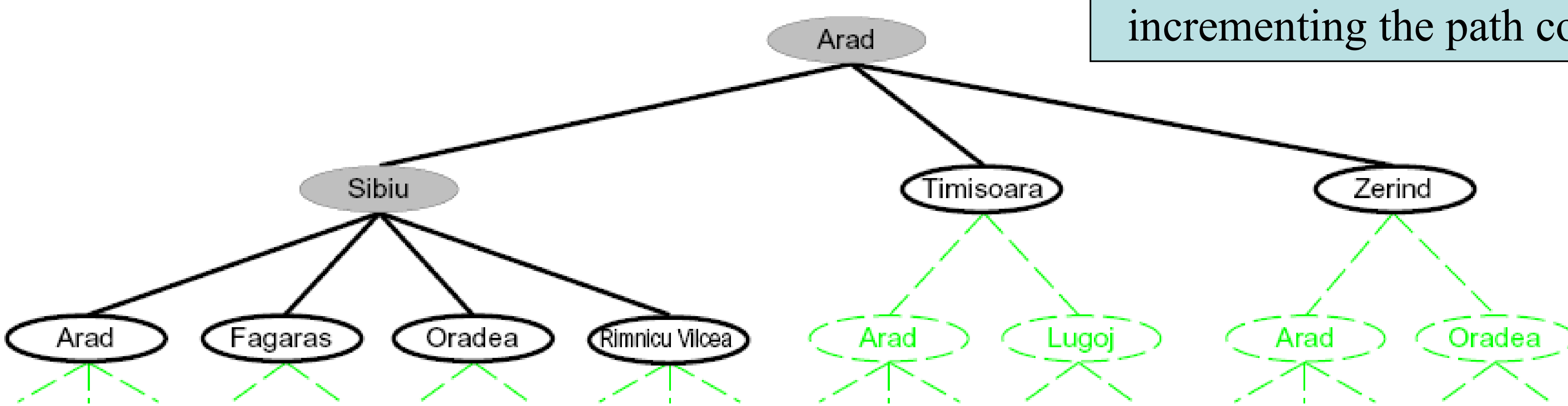
- Search problems:
  - States (configurations of the world)
  - Successor functions, costs, start and goal tests
- Search trees:
  - Nodes: represent paths / plans
  - Paths have costs (sum of action costs)
  - Strategies diff

$$g(n) = \sum_{x \rightarrow y \in n} cost(x \rightarrow y)$$

General Tree Search

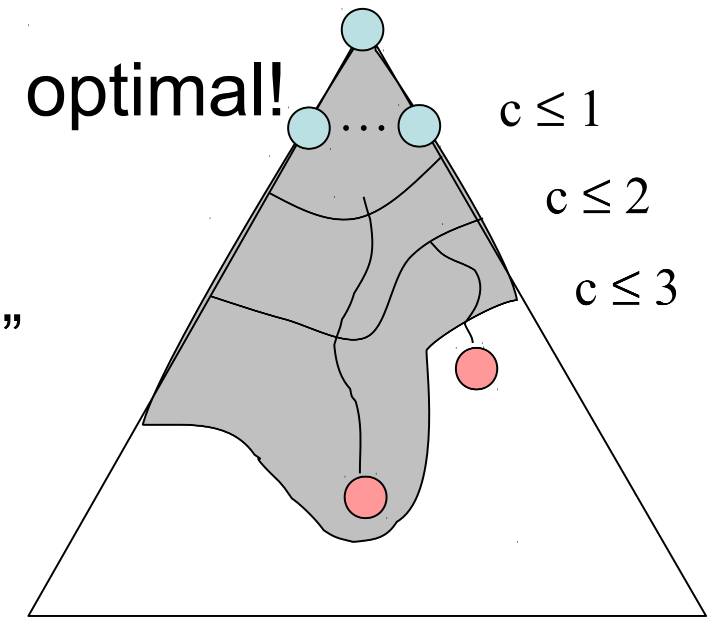
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

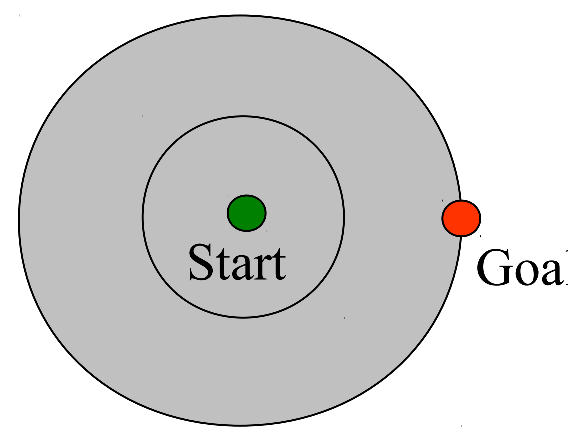
Expanding includes incrementing the path cost!



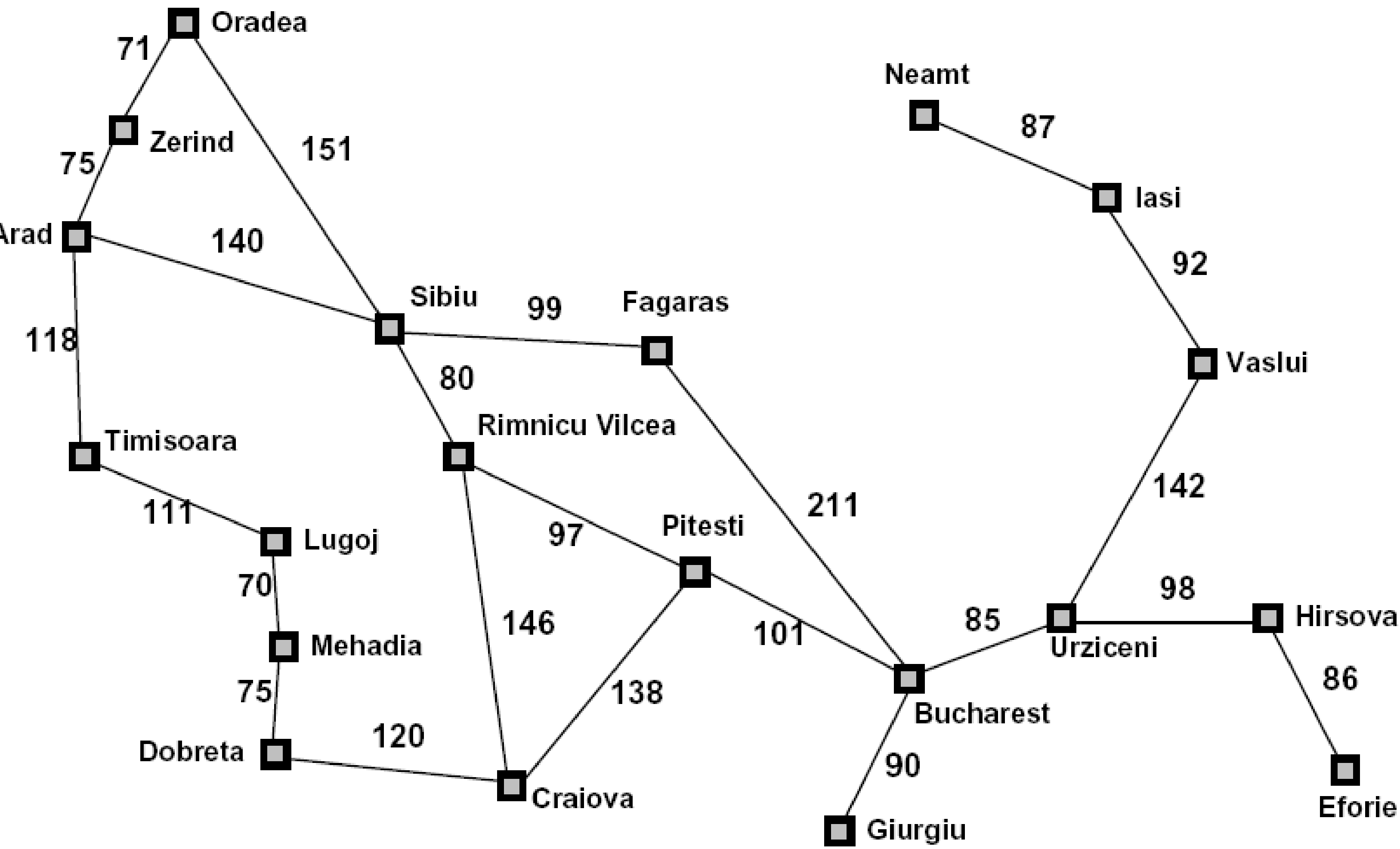
Uniform Cost

- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location





Heuristics

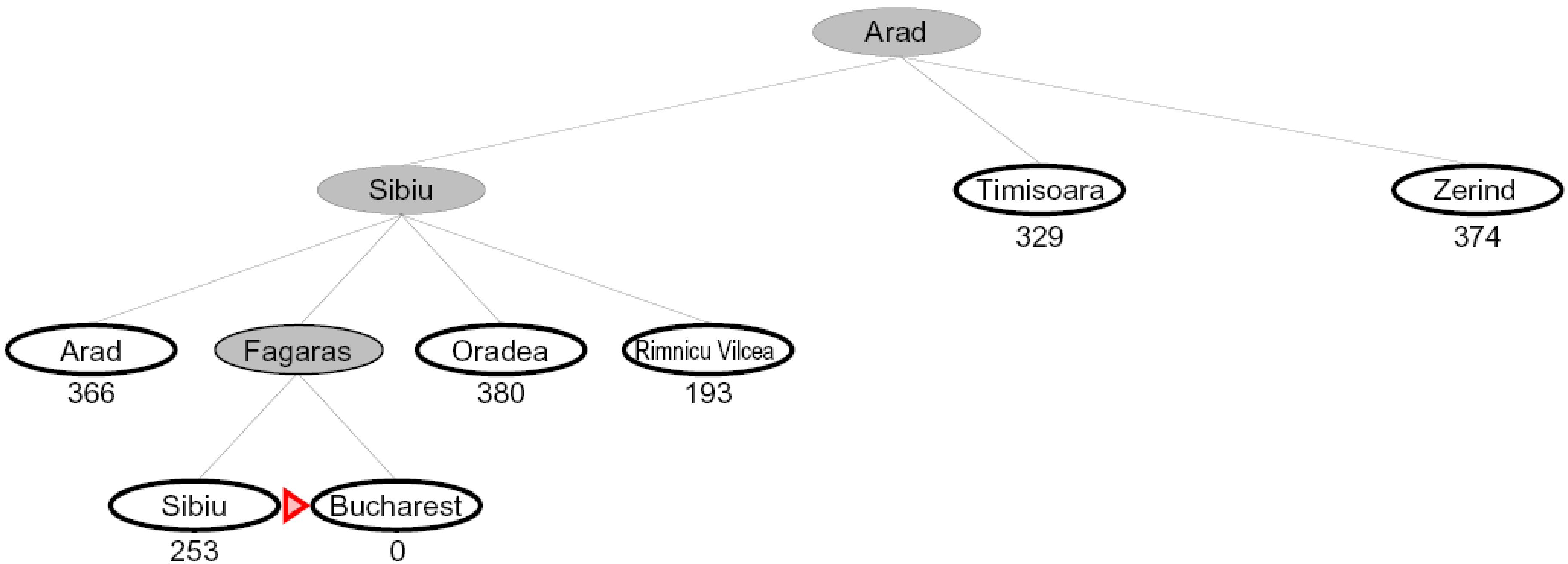


Straight–line distance to Bucharest

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

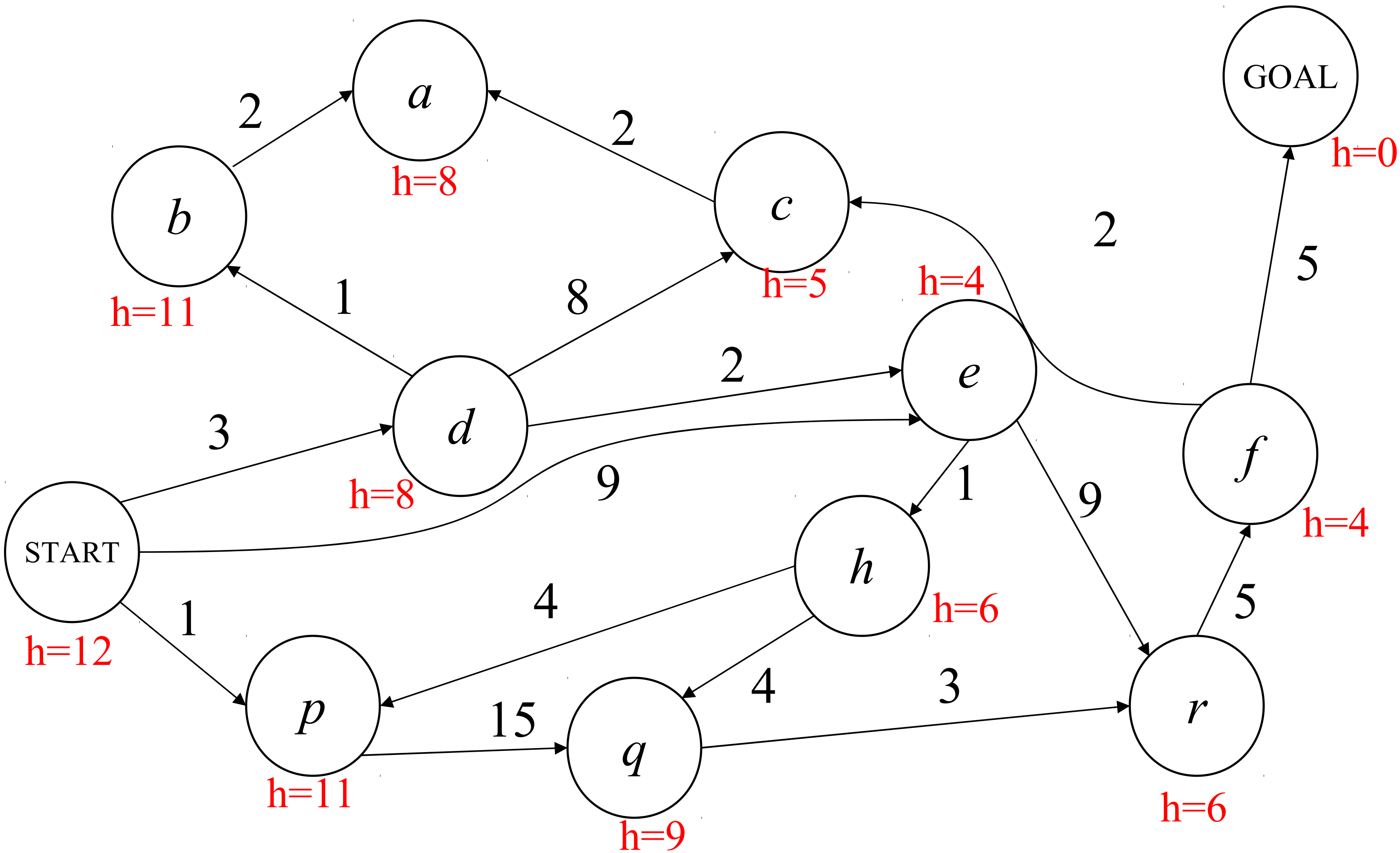
Best First / Greedy Search

- Expand the node that seems closest…



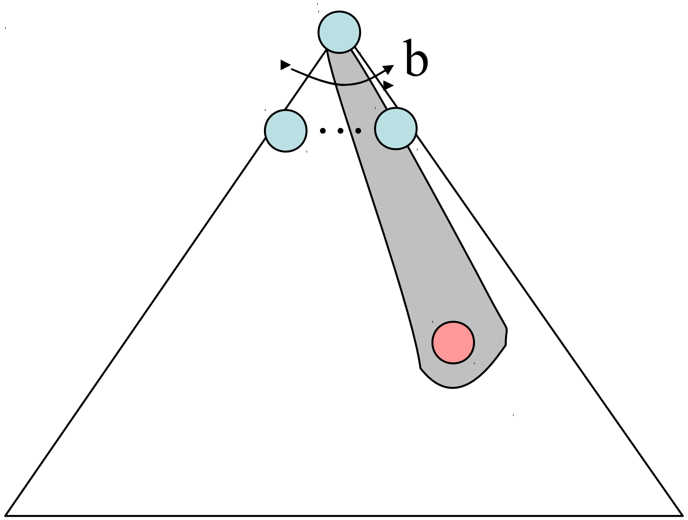
- What can go wrong?

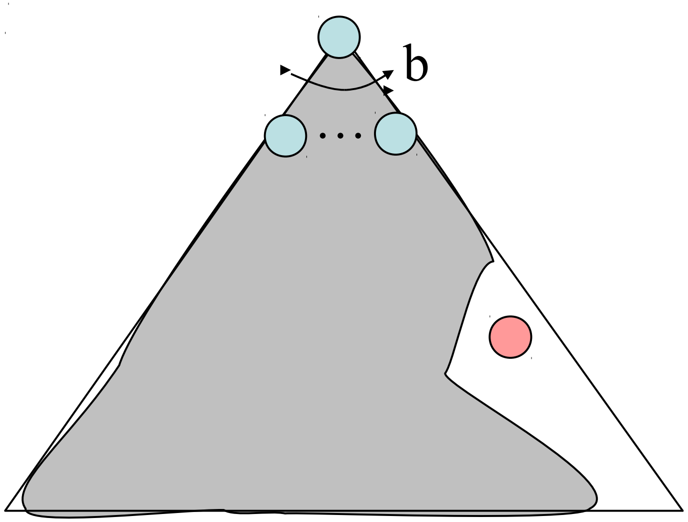
Best First / Greedy Search



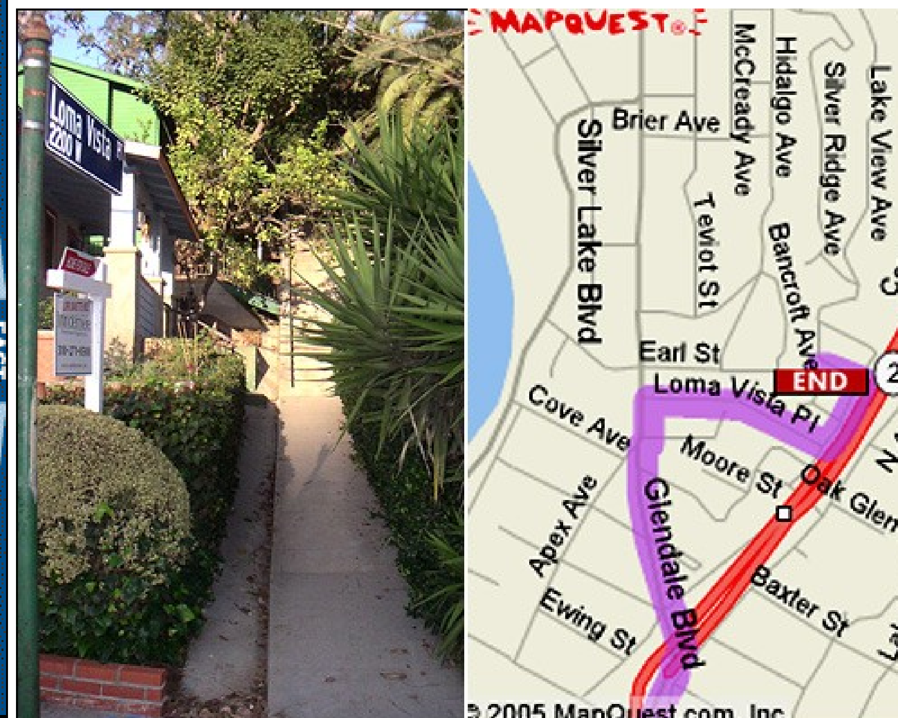
Best First / Greedy Search

- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)



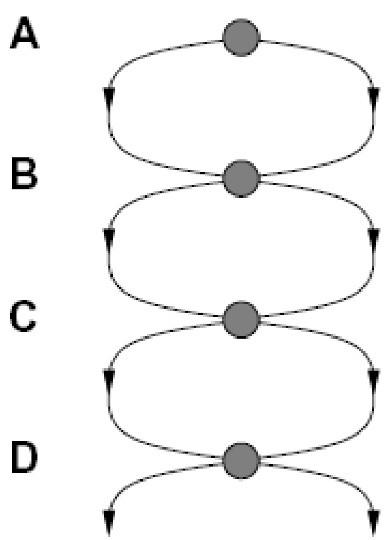


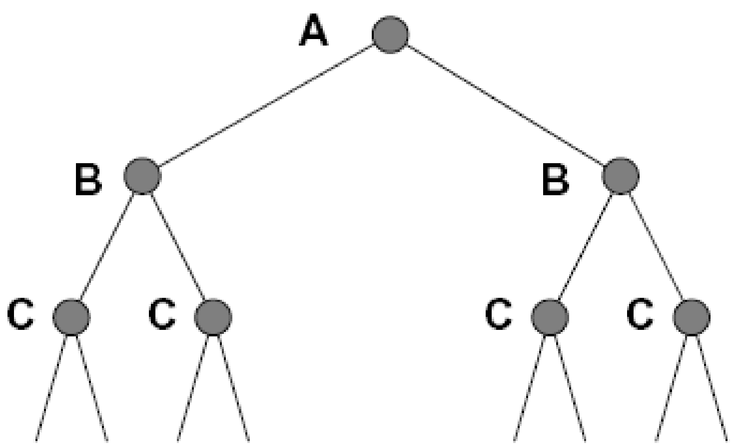
Search Gone Wrong?



Extra Work?

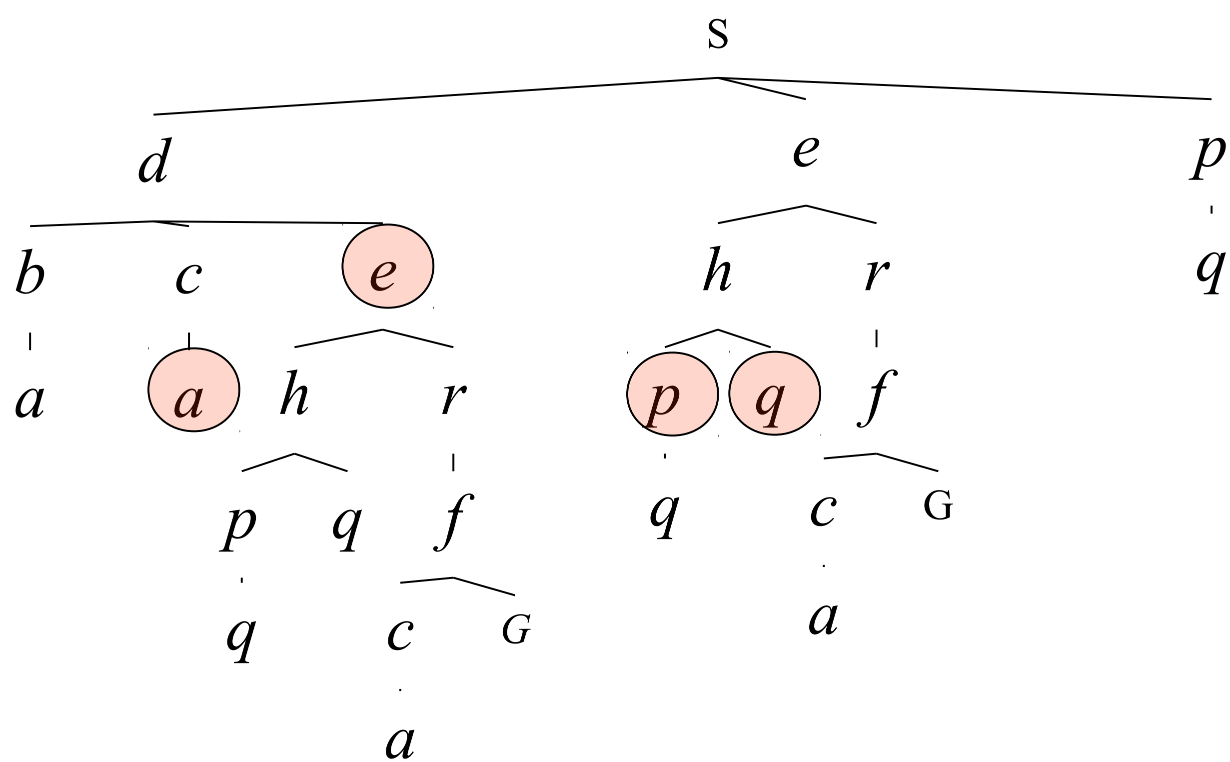
- Failure to detect repeated states can cause exponentially more work. Why?





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

- Very simple fix: never expand a state type twice

```
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
```

- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?

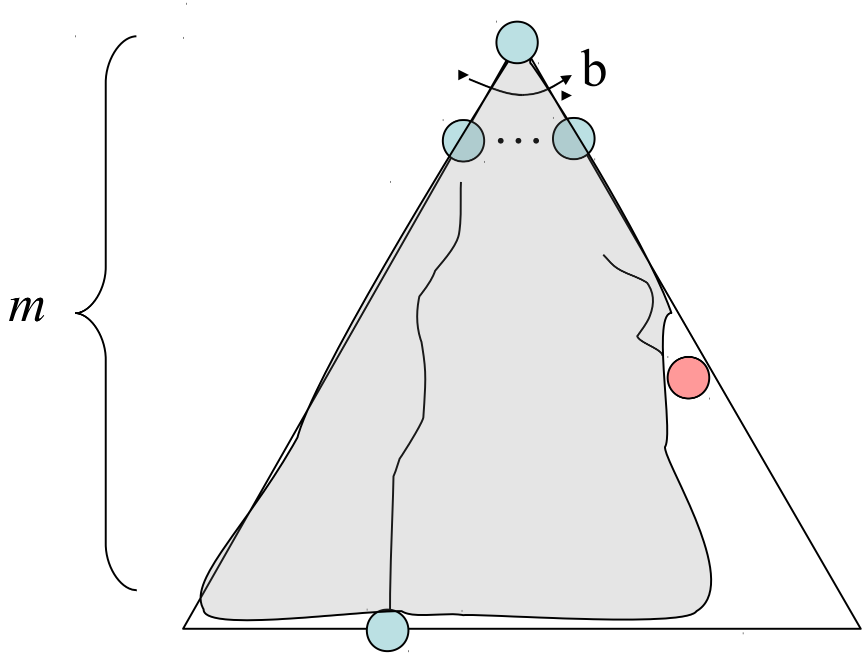
Some Hints

- Graph search is almost always better than tree search (when not?)
- Fringes are sometimes called “closed lists” – but don’t implement them with lists (use sets)!
- Nodes are conceptually paths, but better to represent with a state, cost, and reference to parent node

Best First Greedy Search

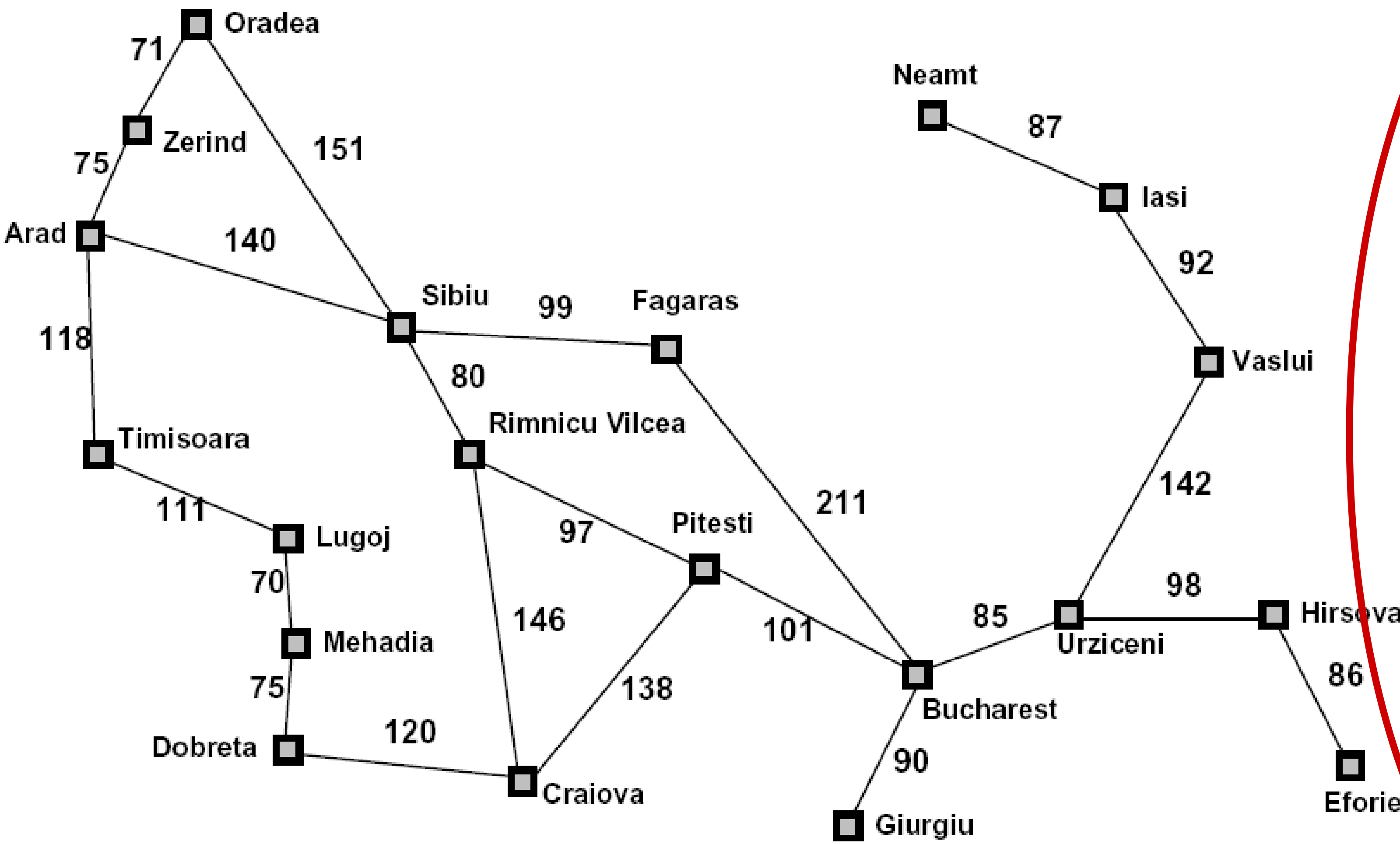
n	# states
b	avg branch
C^*	least cost
s	shallow goal
m	max depth

Algorithm	Complete	Optimal	Time	Space
Greedy Best-First Search	Y*	N	$O(b^m)$	$O(b^m)$



- What do we need to do to make it complete?
- Can we make it optimal?

Example: Heuristic Function

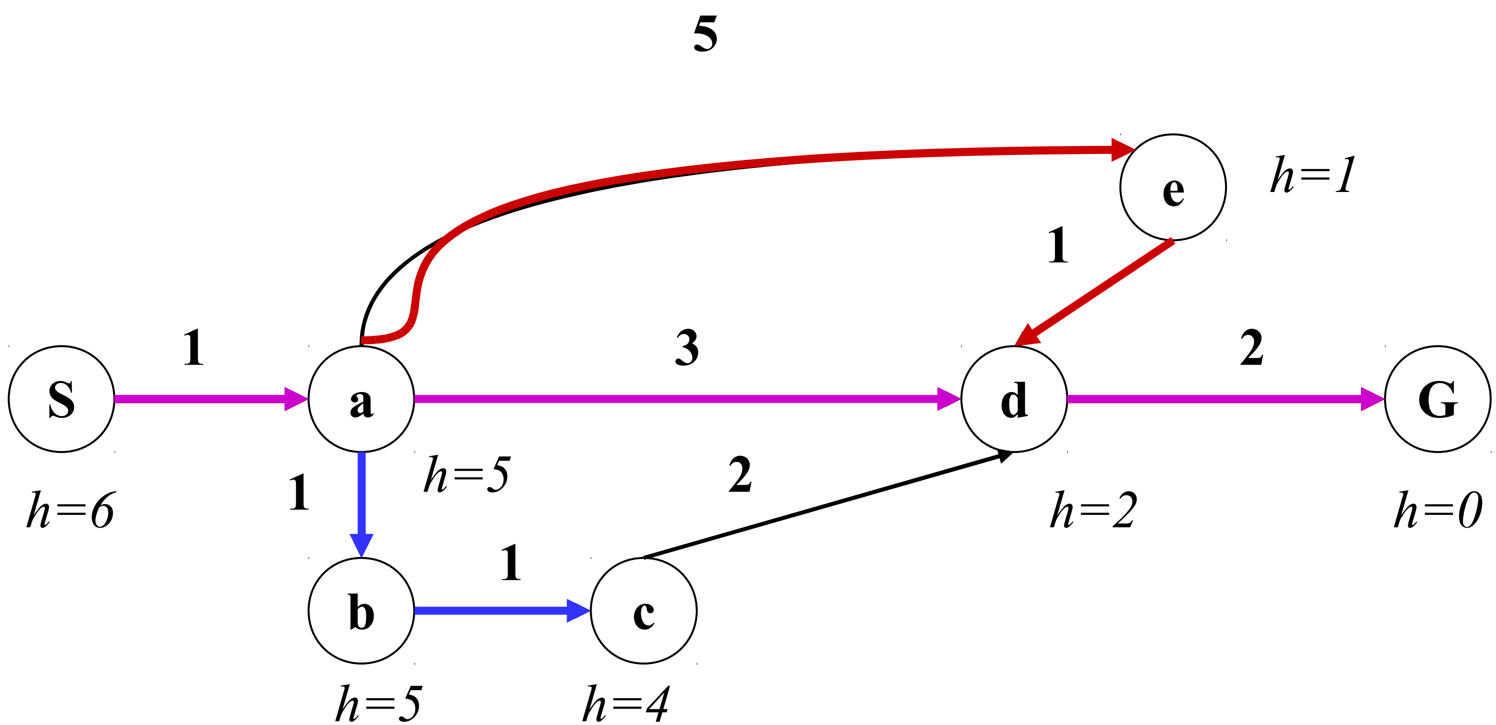


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Best-first orders by goal proximity, or *forward cost* h(n)

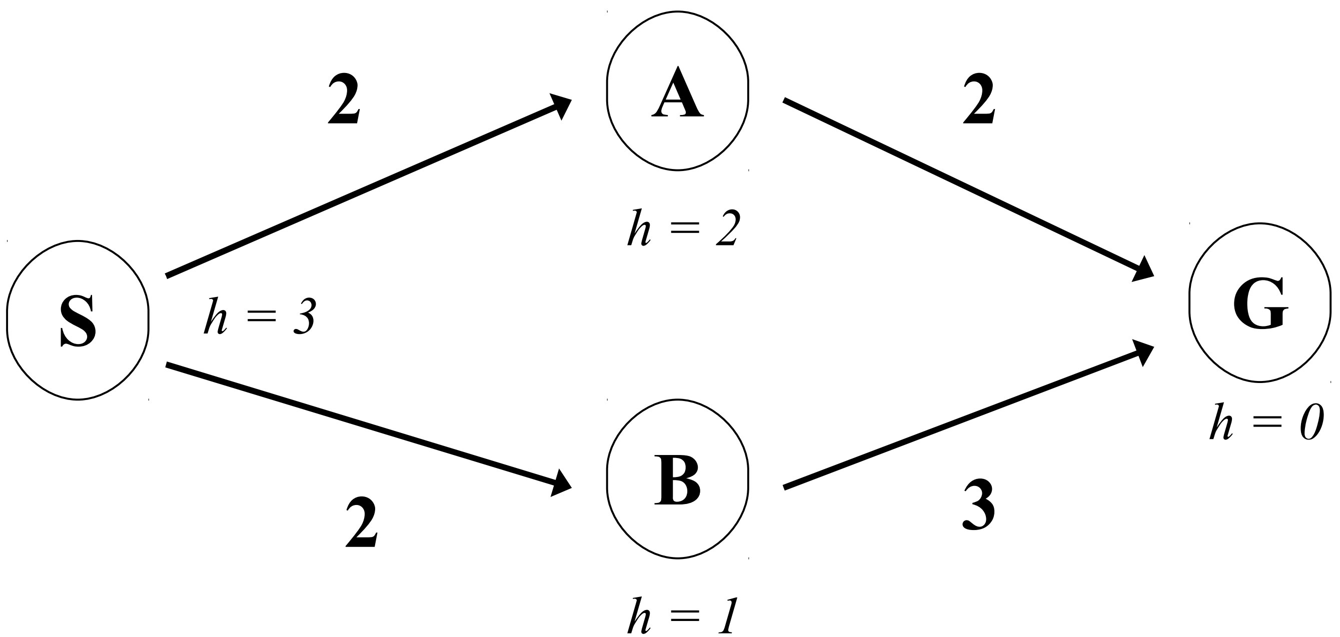


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

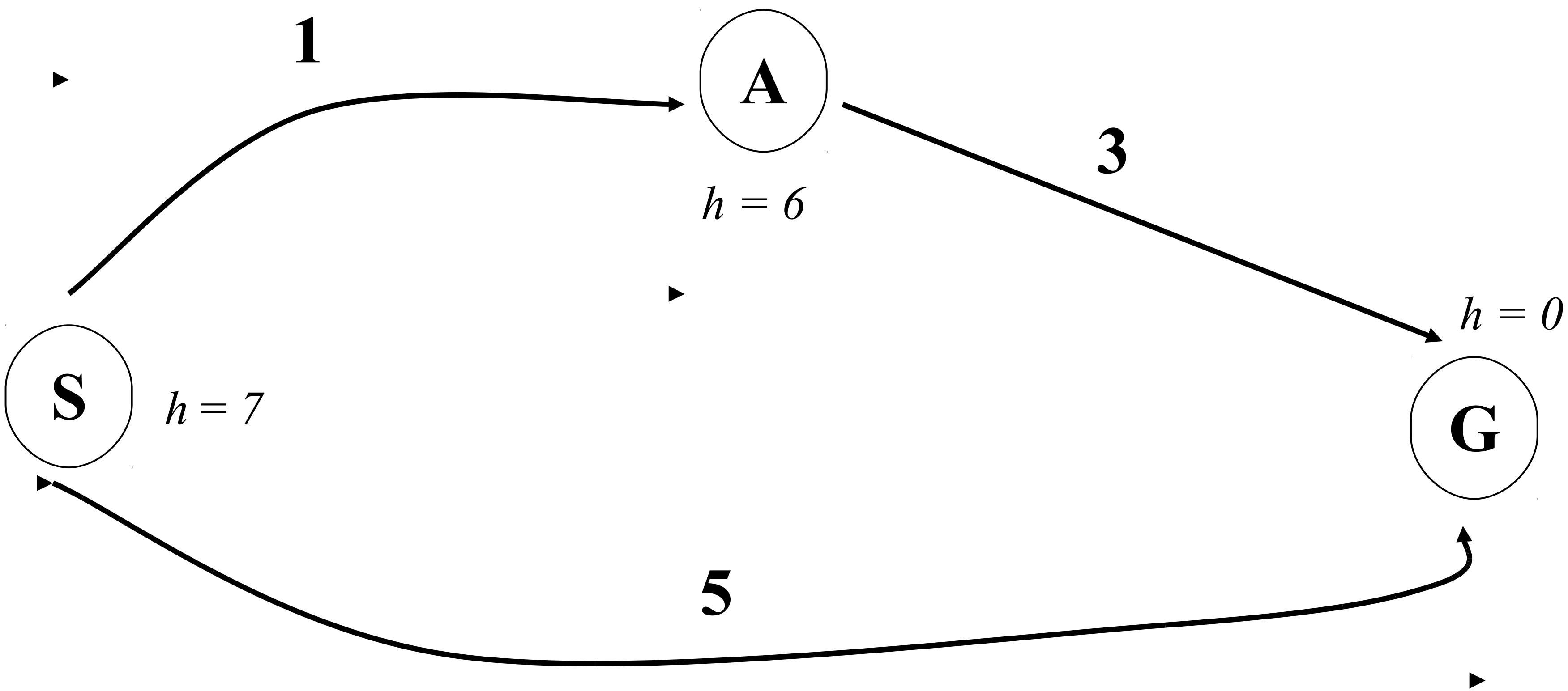
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



- What went wrong?
- Actual bad goal cost > estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

$$h(n) \leq h^*(n)$$

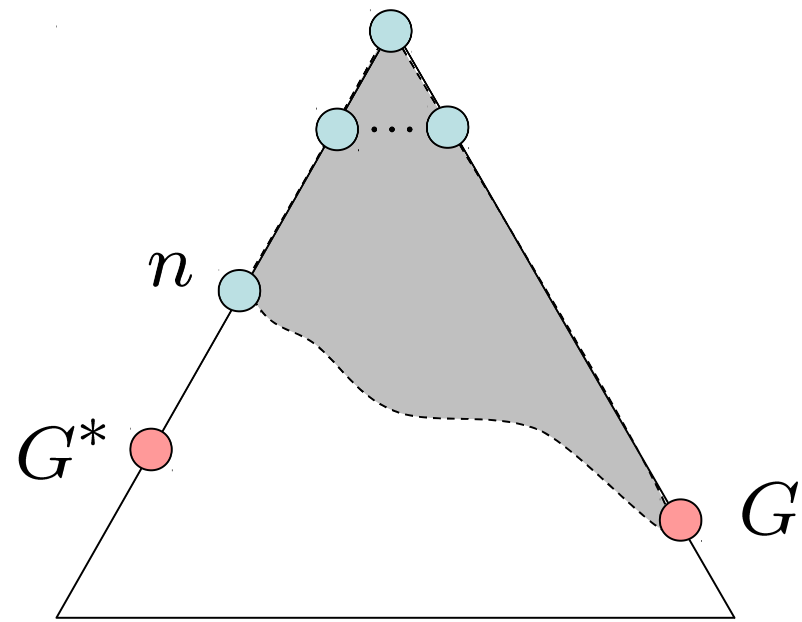
where $h^*(n)$ is the true cost to a nearest goal

- E.g. Euclidean distance on a map problem

- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A*: Blocking

- Proof:
  - What could go wrong?
  - We'd have to have to pop a suboptimal goal G off the fringe before G*
  - This can't happen:
    - Imagine a suboptimal goal G is on the queue
    - Some node *n* which is a subpath of G* must be on the fringe (why?)
    - *n* will be popped before G



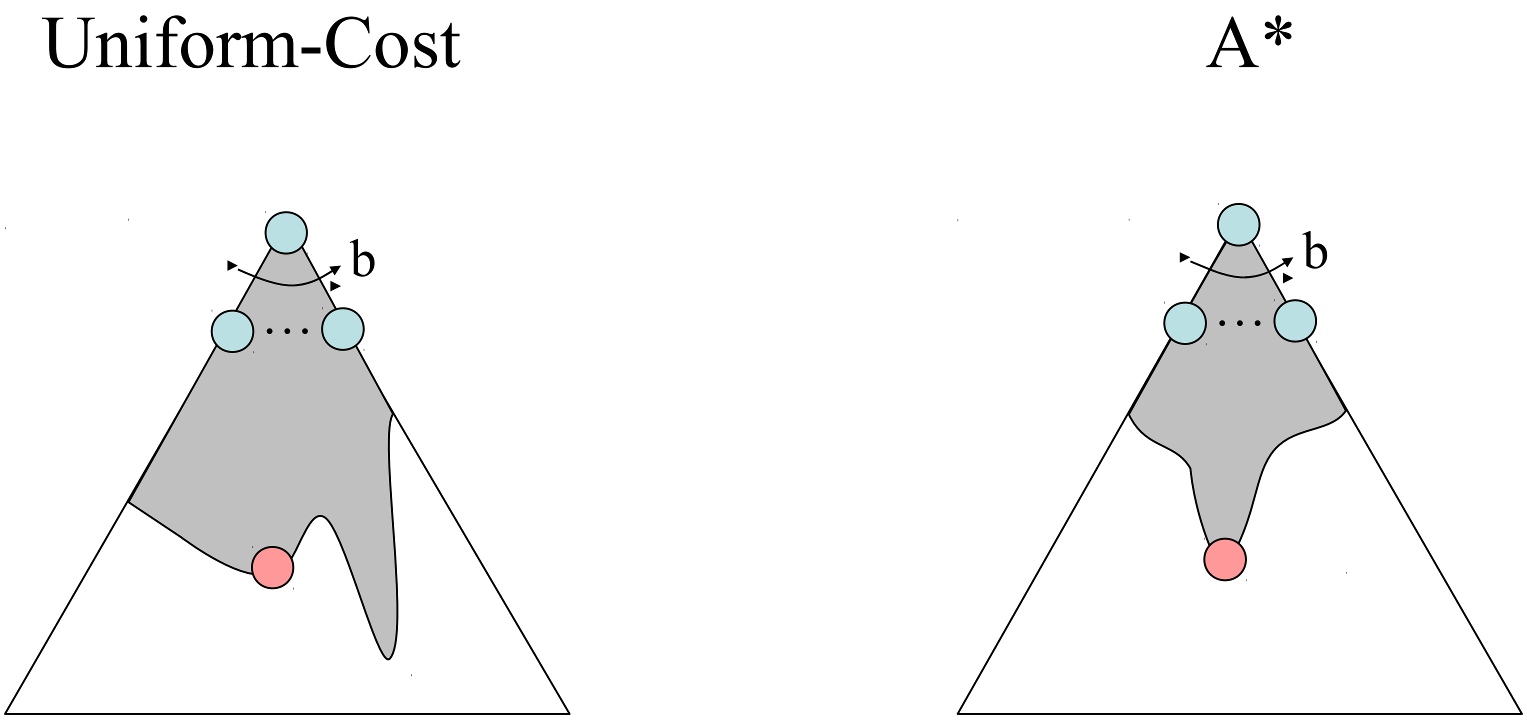
$$f(n) \leq g(G^*)$$

$$g(G^*) < g(G)$$

$$g(G) = f(G)$$

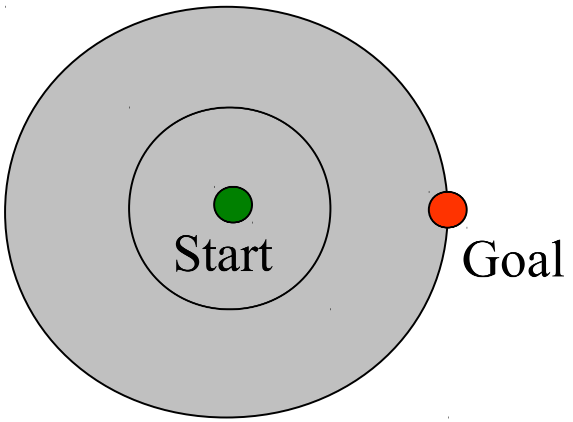
$$f(n) < f(G)$$

Properties of A*

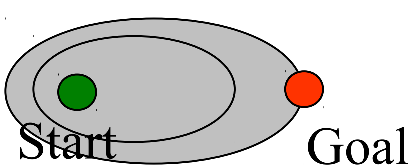


UCS vs A* Contours

- Uniform-cost expanded in all directions



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

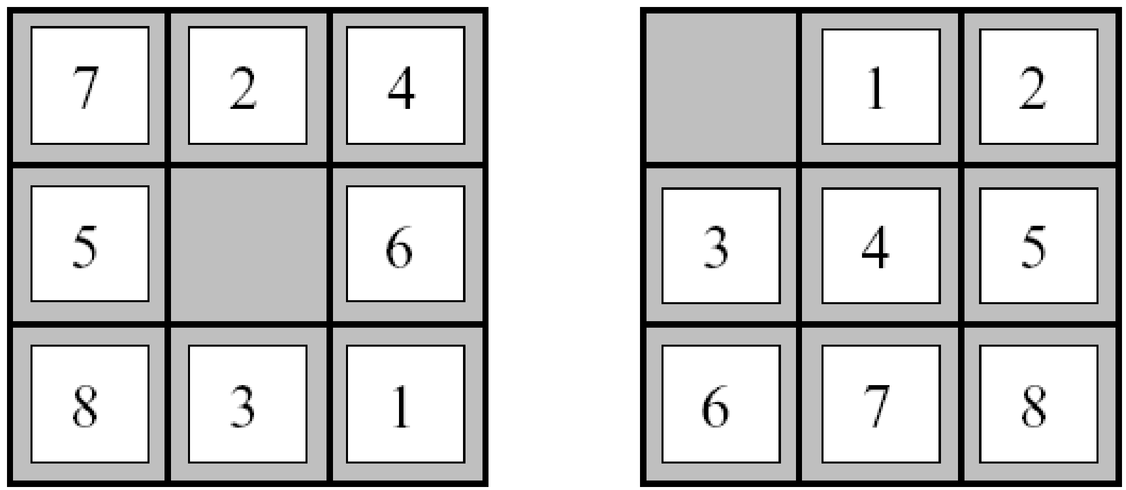


[demo: position search UCS / A*]

Admissible Heuristics

- Most of the work is in coming up with admissible heuristics
- Inadmissible heuristics are often quite effective (especially when you have no choice)
- Very common hack: use α x h(n) for admissible h, α > 1 to generate a faster but less optimal inadmissible h' from admissible h

Example: 8 Puzzle

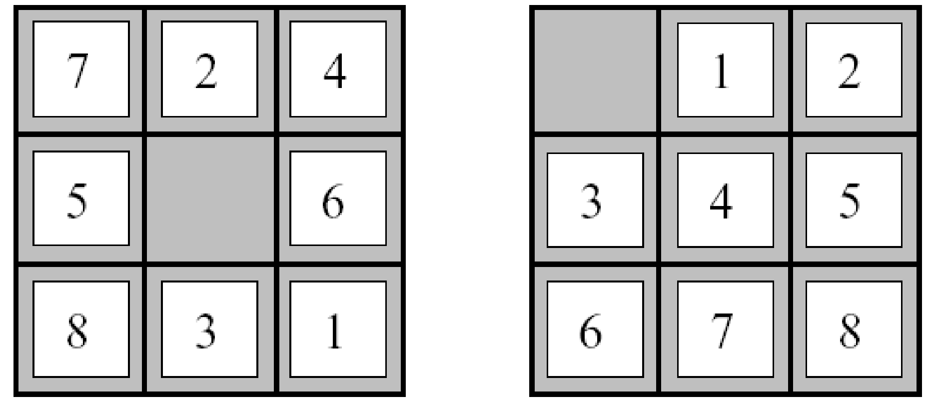


Start State Goal State

- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

8 Puzzle I

- Number of tiles misplaced?
- Why is it admissible?
- h(start) =
- This is a relaxed-problem heuristic



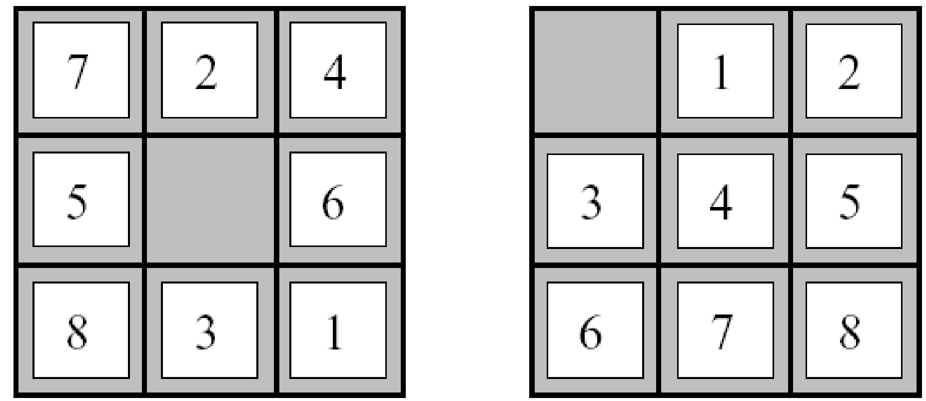
Start State Goal State

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
ID	112	6,300	3.6×10^6
TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?

- h(start) =



Start State Goal State

	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MAN-HATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

Trivial Heuristics, Dominance

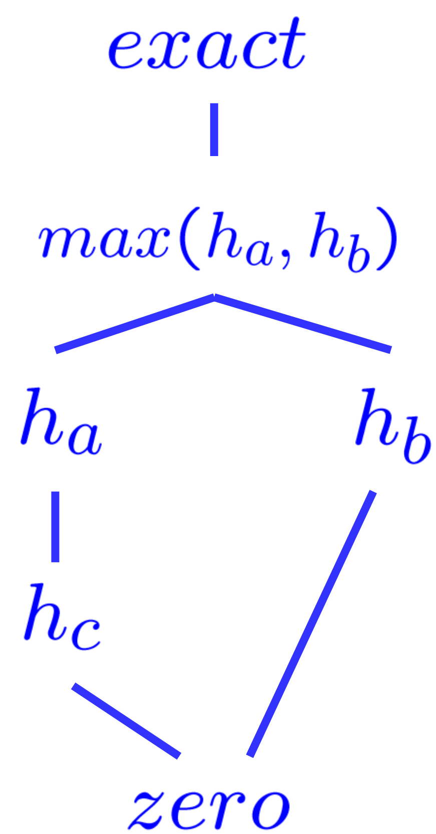
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

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

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



Where do heuristics come from?

- Classically designed by hand (and still...)
- Alternatively, can you watch a person (or “optimal” agent) and try to *learn* heuristics from their demonstrations?

Examples of demonstrations

Examples of demonstrations

Course Scheduling

- From the university's perspective:
  - Set of courses $\{c_1, c_2, \ldots c_n\}$
  - Set of room / times $\{r_1, r_2, \ldots r_n\}$
  - Each pairing (c_k, r_m) has a cost w_{km}
  - What's the best assignment of courses to rooms?
- States: list of pairings
- Actions: add a legal pairing
- Costs: cost of the new pairing

- Admissible heuristics?

- (Who can think of a *algorithms* answer to this problem?)

Other A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

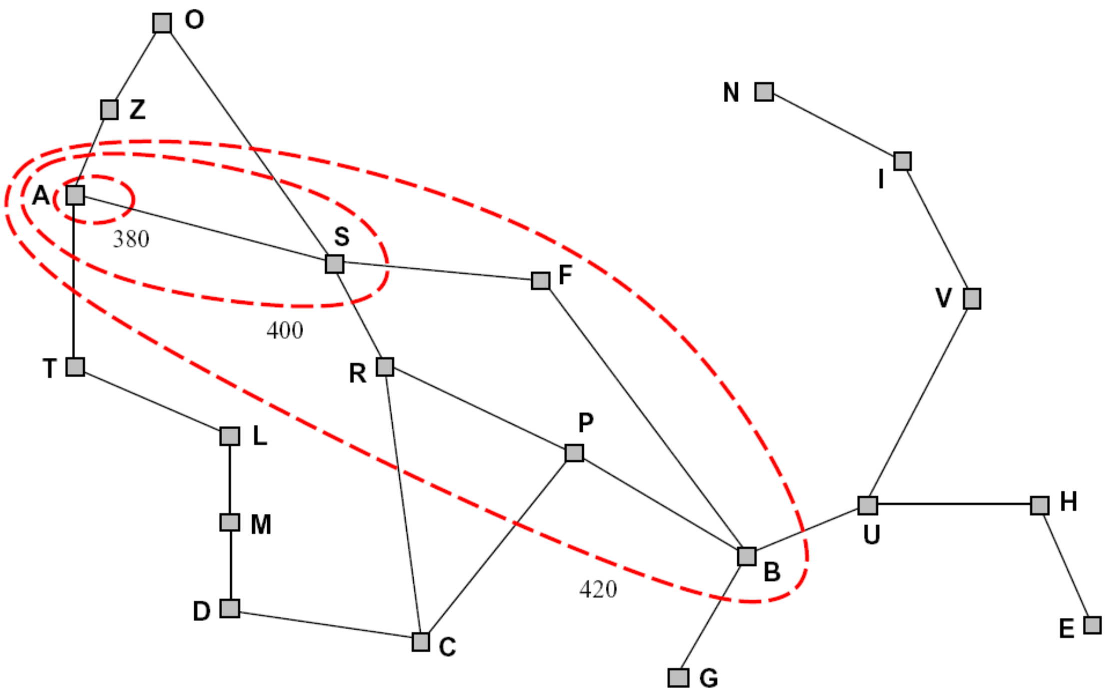
Graph Search

- Very simple fix: never expand a state twice

```
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
```

Optimality of A* Graph Search

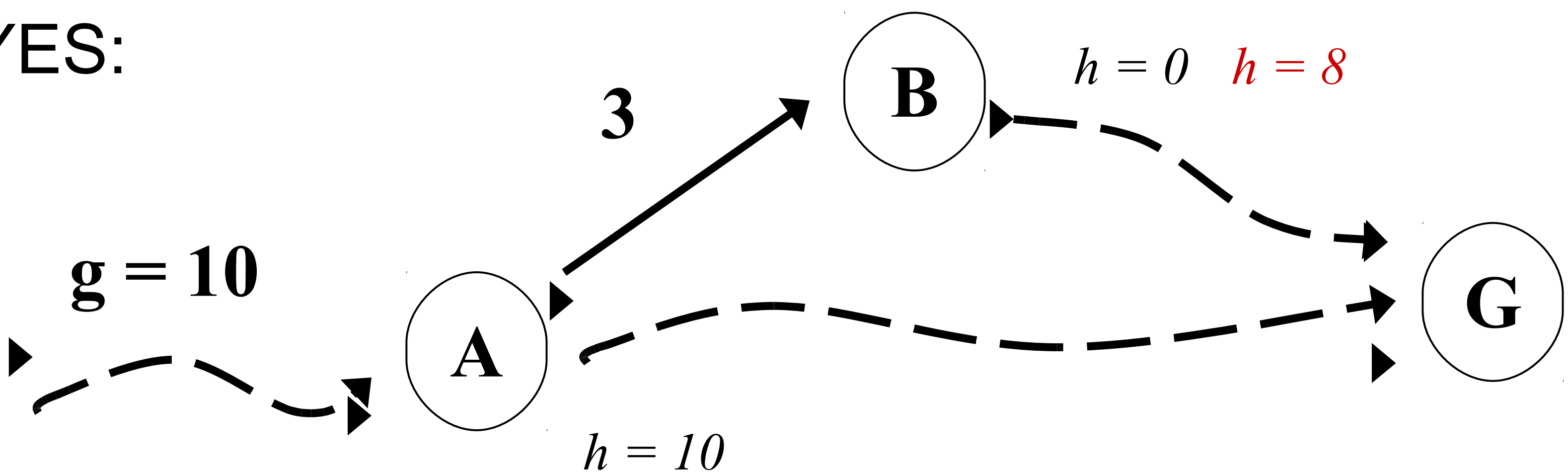
- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
  - Proof idea: optimal goals have lower f value, so get expanded first



We're making a stronger assumption than in the last proof… What?

Consistency

- Wait, how do we know we expand in increasing f value?
- Couldn't we pop some node *n*, and find its child *n'* to have lower f value?
- YES:



- What can we require to prevent these inversions?
- Consistency: $c(n, a, n') \geq h(n) - h(n')$
- Real cost must always exceed reduction in heuristic

Optimality

- Tree search:
  - A* optimal if heuristic is admissible (and non-negative)
  - UCS is a special case ($h = 0$)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal ($h = 0$ is consistent)
- In general, natural admissible heuristics tend to be consistent

Summary: A*

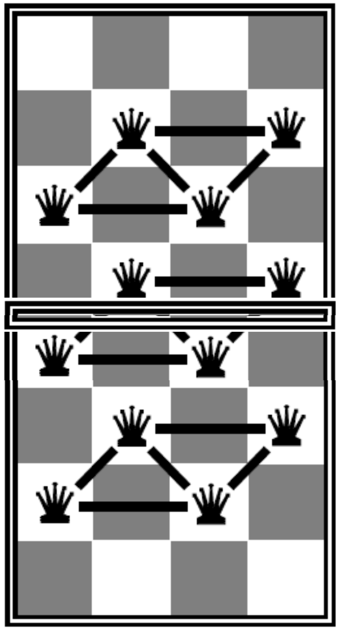
- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible heuristics
- Heuristic design is key: often use relaxed problems

Limited Memory Options

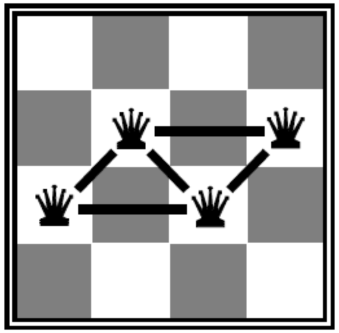
- Bottleneck: not enough memory to store entire fringe
- Hill-Climbing Search:
  - Only “best” node kept around, no fringe!
  - Usually prioritize successor choice by h (greedy hill climbing)
  - Compare to greedy backtracking, which still has fringe
- Beam Search (Limited Memory Search)
  - In between: keep K nodes in fringe
  - Dump lowest priority nodes as needed
  - Can prioritize by h alone (greedy beam search), or h+g (limited memory A*)
  - Why not applied to UCS?
  - We’ll return to beam search later…
- No guarantees once you limit the fringe size!

Types of Problems

- Planning problems:
  - We want a path to a solution (examples?)
  - Usually want an optimal path
  - *Incremental formulations*



- Identification problems:
  - We actually just want to know what the goal is (examples?)
  - Usually want an optimal goal
  - *Complete-state formulations*
  - Iterative improvement algorithms



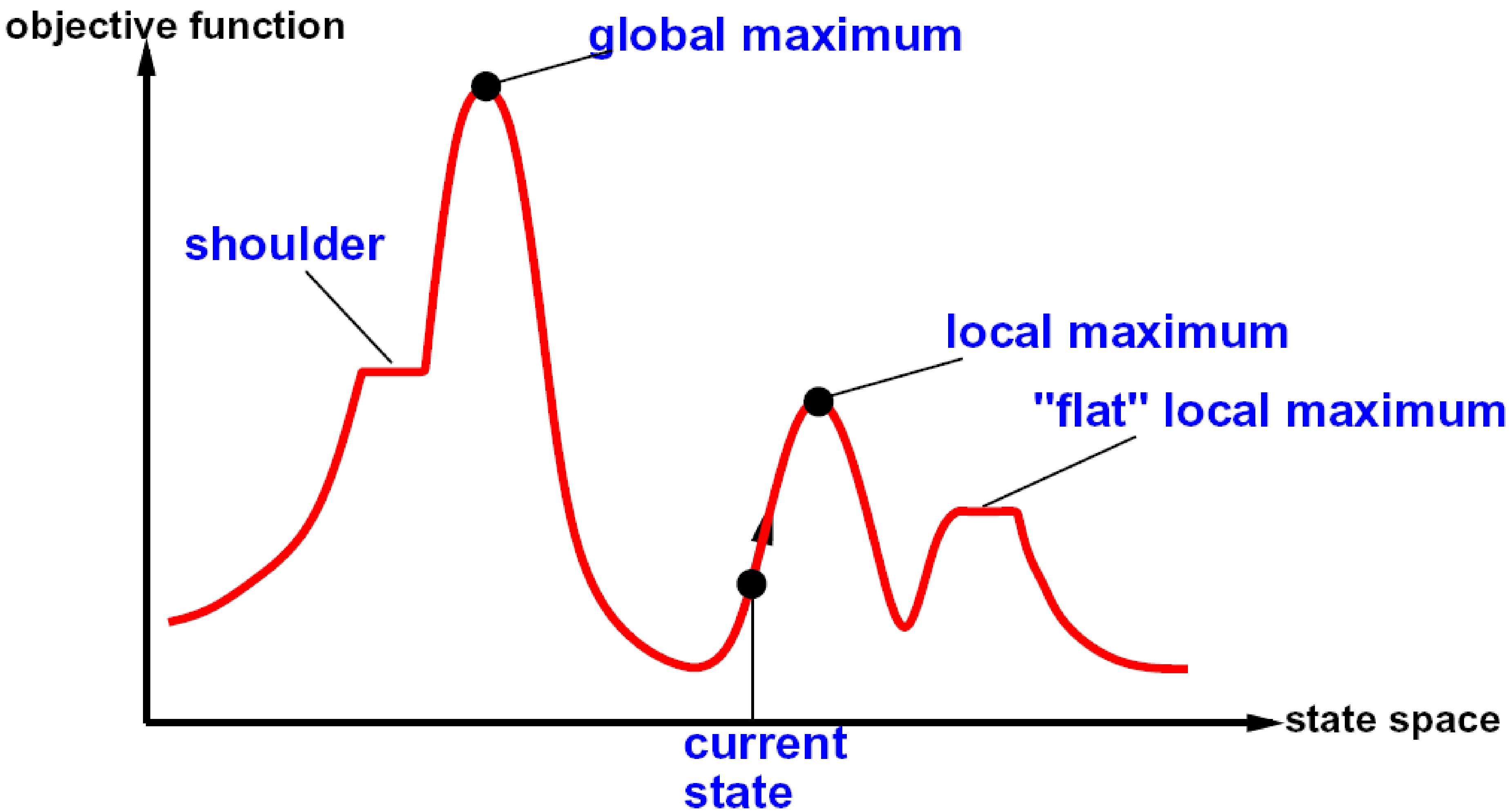
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit

- Why can this be a terrible idea?
  - Complete?
  - Optimal?

- What's good about it?

Hill Climbing Diagram



- Random restarts?
- Random sideways steps?

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

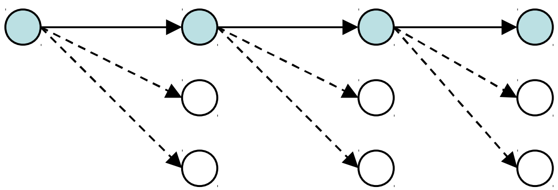
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward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

Simulated Annealing

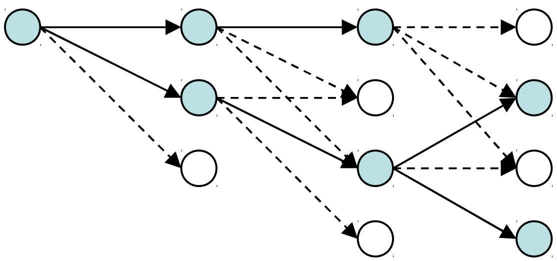
- Theoretical guarantee:
  - Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you are to every make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

Beam Search

- Like greedy search, but keep K states at all times:



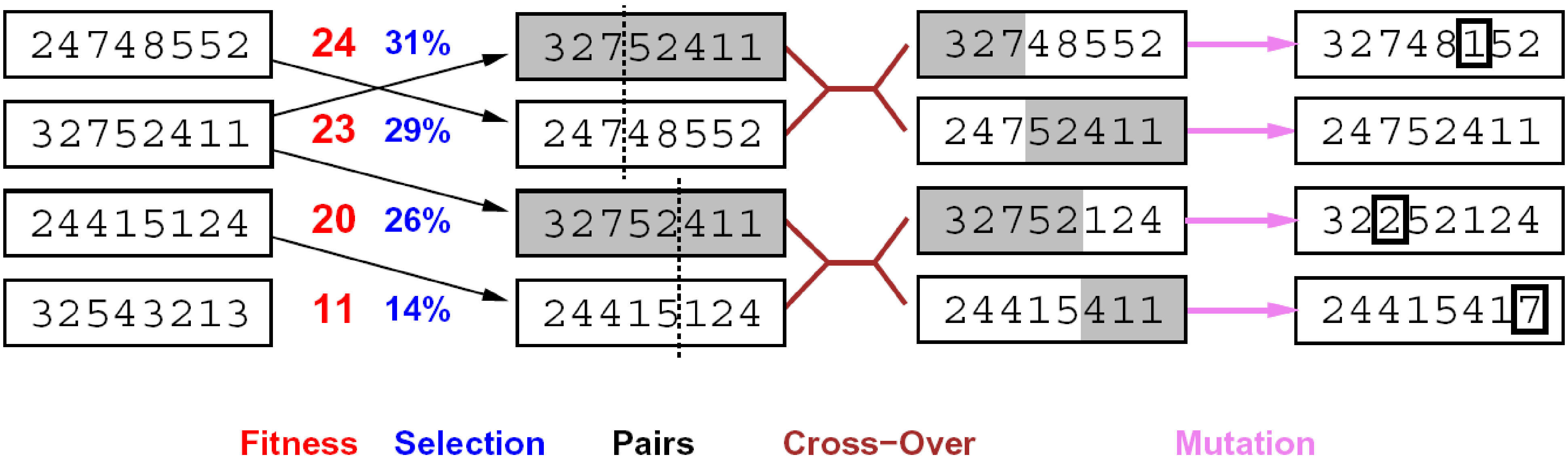
Greedy Search



Beam Search

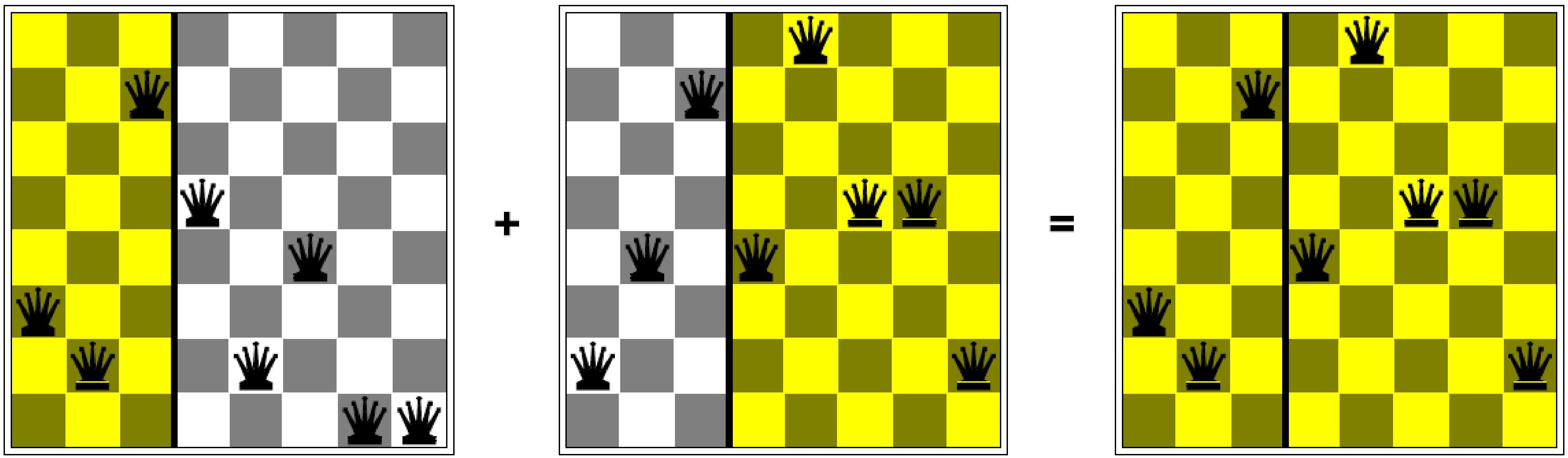
- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
- Like beam search (selection), but also have pairwise crossover operators, with optional mutation
- Probably the most misunderstood, misapplied (and even maligned) technique around!

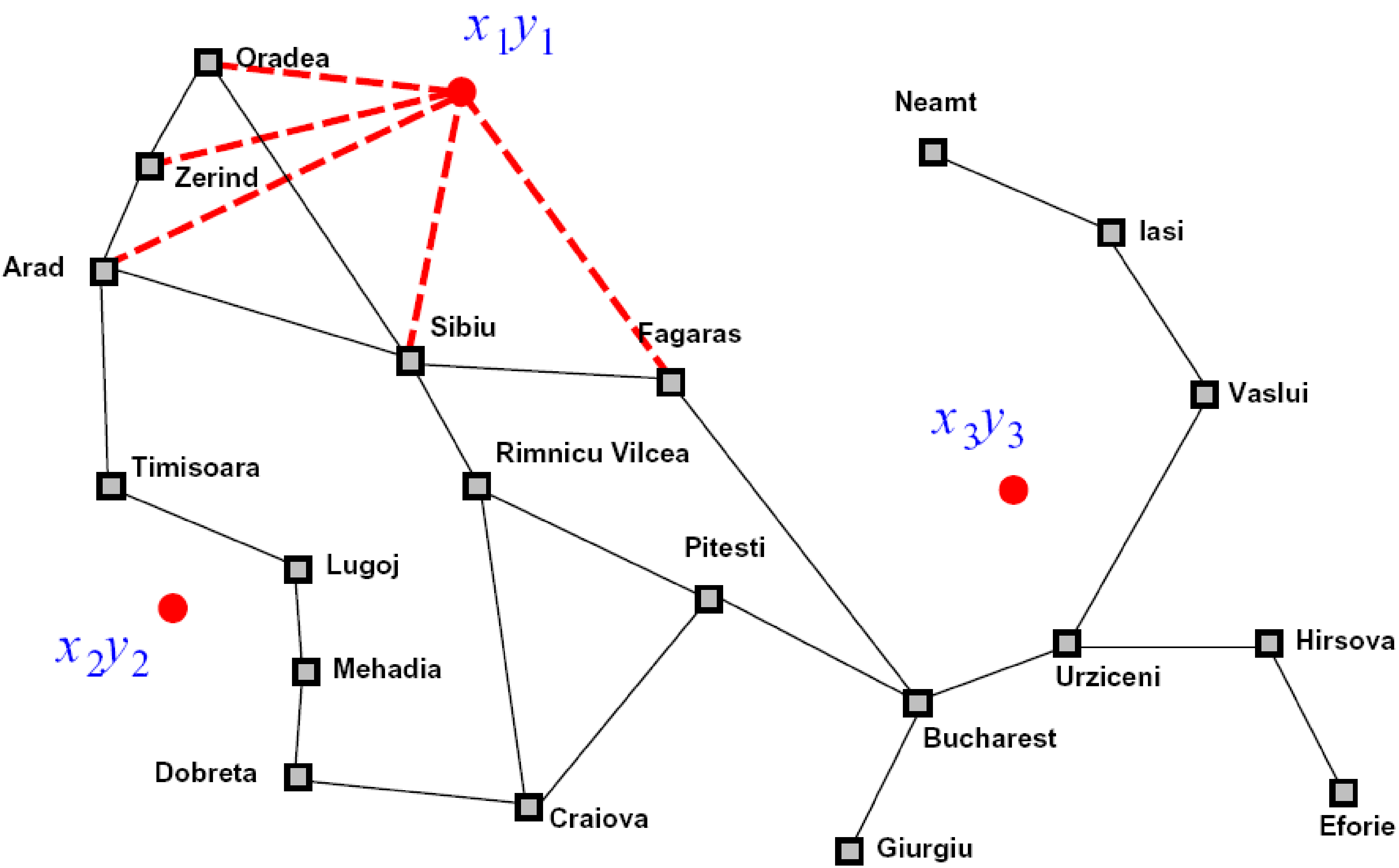
Example: N-Queens



- Why does crossover make sense here?
- When wouldn't it make sense?
- What would mutation be?
- What would a good fitness function be?

Continuous Problems

- Placing airports in Romania
  - States: $(x_1,y_1,x_2,y_2,x_3,y_3)$
  - Cost: sum of squared distances to closest city



Gradient Methods

- How to deal with continuous (therefore infinite) state spaces?
- Discretization: bucket ranges of values
  - E.g. force integral coordinates
- Continuous optimization
  - E.g. gradient ascent

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

$$x \leftarrow x + \alpha \nabla f(x)$$

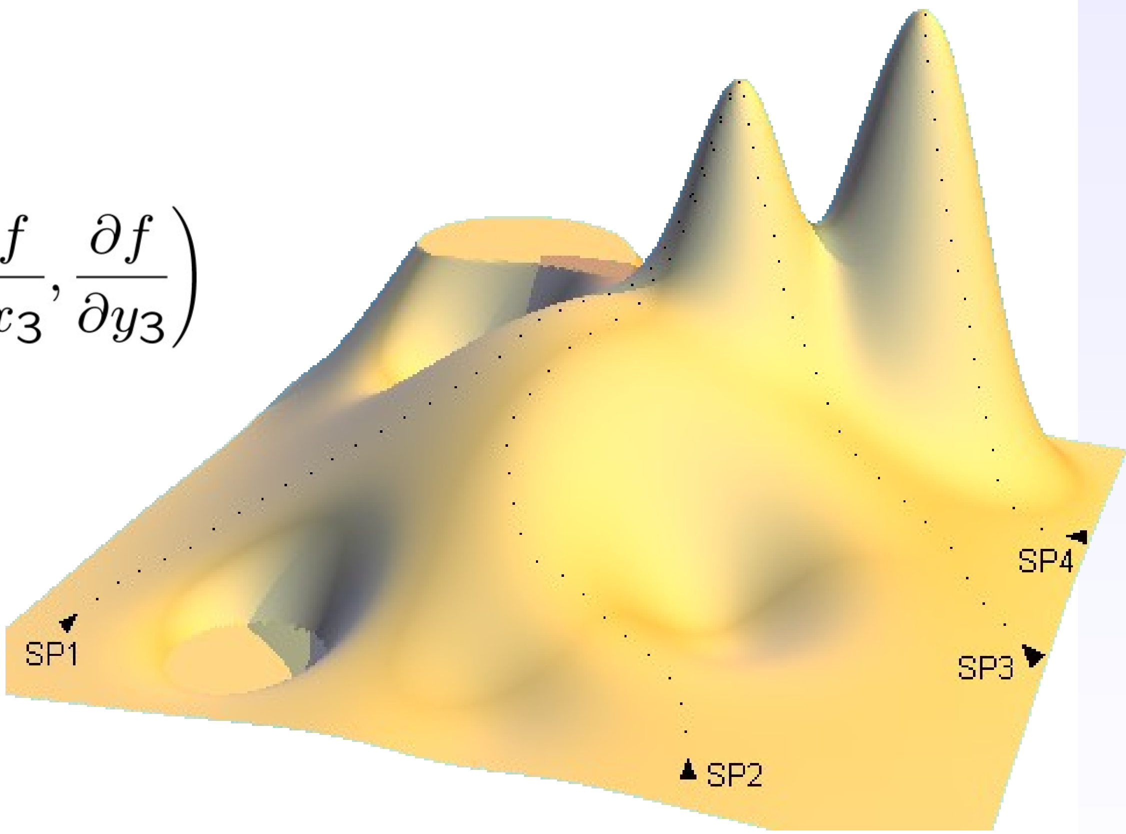


Image from vias.org