CS 188: Artificial Intelligence
Spring 2007

Lecture 4: A* Search

Srini Narayanan – ICSI and UC Berkeley

Many slides over the course adapted from Dan Klein, Stuart Russell and Andrew Moore

Announcements

- Submission of Assignment 1
  - Submit program should be updated by today
  - Use submit hw1 for this assignment
- Enrollment issues

Today

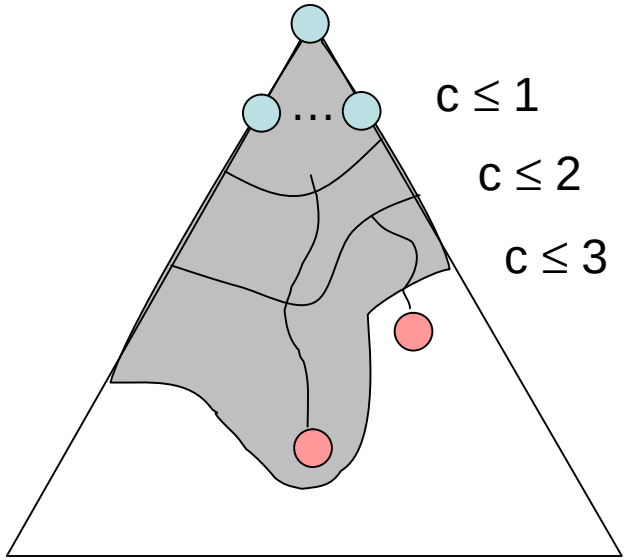
- A* Search
- Heuristic Design
- Local Search

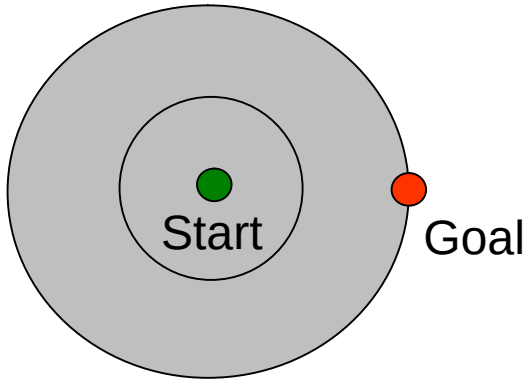
Recap: Search

- Search problems:
  - States, successors, costs, start and goal tests
- Search trees:
  - Nodes: represent paths, have costs
  - Strategies differing fringe management
- Tree vs. graph search

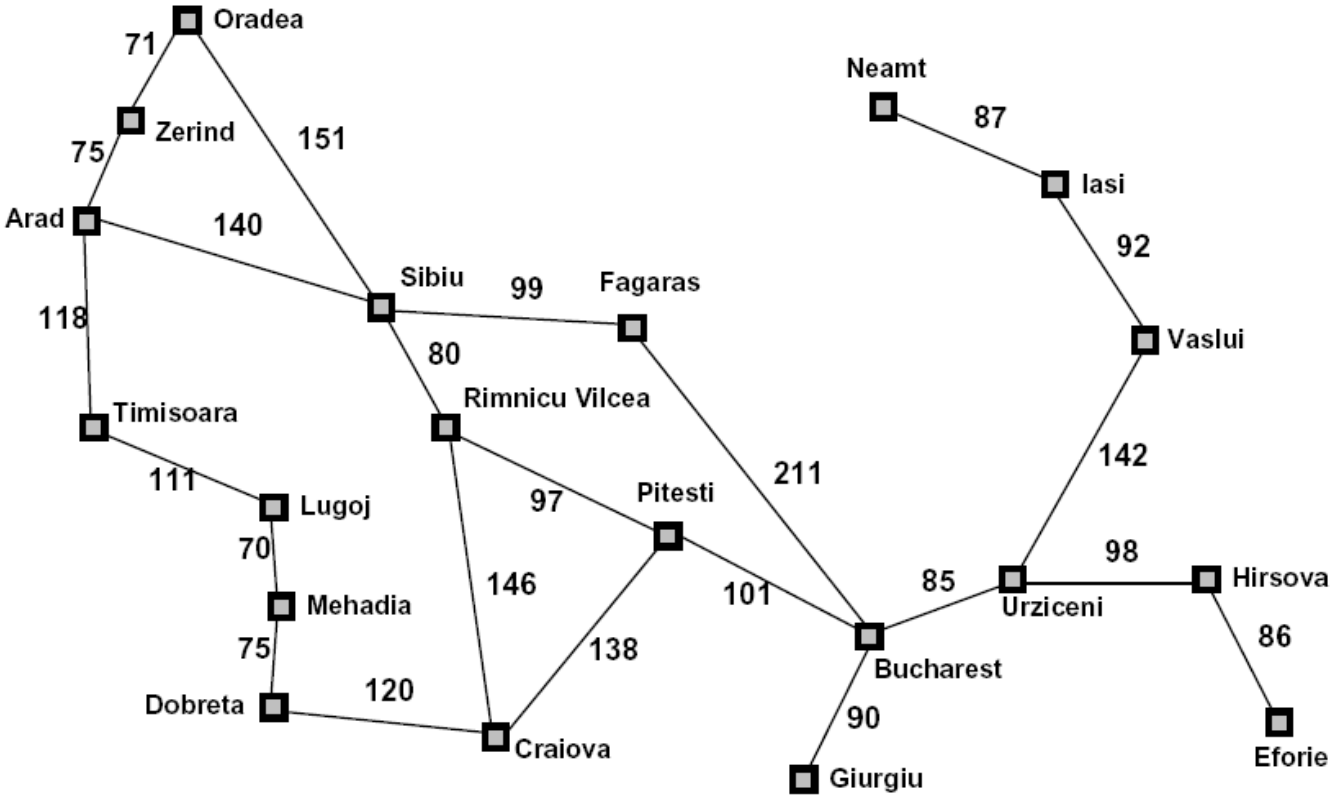
Uniform Cost: Problems

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location





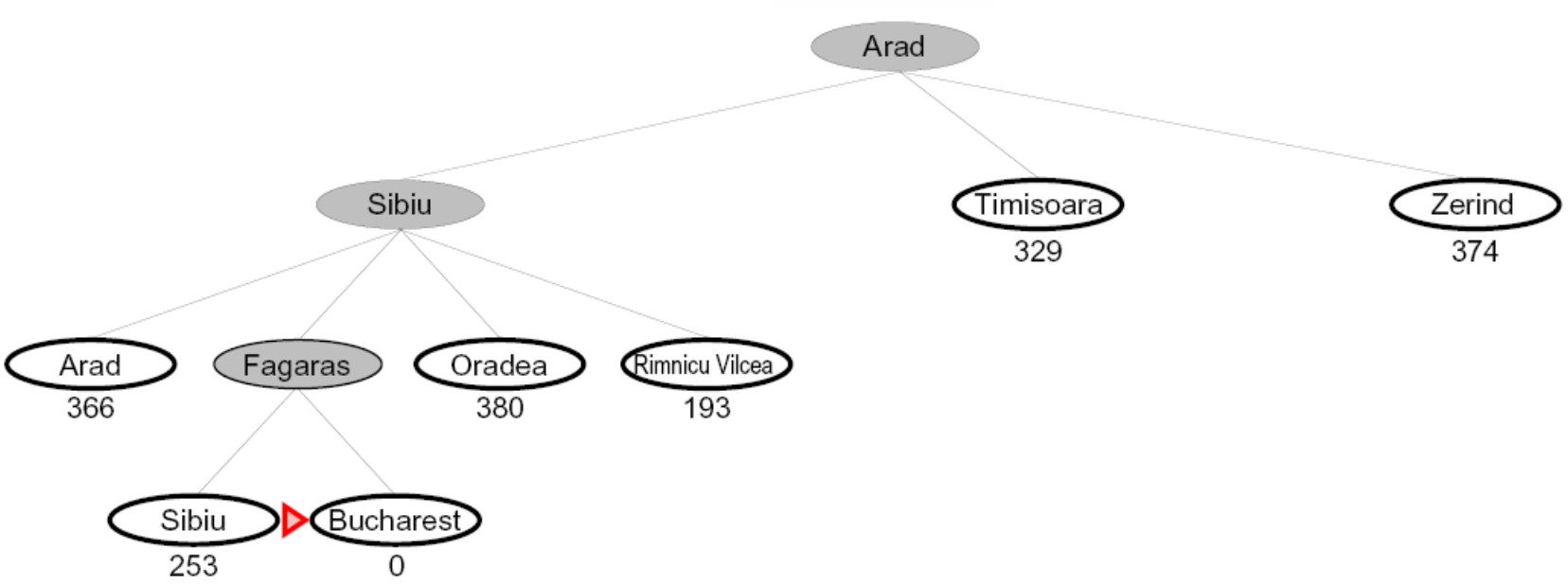
Best-First / Greedy Search



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

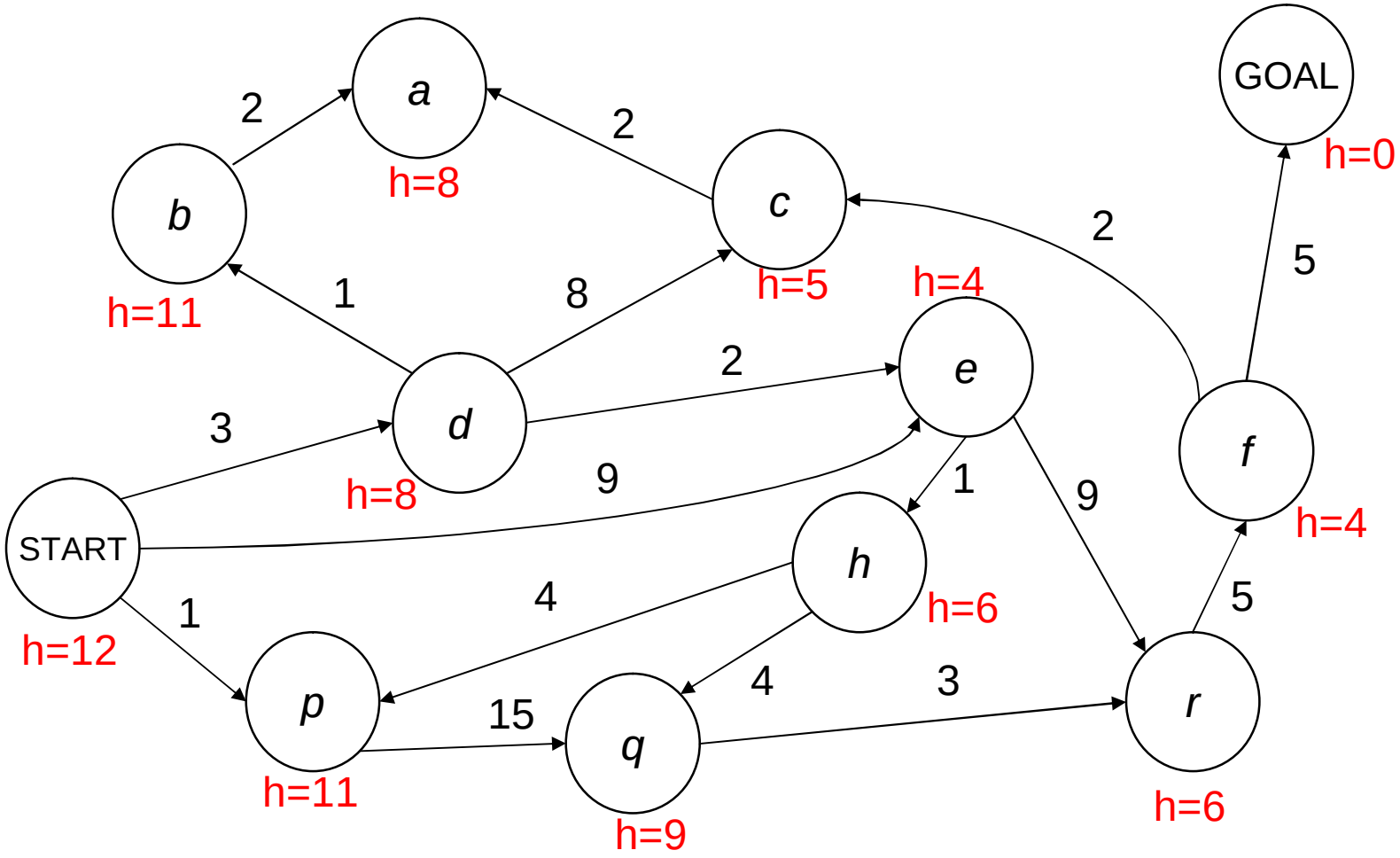
Best-First / Greedy Search

- Expand the node that seems closest…



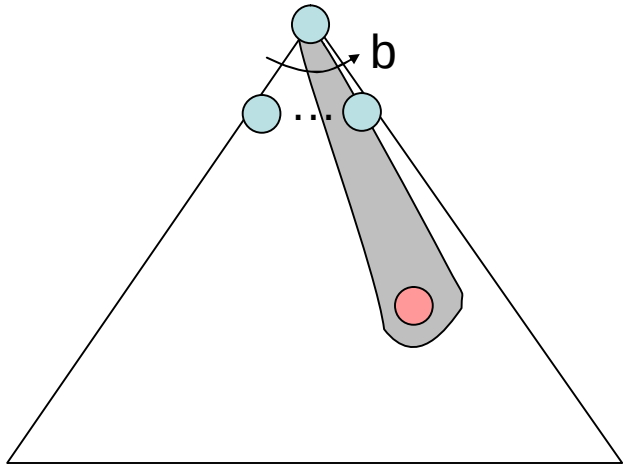
- What can go wrong?

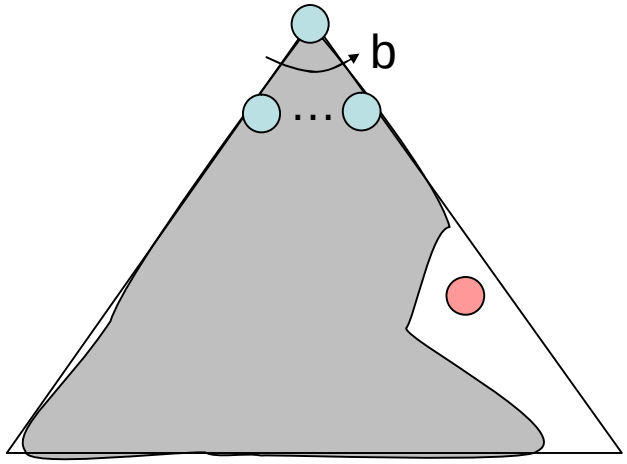
Best-First / Greedy Search



Best-First / Greedy Search

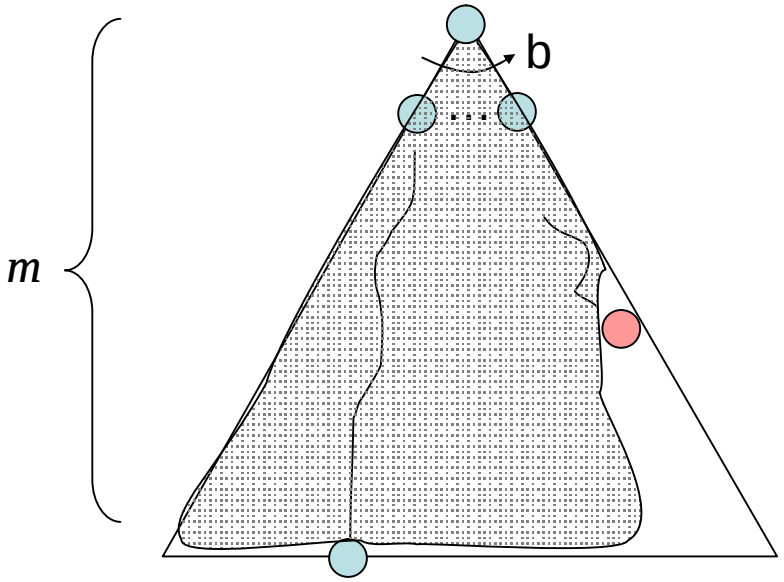
- A common case:
  - Best-first takes you straight to the goal on a wrong path
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)





Best First Greedy Search

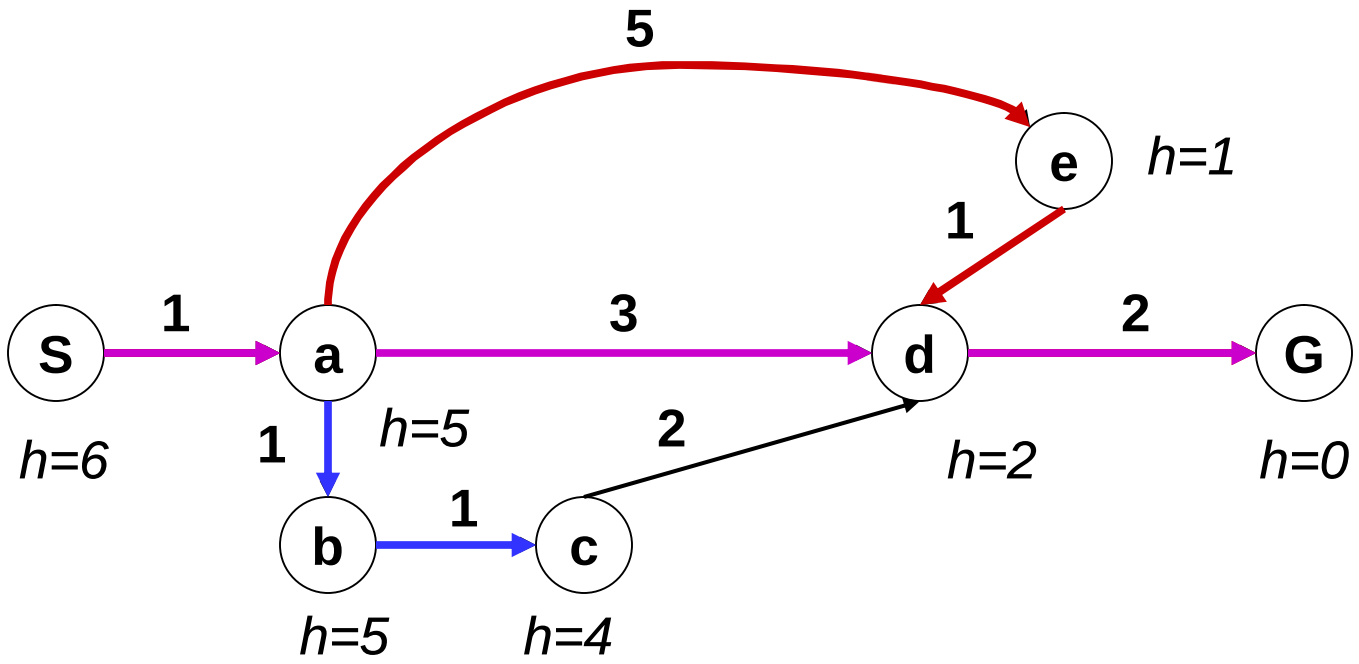
Algorithm	Complete	Optimal	Time	Space
Greedy Best-First Search	Y*	N	$O(b^m)$	$O(b^m)$



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

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)

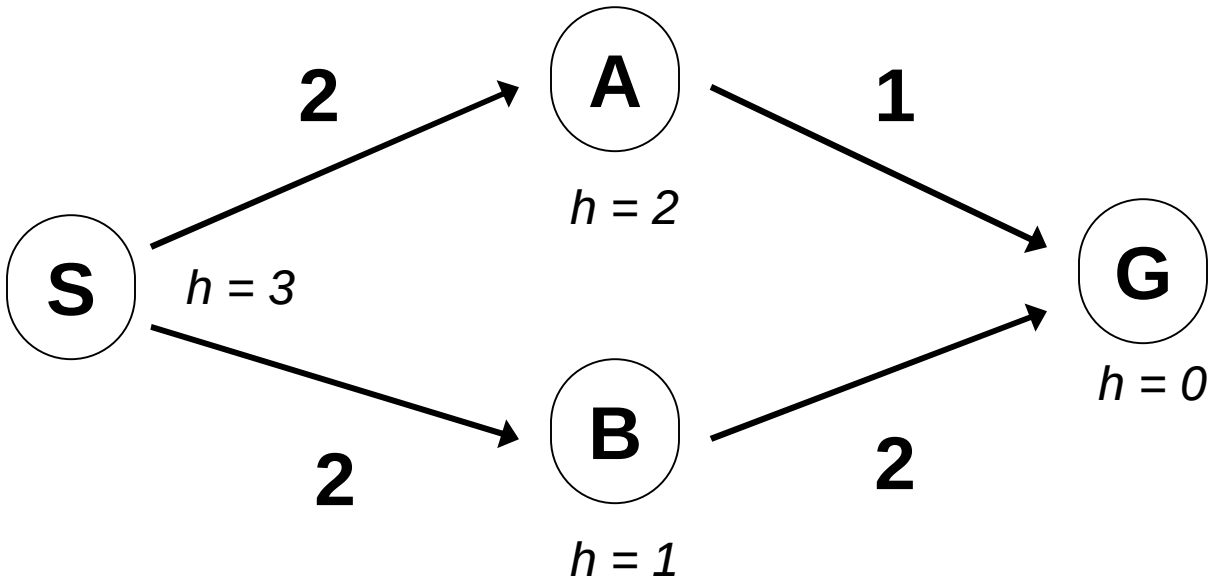


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

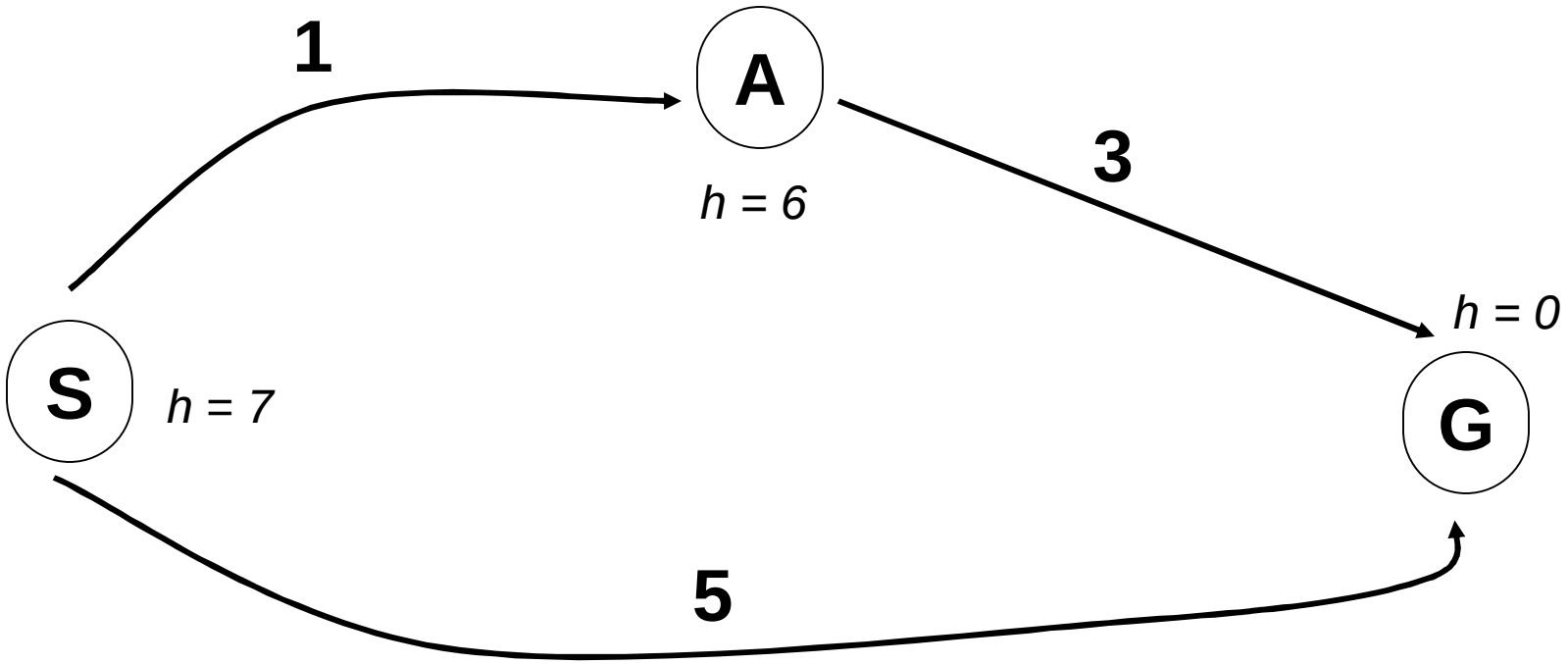
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



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

Admissible Heuristics

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

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

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

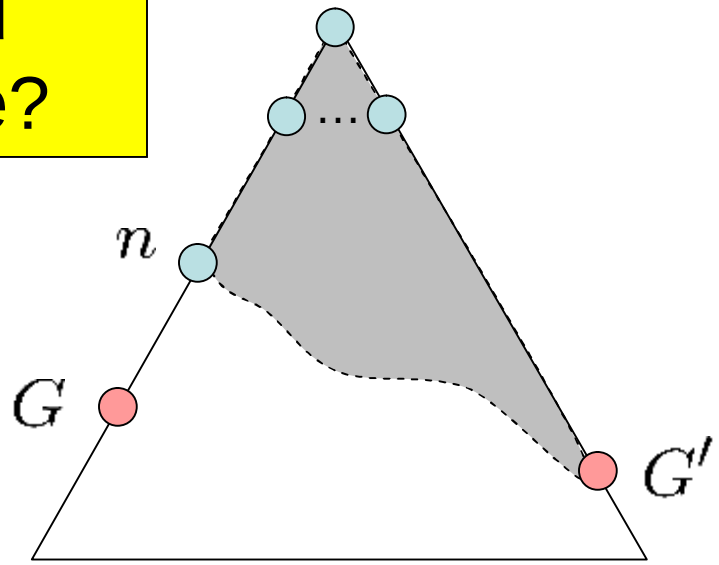
- E.g. Euclidean distance on a map problem

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

Optimality of A*: Blocking

This proof assumed tree search! Where?

- Proof:
  - What could go wrong?
  - We'd have to have to pop a suboptimal goal off the fringe queue
  - This can't happen:
    - Imagine a suboptimal goal G' is on the queue
    - Consider any unexpanded (fringe) node *n* on a shortest path to optimal G
    - *n* will be popped before G



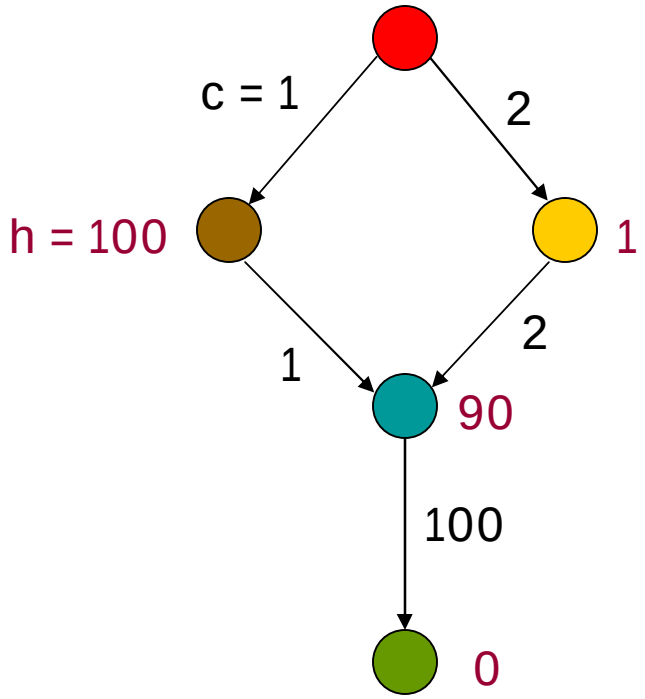
$$f(n) \leq g(G)$$

$$g(G) < g(G')$$

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

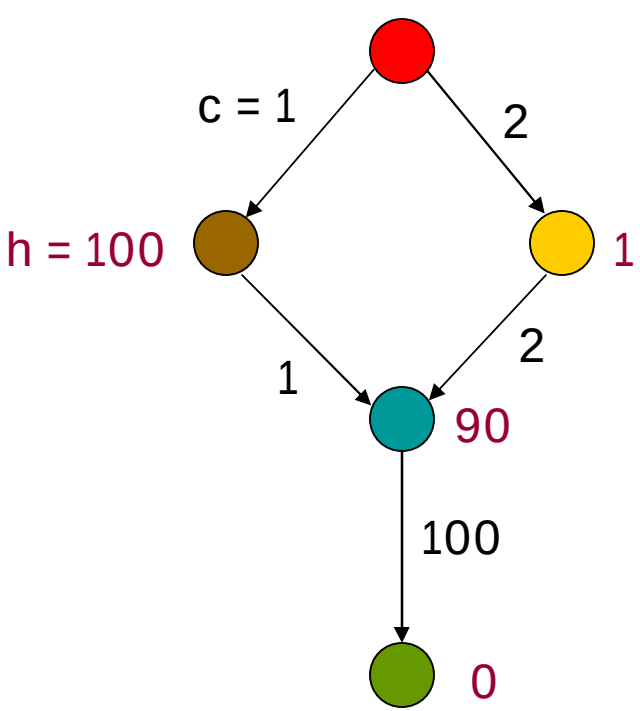
$$f(n) < f(G')$$

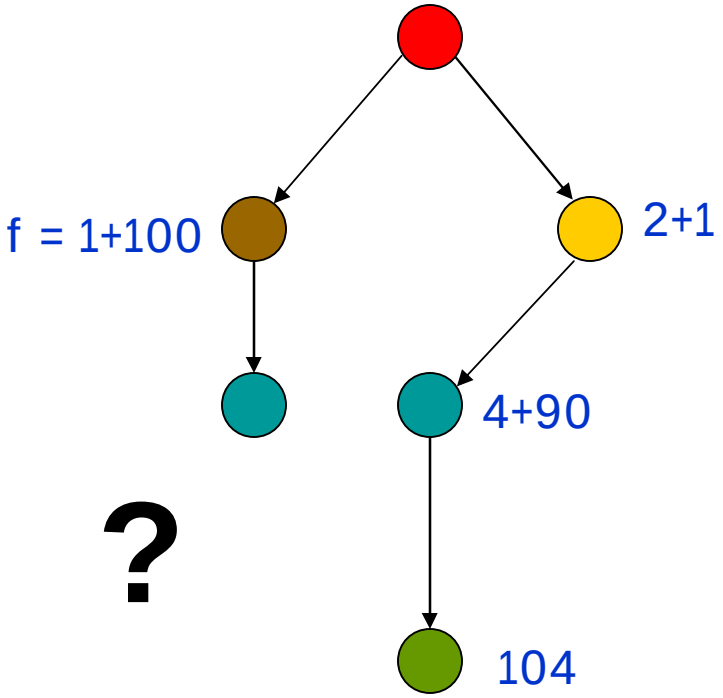
What to do with revisited states?



The heuristic h is clearly admissible

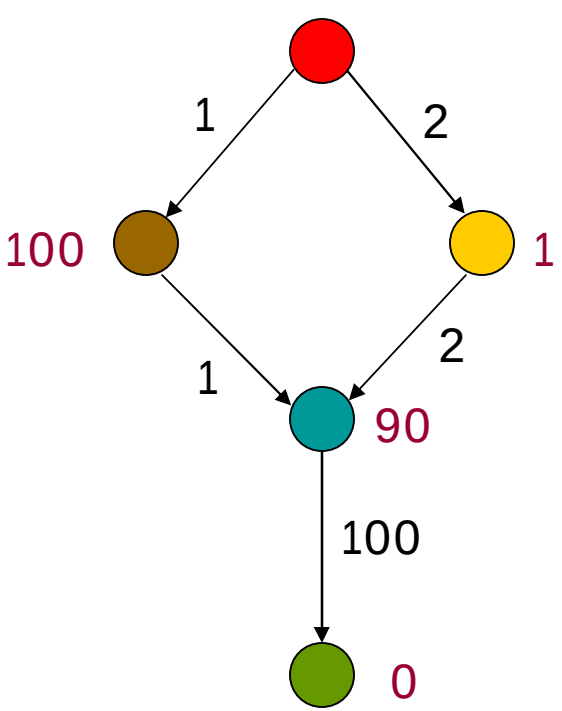
What to do with revisited states?

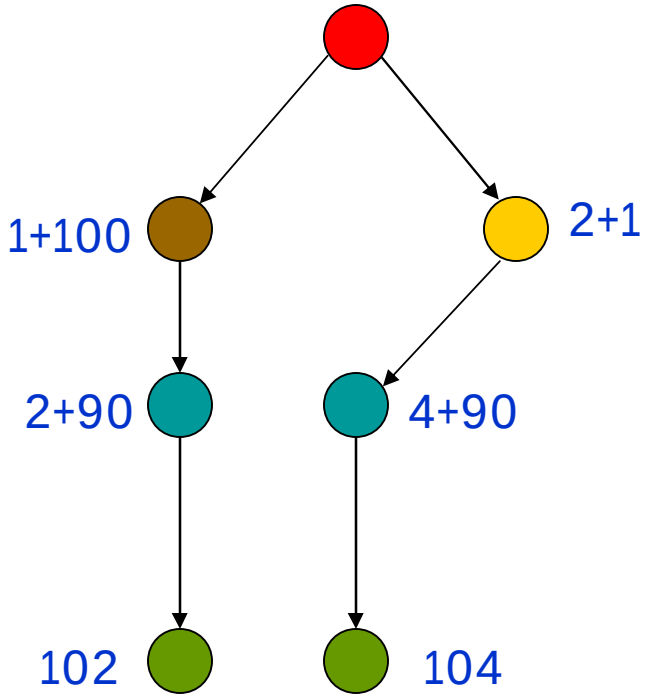




If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution

What to do with revisited states?

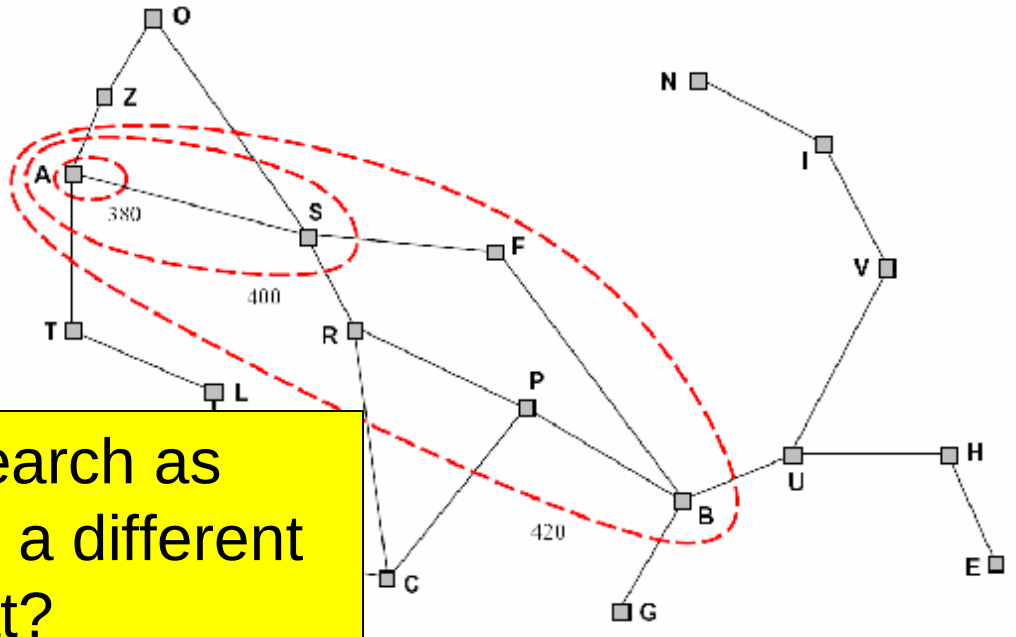




Instead, if we do not discard nodes revisiting states, the search terminates with an optimal solution

Optimality of A*: Contours

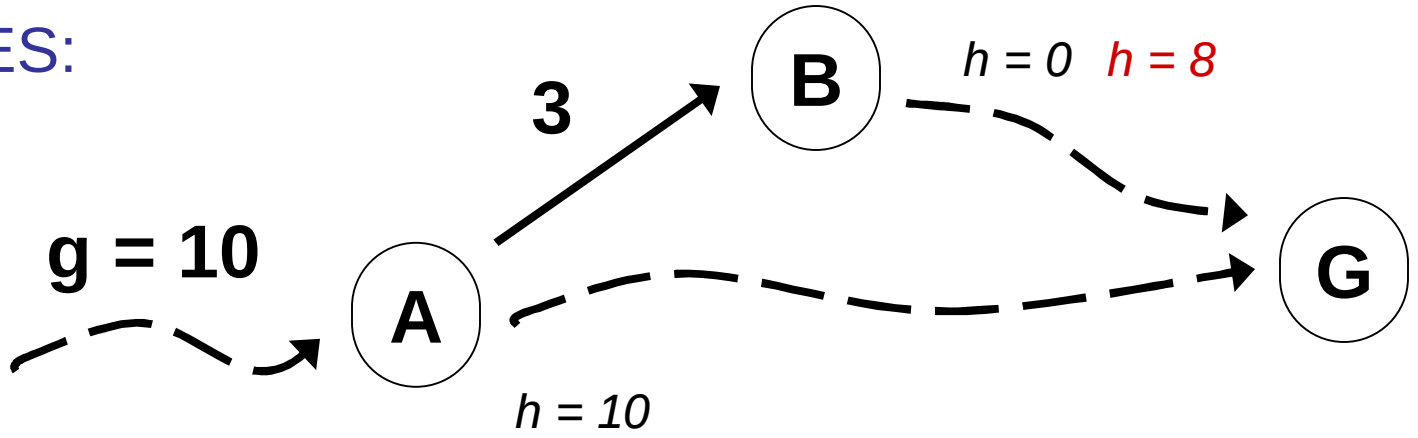
- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
  - Proof idea: optimal goals have lower f value, so get expanded first



Holds for graph search as well, but we made a different assumption. 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:



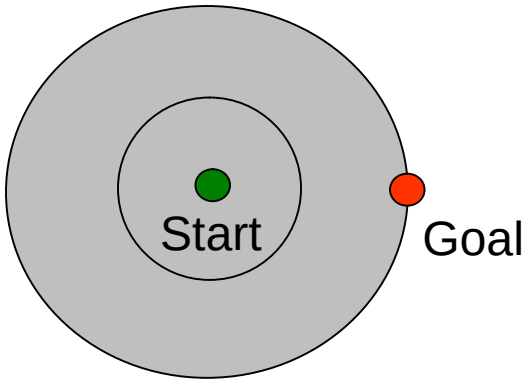
- What do we need to do to fix this?
- Consistency: $c(n, a, n') \geq h(n) - h(n')$
- Real cost always exceeds reduction in heuristic

Admissibility and Consistency

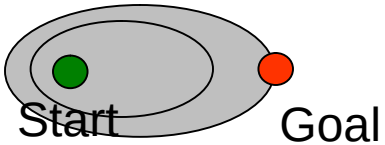
- A consistent heuristic is also admissible
  [Left as an exercise]
- An admissible heuristic may not be consistent, but many admissible heuristics are consistent

UCS vs A* Contours

- Uniform-cost expanded in all directions

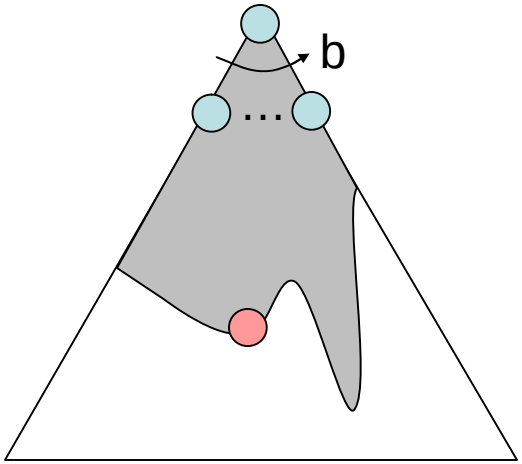


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

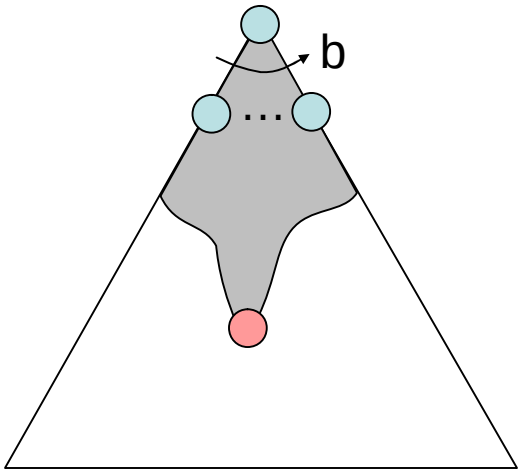


Properties of A*

Uniform-Cost



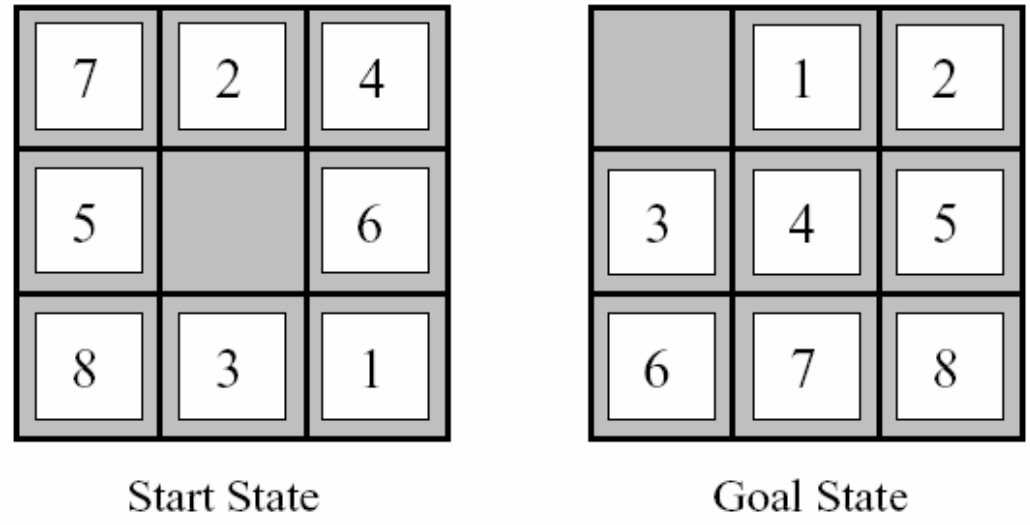
A*



Admissible Heuristics

- Most of the work is in coming up with admissible heuristics
- Good news: usually admissible heuristics are also consistent
- More good news: inadmissible heuristics are often quite effective (especially when you have no choice)
- Very common hack: use α x h(n) for admissible h, $\alpha > 1$ to generate a faster but less optimal inadmissible h'

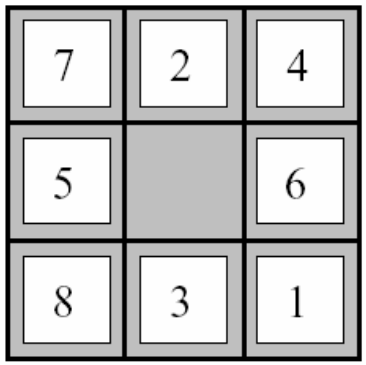
Example: 8-Puzzle



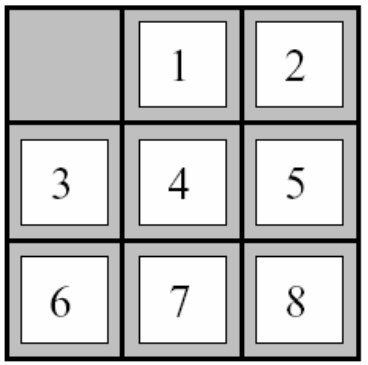
- What are the states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

8-Puzzle I

- Number of tiles misplaced?
- Why is it admissible?
- h(start) = 8
- 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

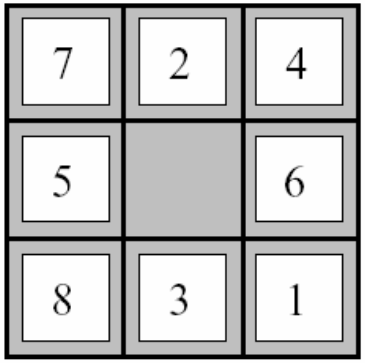
8-Puzzle II

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

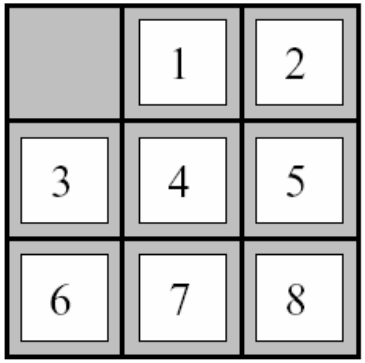
- h(start) =

  3 + 1 + 2 + …

  = 18



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?
  - What's wrong with it?

- With A*, trade-off between quality of estimate and work per node!

Trivial Heuristics, Dominance

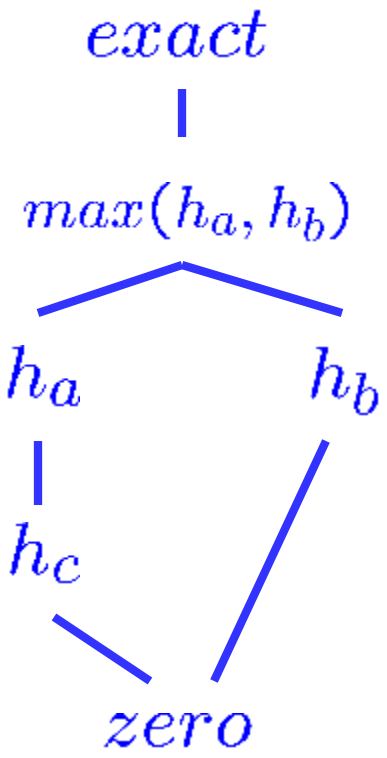
- Dominance:

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

- 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



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?

Other A* Applications

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

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

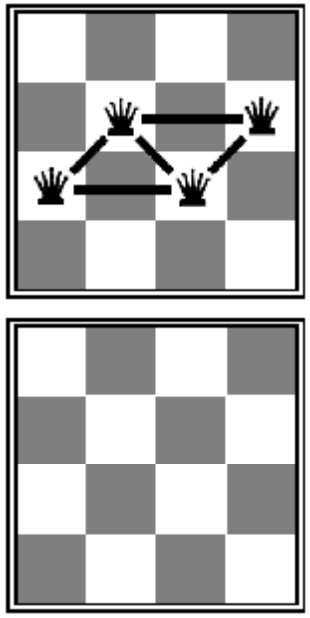
On Completeness and Optimality

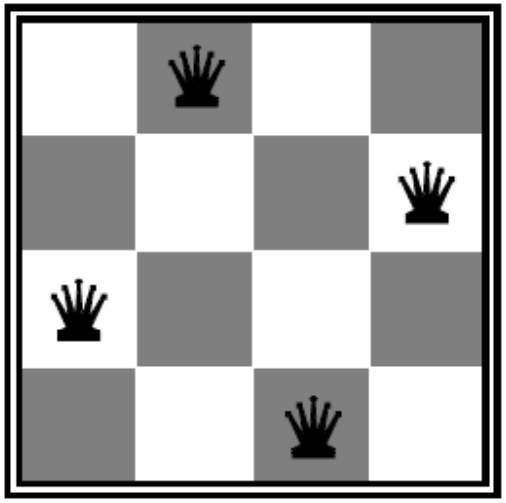
- A* with a consistent heuristic function has nice properties: completeness, optimality, no need to revisit states
- Theoretical completeness does not mean "practical" completeness if you must wait too long to get a solution (time limit issue)
- So, if one can't design an accurate consistent heuristic, it may be better to settle for a non-admissible heuristic that "works well in practice", even through completeness and optimality are no longer guaranteed

Local Search Methods

- Queue-based algorithms keep fallback options (backtracking)
- Local search: improve what you have until you can't make it better
- Generally much more efficient (but incomplete)

Example: N-Queens

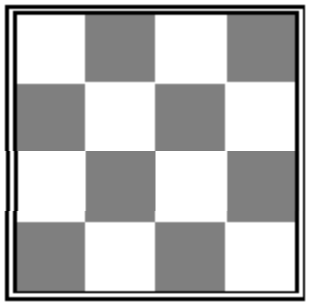




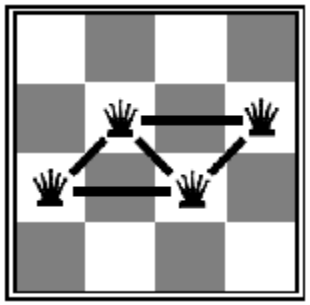
- What are the states?
- What is the start?
- What is the goal?
- What are the actions?
- What should the costs be?

Types of Problems

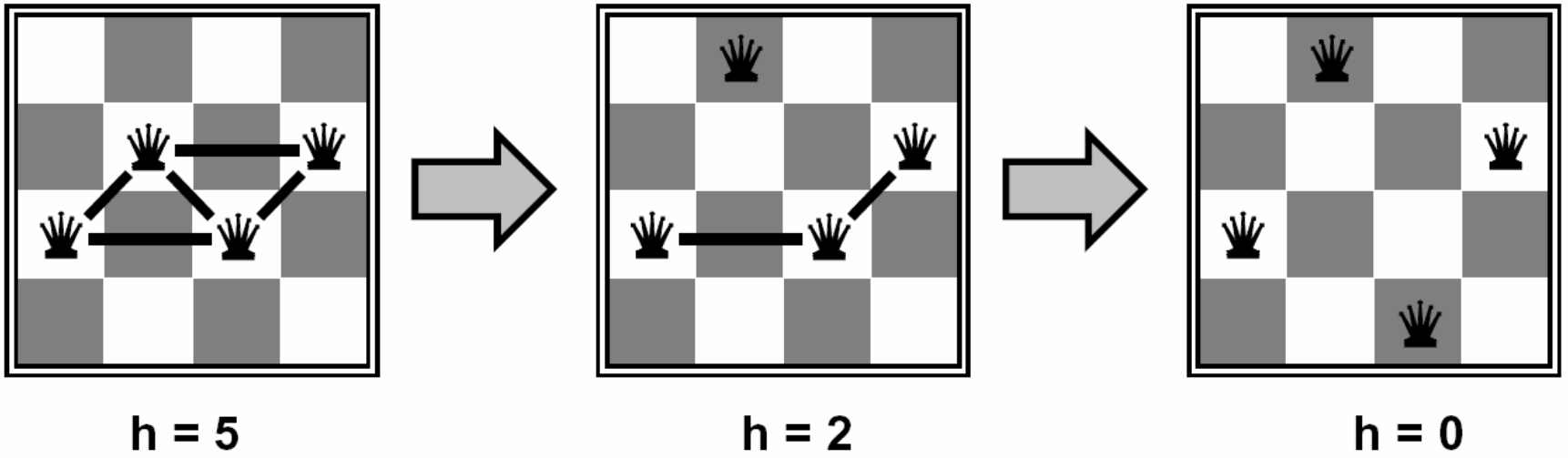
- Planning problems:
  - We want a path to a solution (examples?)
  - Usually want an optimal path
  - Incremental formulations



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



Example: N-Queens

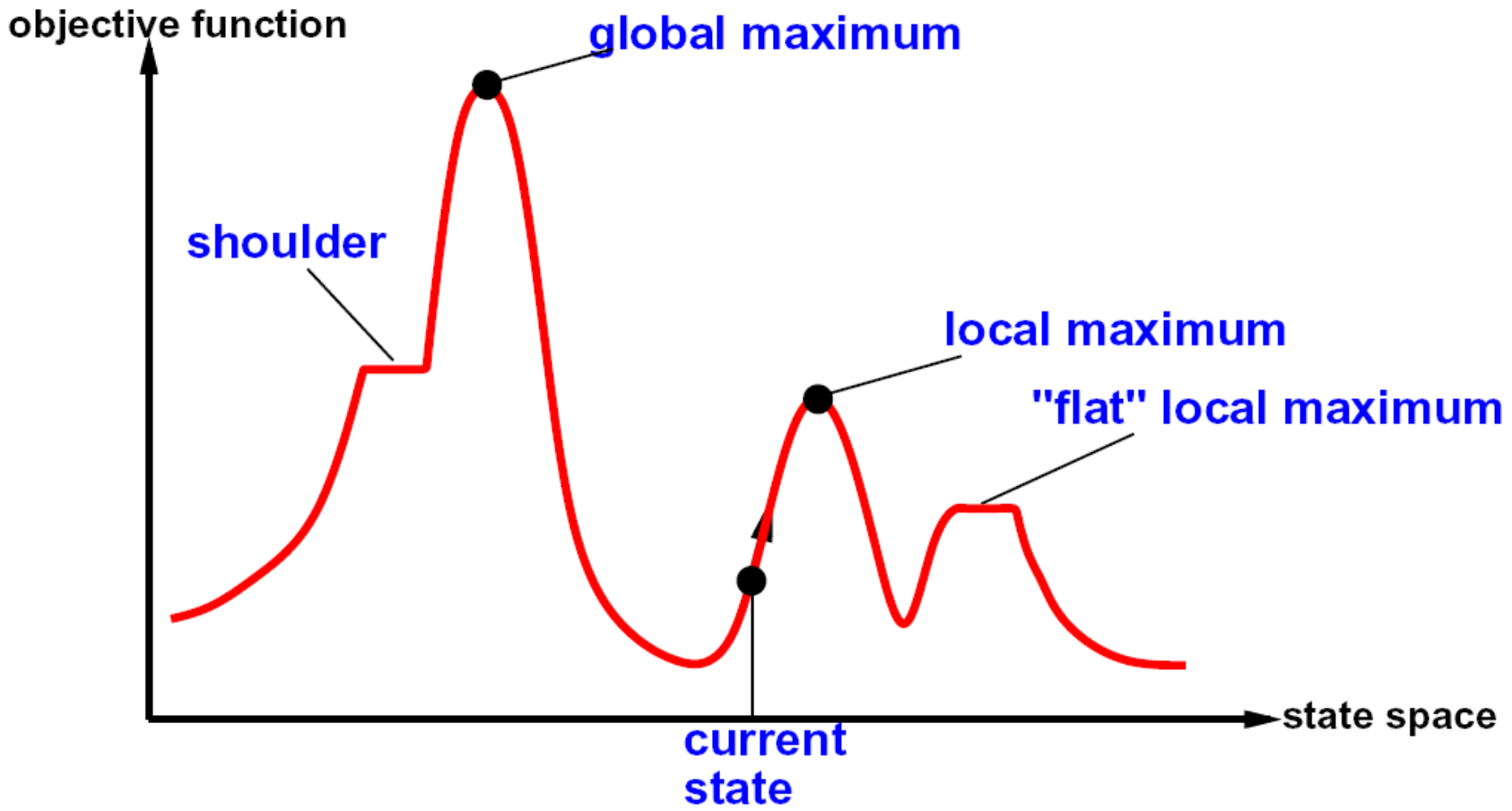


- Start wherever, move queens to reduce conflicts
- Almost always solves large n-queens nearly instantly
- How is this different from best-first search?

Hill Climbing

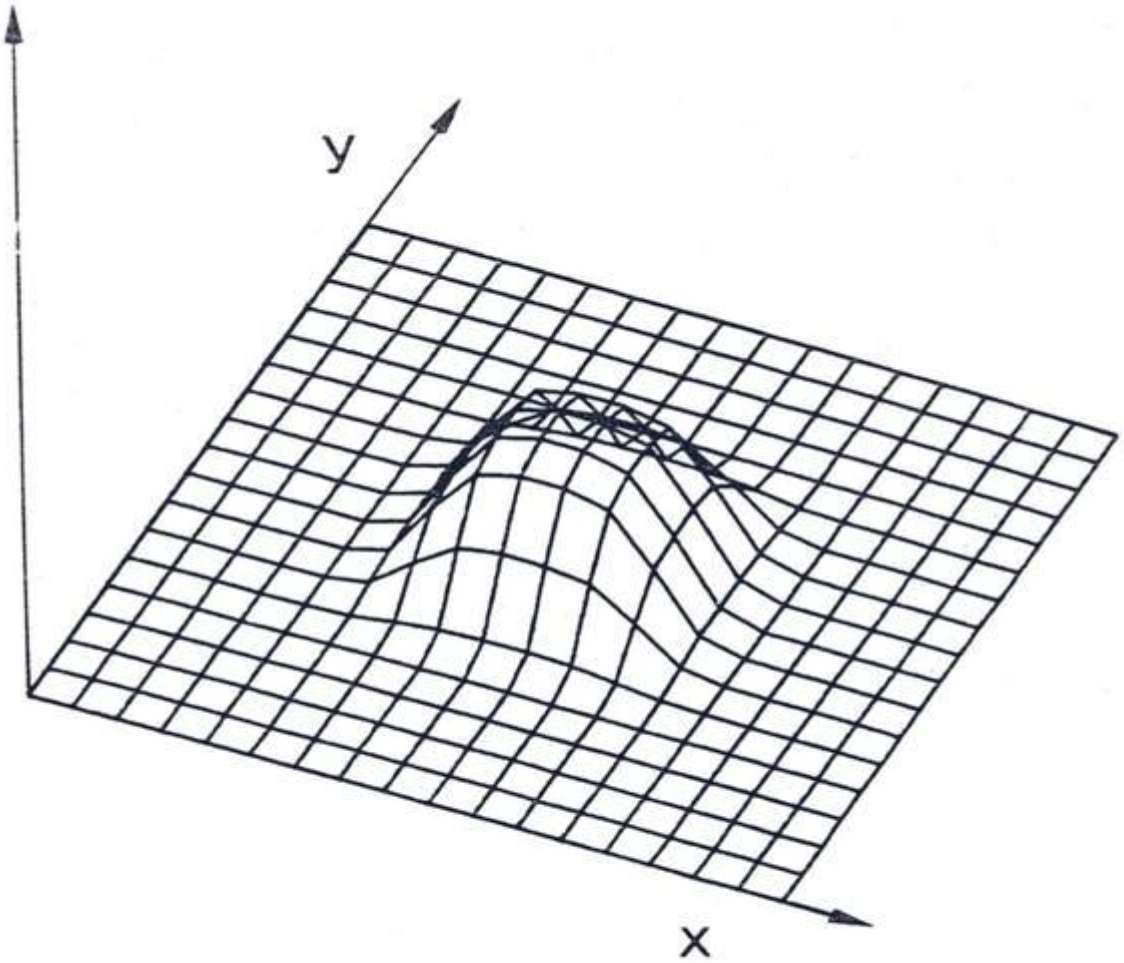
- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What's good about it?

Hill Climbing Diagram



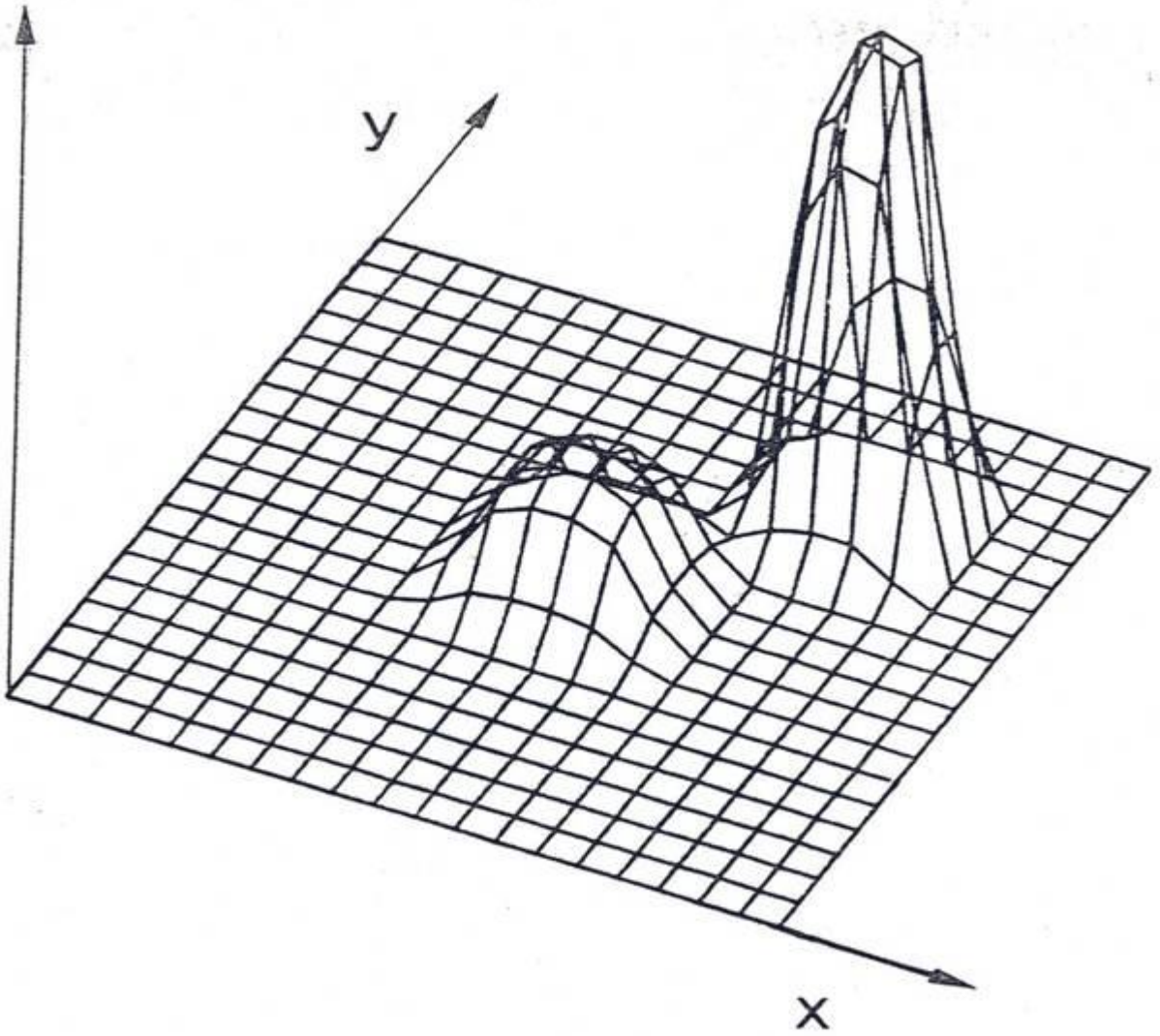
- Random restarts?
- Random sideways steps?

The Shape of an Easy Problem

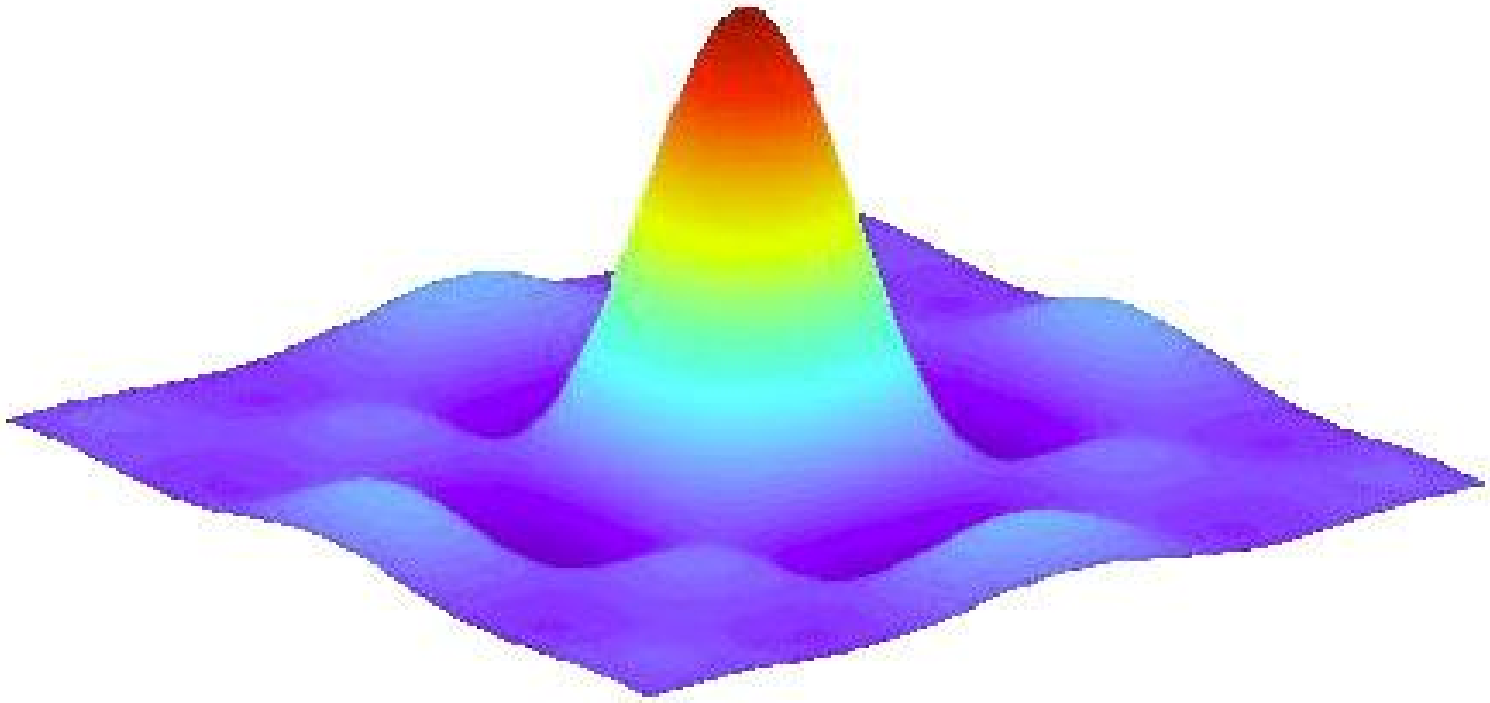


This and next several slides from Goldberg '89

The Shape of a Harder Problem



The Shape of a Yet Harder Problem



Remedies to drawbacks of hill climbing

- Random restart
- Problem reformulation
- In the end: Some problem spaces are great for hill climbing and others are terrible.

Monte Carlo Descent

1) S ß initial state
2) Repeat k times:
   a) If GOAL?(S) then return S
   b) S' ß successor of S picked at random
   c) if h(S') ≤ h(S) then S ß S'
   d) else
      - $\Delta h = h(S')-h(S)$
      - with probability ~ $\exp(-\Delta h/T)$, where T is called the "temperature" S ß S' [Metropolis criterion]
3) Return failure

Simulated annealing lowers T over the k iterations.

It starts with a large T and slowly decreases T

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

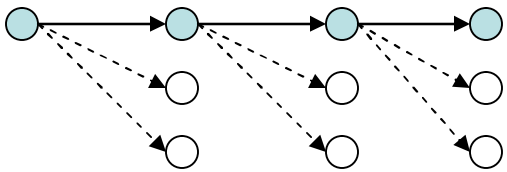
  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then**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

Simulated Annealing

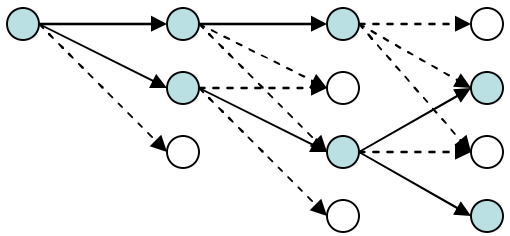
- Theoretical guarantee:
  - Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you 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:



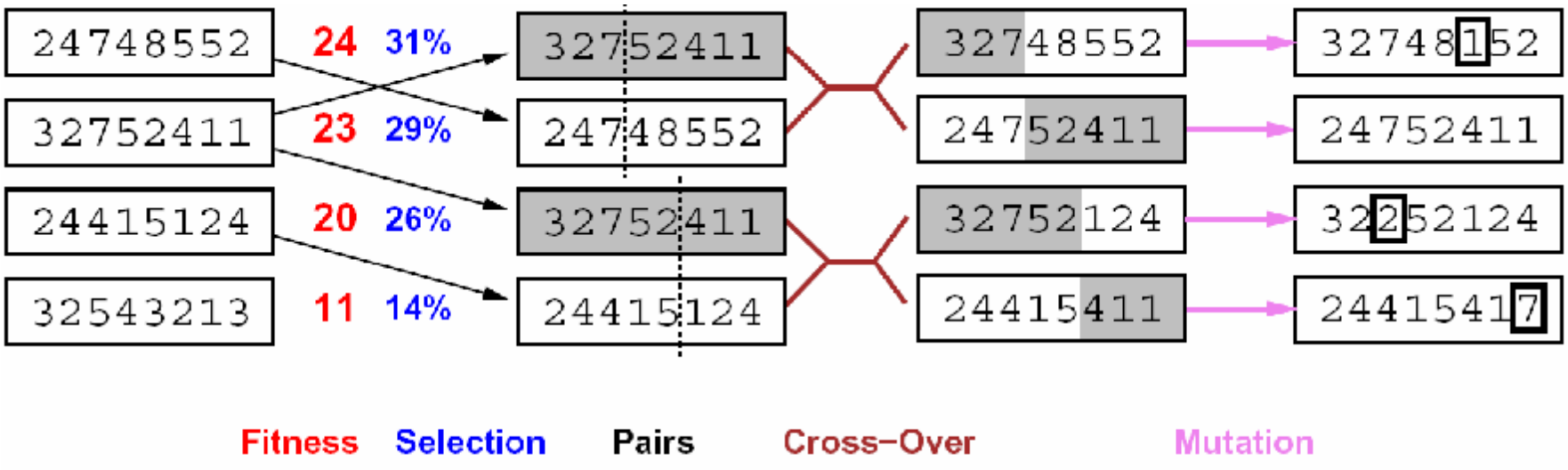
Greedy Search



Beam Search

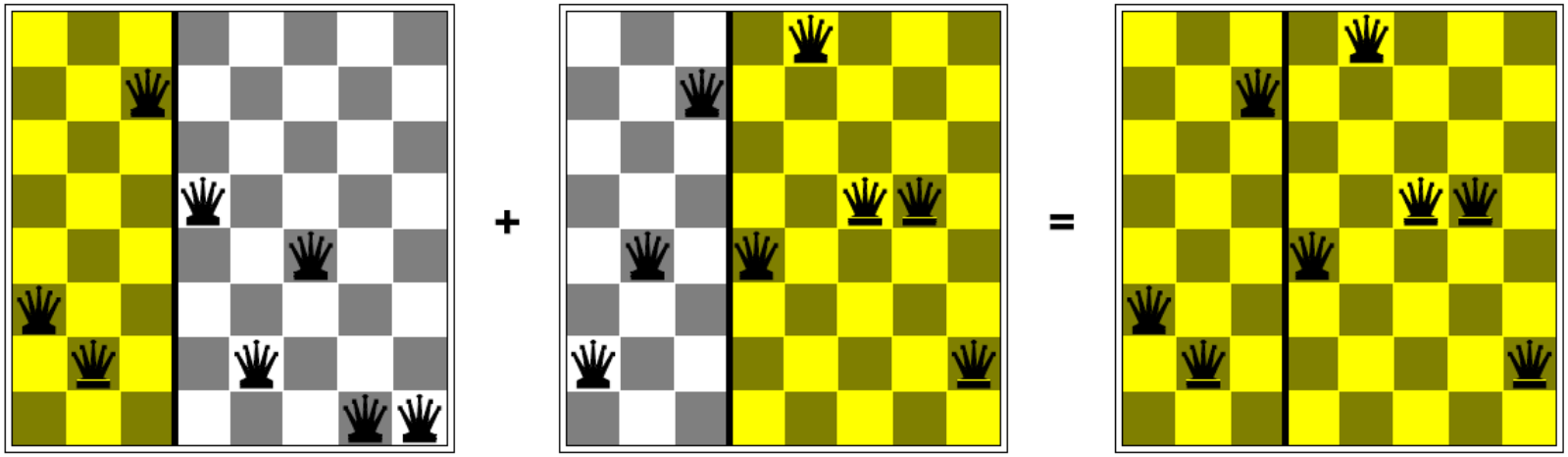
- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

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!

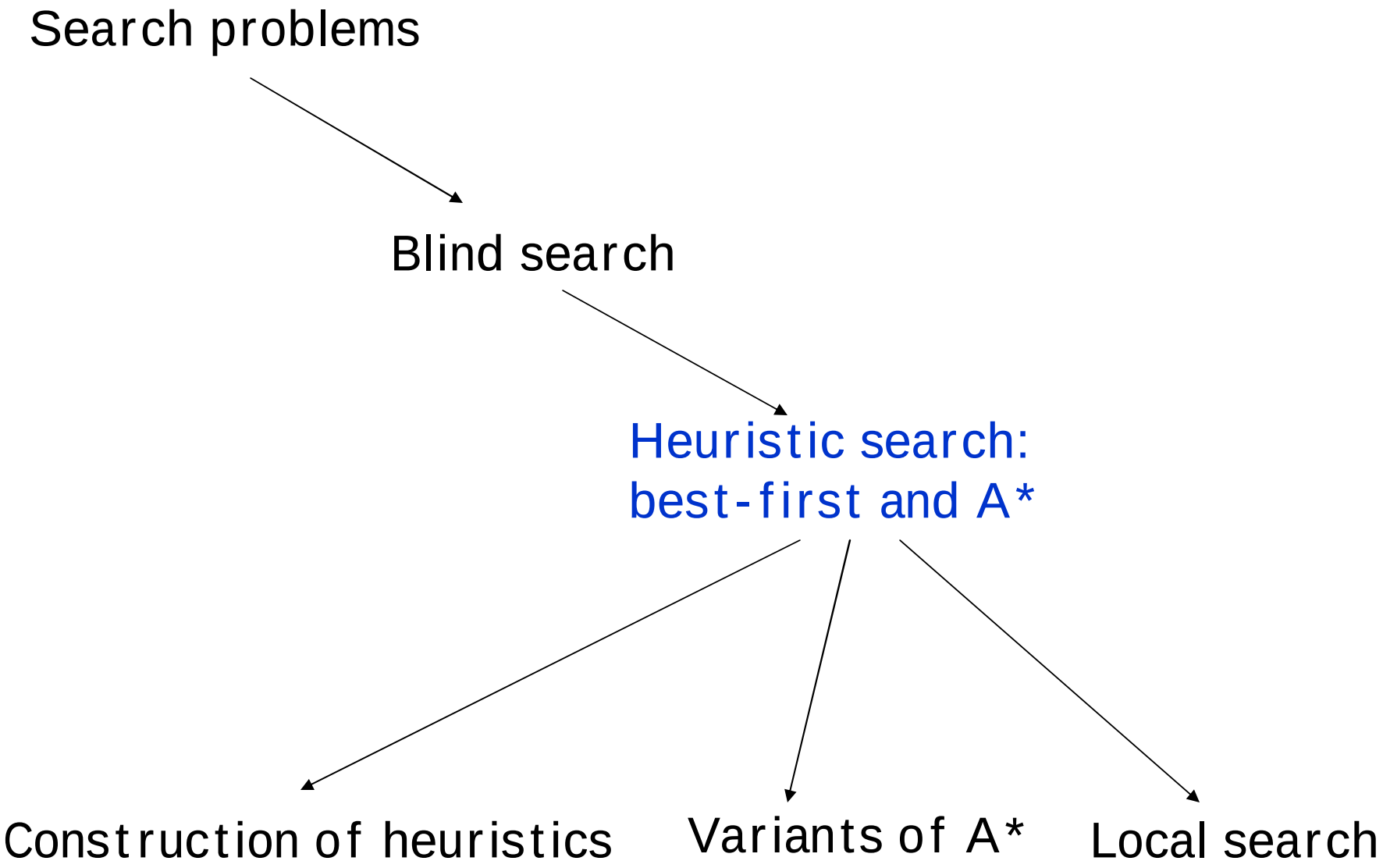
Example: N-Queens



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

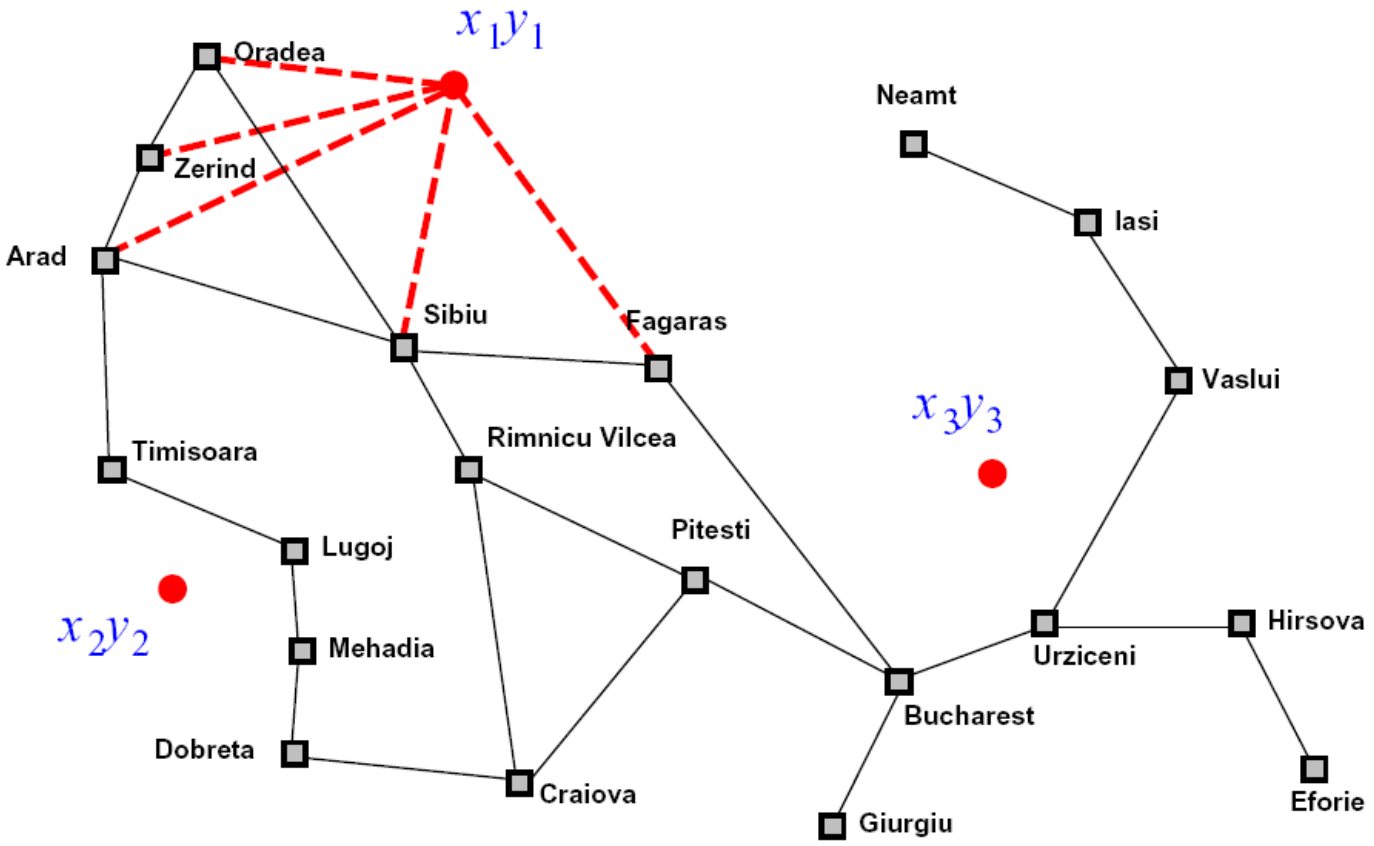
The Basic Genetic Algorithm

1. Generate random population of chromosomes
2. Until the end condition is met, create a new population by repeating following steps
   1. Evaluate the fitness of each chromosome
   2. Select two parent chromosomes from a population, weighed by their fitness
   3. With probability p_c cross over the parents to form a new offspring.
   4. With probability p_m mutate new offspring at each position on the chromosome.
   5. Place new offspring in the new population
3. **Return** the best solution in current population



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

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

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

  - More on this next class…

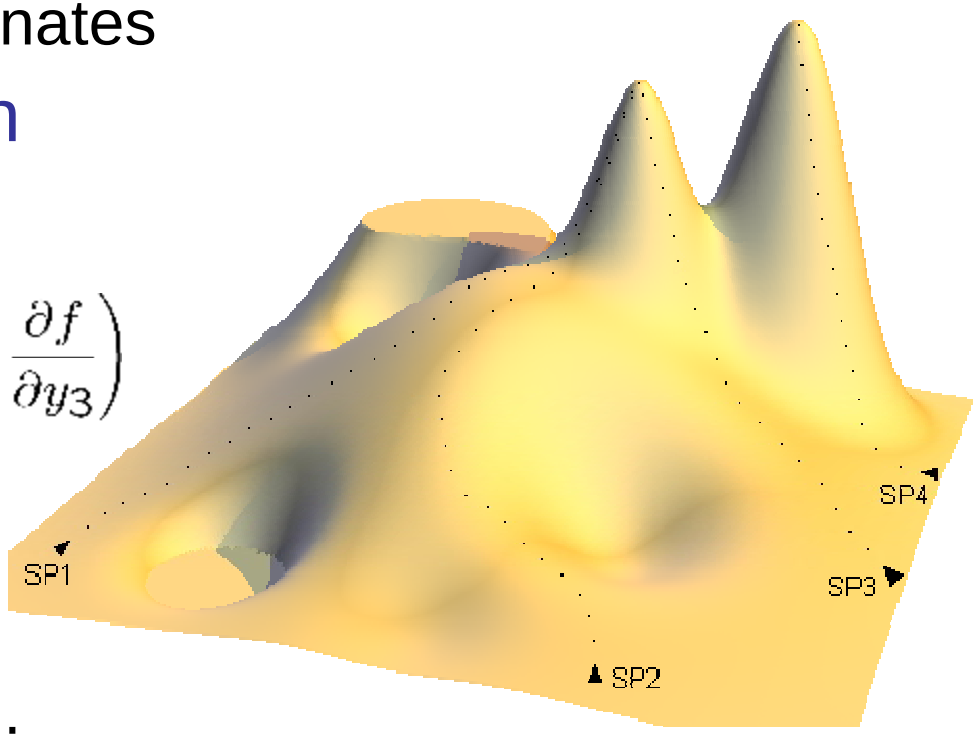


Image from vias.org