343H: Honors AI

Lecture 5 – Beyond classical search

1/30/2014

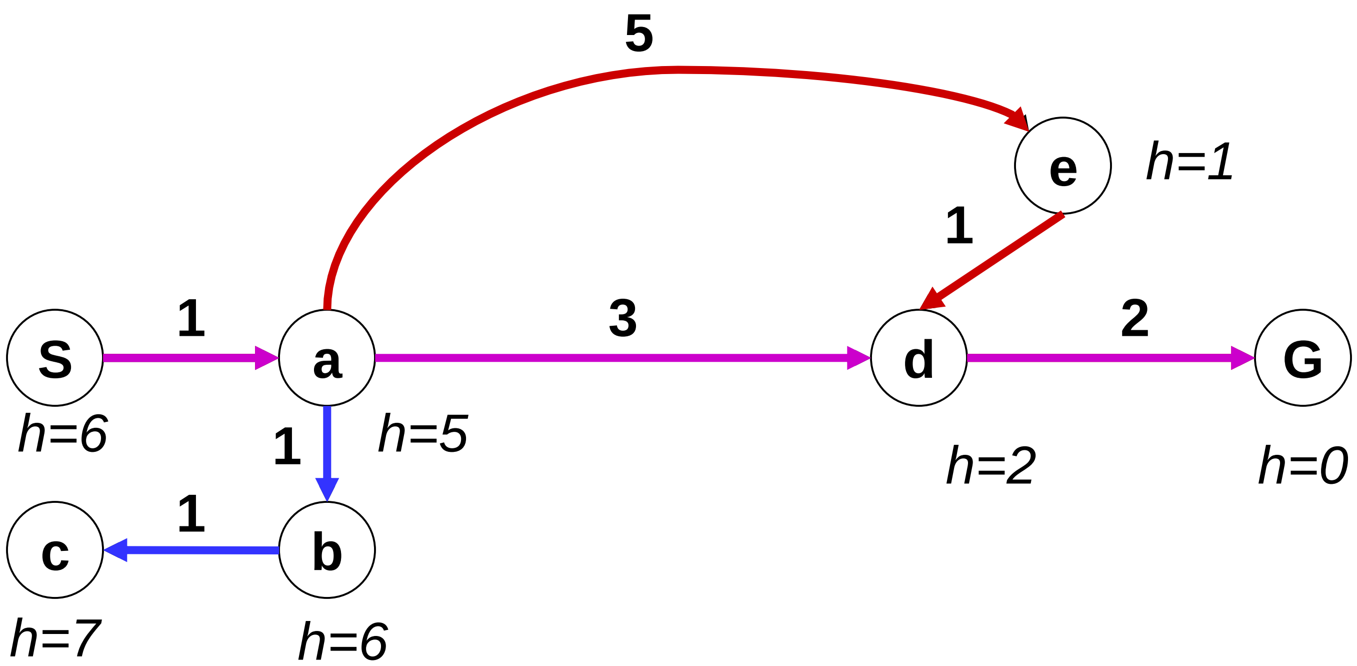
Slides courtesy of Dan Klein, UC-Berkeley
Unless otherwise noted

Today

- Review of A* and admissibility
- Graph search
- Consistent heuristics
- Local search
  - Hill climbing
  - Simulated annealing
  - Genetic algorithms
  - Continuous search spaces

Recall: A* Search

- Uniform-cost orders by <u>path cost</u>, or *backward cost* g(n)
- Greedy orders by <u>goal proximity</u>, or *forward cost* h(n)



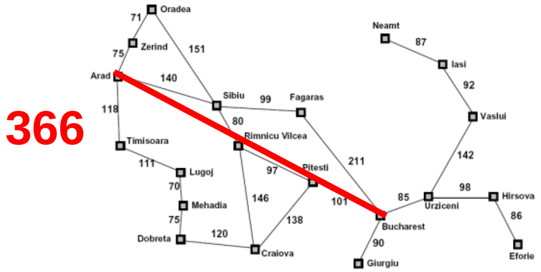
- A* Search orders by the sum: f(n) = g(n) + h(n)

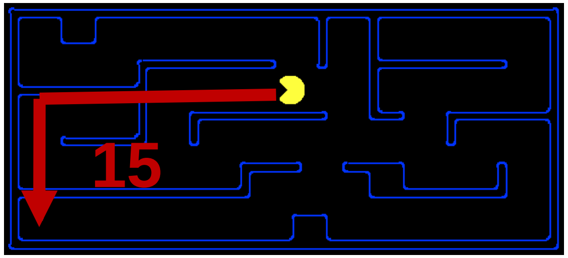
Example: Teg Grenager

Recall: Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available





- Inadmissible heuristics are often useful too (why?)

Generating heuristics

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

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

Trivial Heuristics, Dominance

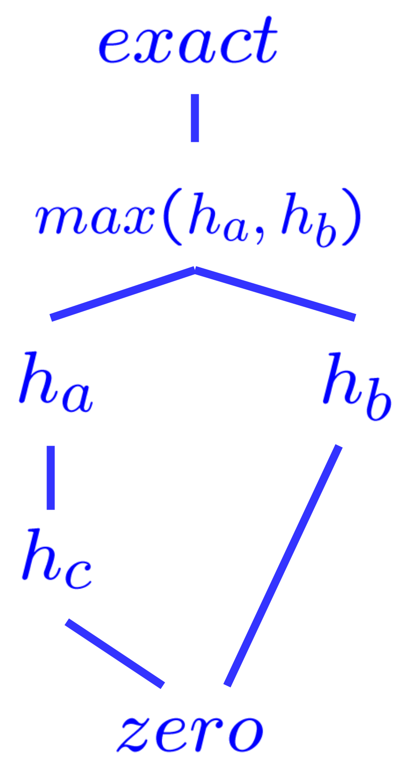
- Dominance: $h_a \geq h_c$ if

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

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

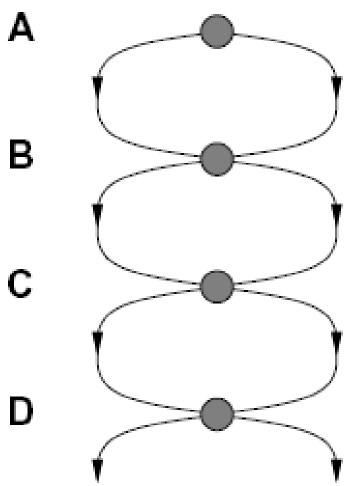
$$h(n) = max(h_a(n), h_b(n))$$

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

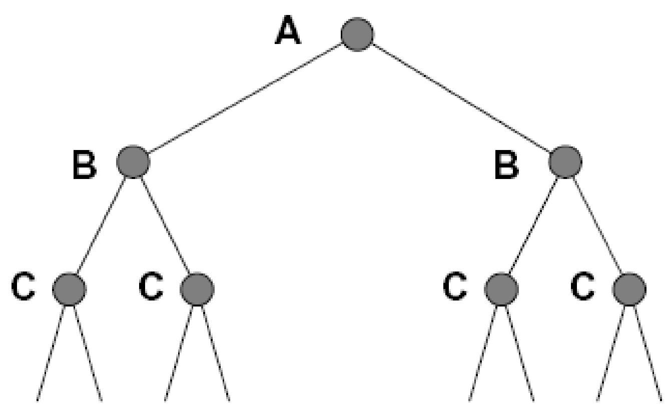


Tree Search: Extra Work!

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



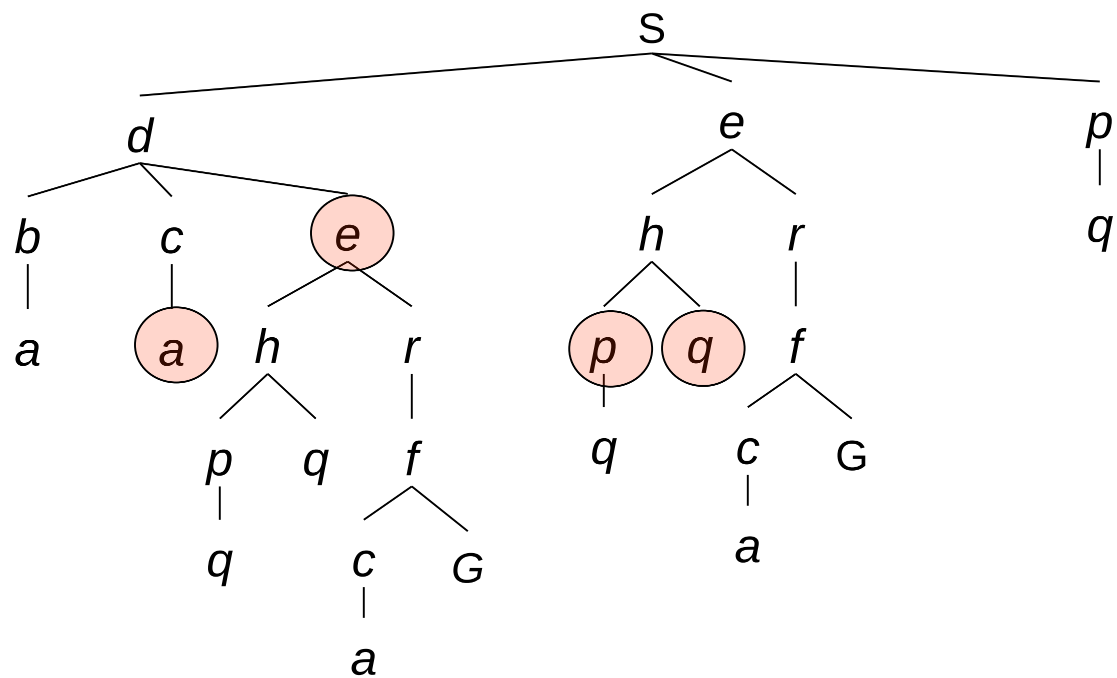
State graph



Search tree

Graph Search

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



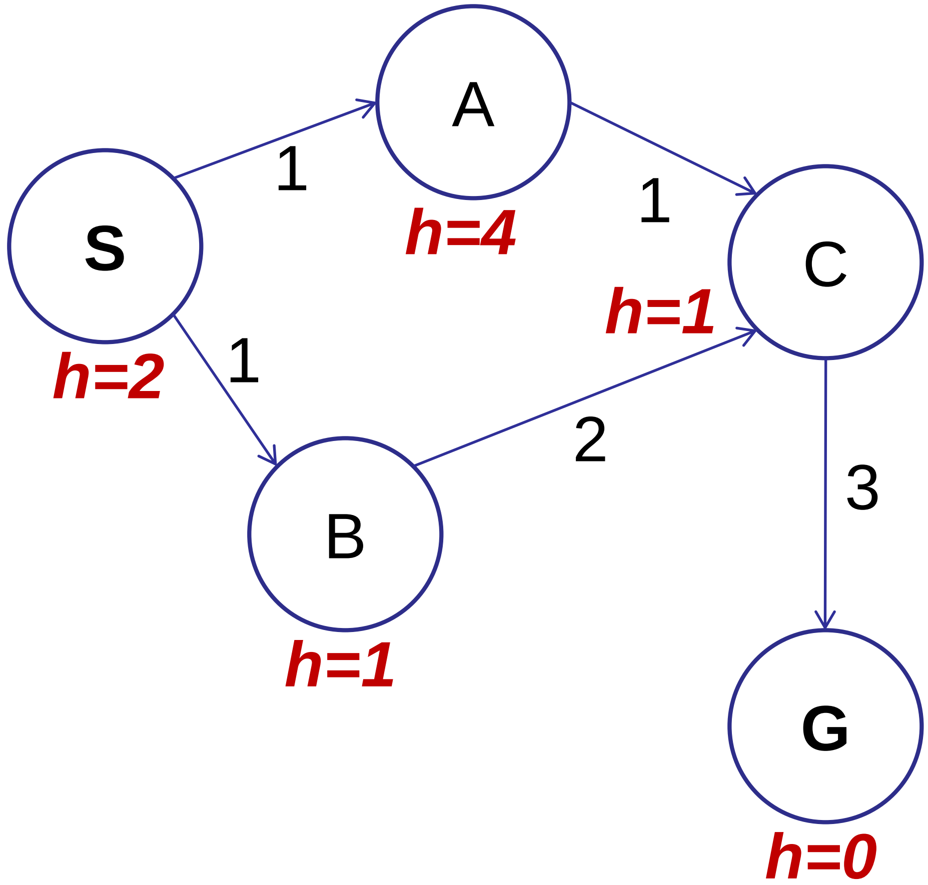
Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state is new
  - If not new, skip it
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

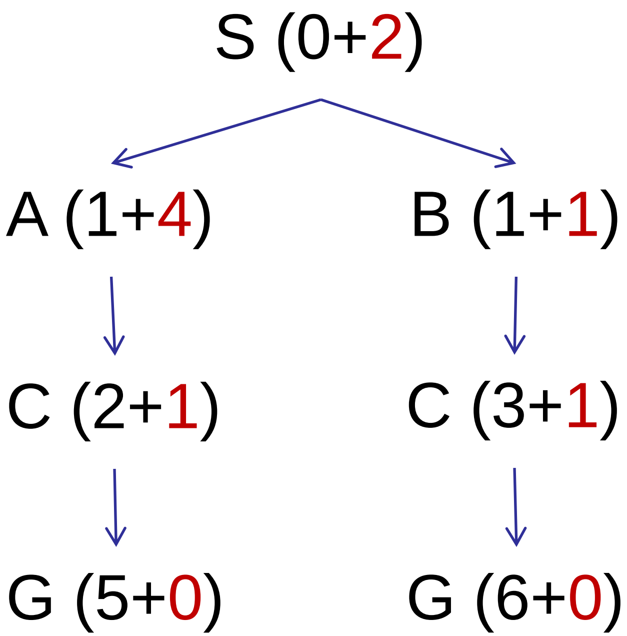
Warning: 3e book has a more complex, but also correct, variant

A* Graph Search Gone Wrong?

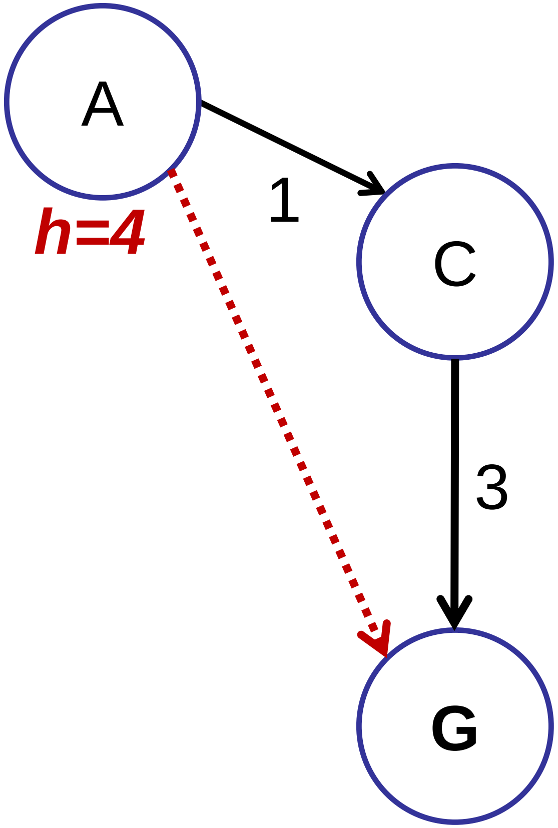
State space graph



Search tree

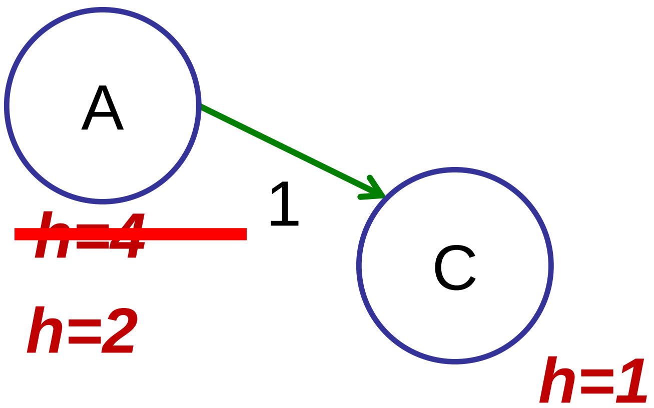


Consistency of Heuristics



- Admissibility: heuristic cost <= actual cost to goal
  - h(A) <= actual cost from A to G

Consistency of Heuristics



- Stronger than admissibility
- Definition:
- heuristic cost <= actual cost *per arc*
- h(A) - h(C) <= cost(A to C)
- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal

Optimality

- Tree search:
  - A* is 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)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

Summary: A*

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

Today

- Review of A* and admissibility
- Graph search
- Consistent heuristics
- Local search
  - Hill climbing
  - Simulated annealing
  - Genetic algorithms
  - Continuous search spaces

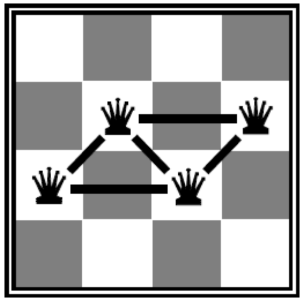
Local Search Methods

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve what you have until you can't make it better
- Tradeoff: Generally much faster and more memory efficient (but incomplete)

Types of Search Problems

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

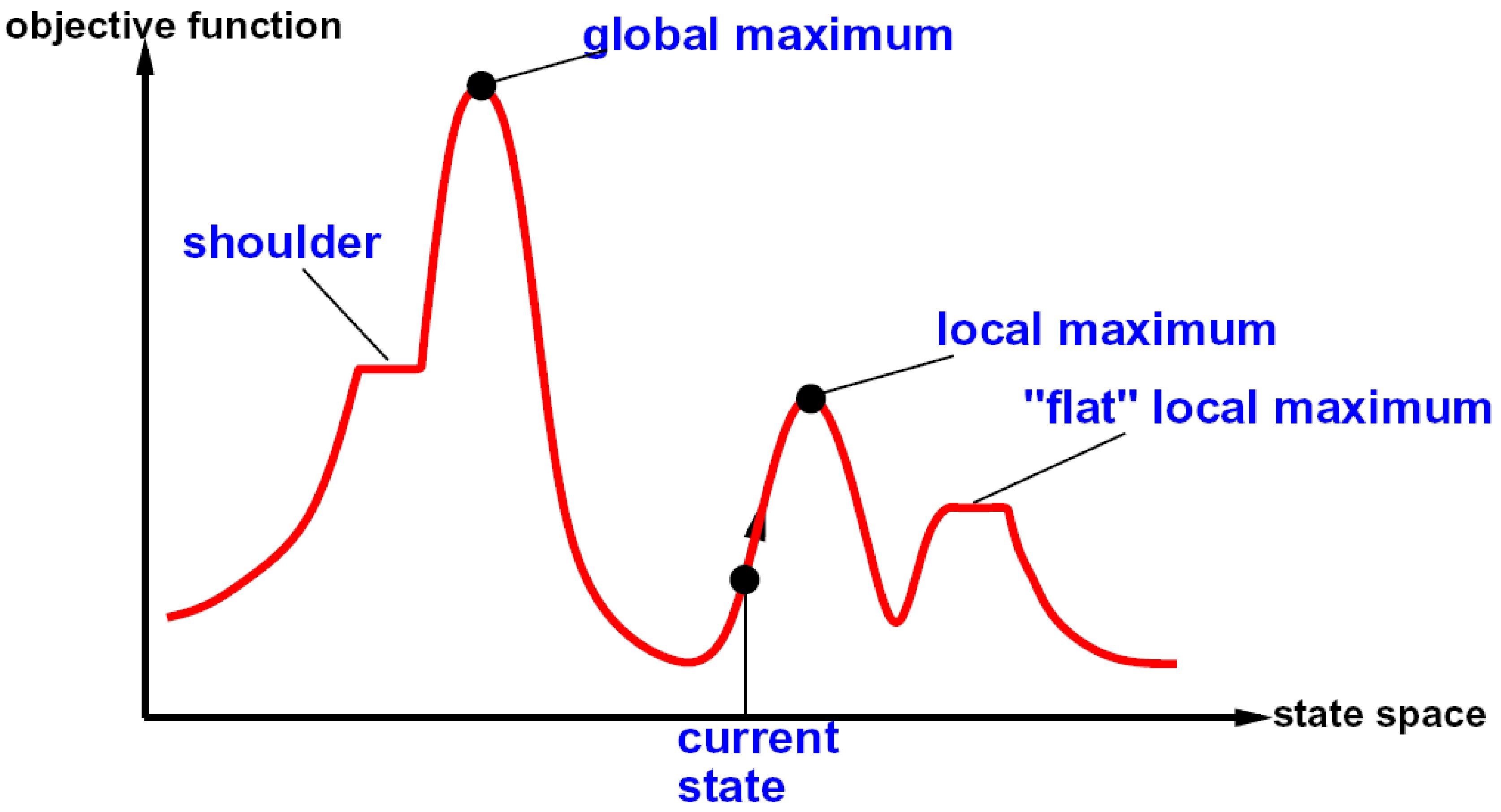
- Identification problems:
  - We actually just want to know what the goal is (examples?)
  - Usually want an optimal goal
  - *Complete-state formulations*
  - Iterative improvement algorithms



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?

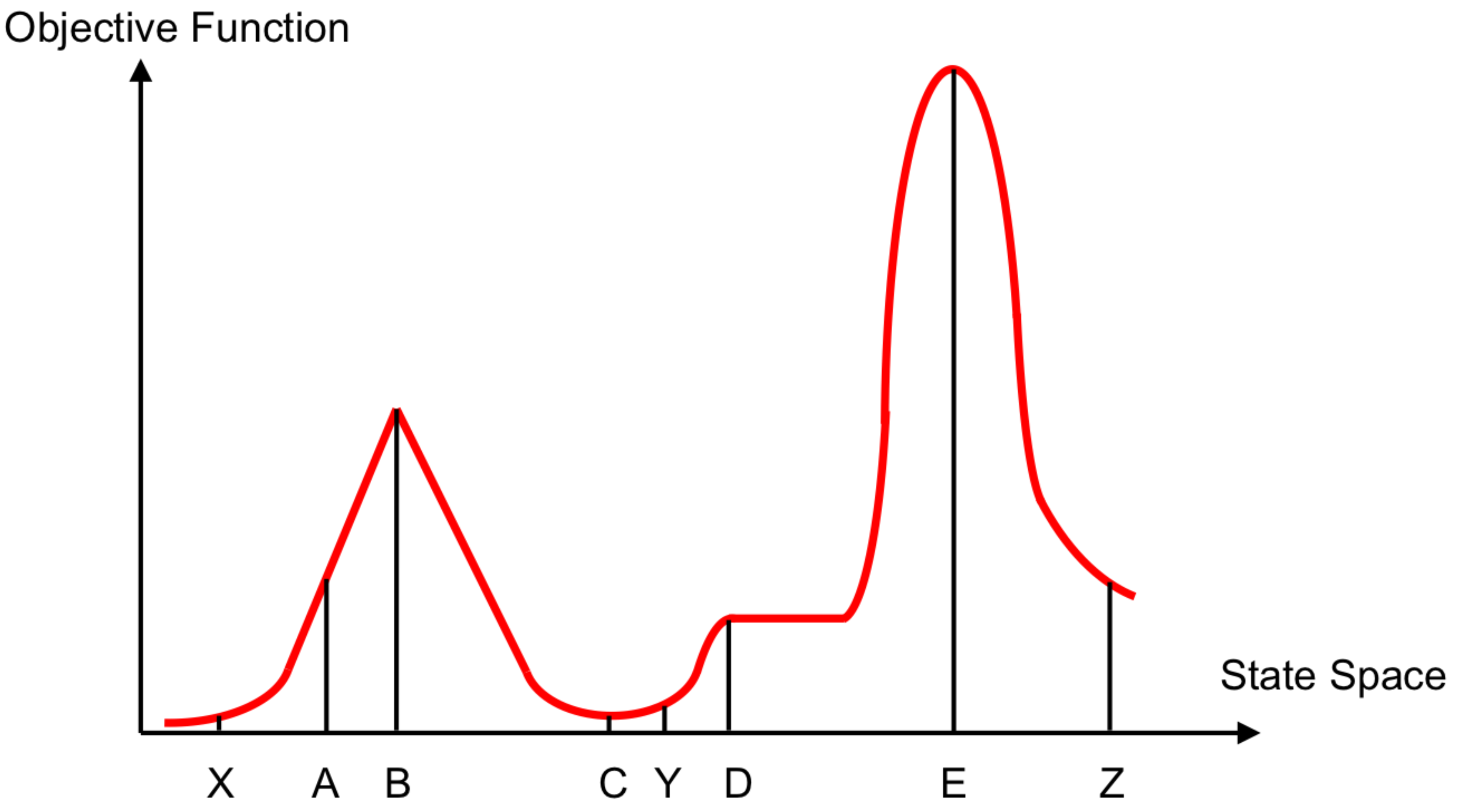
Hill Climbing Diagram



- Sideways steps?
- Random restarts?

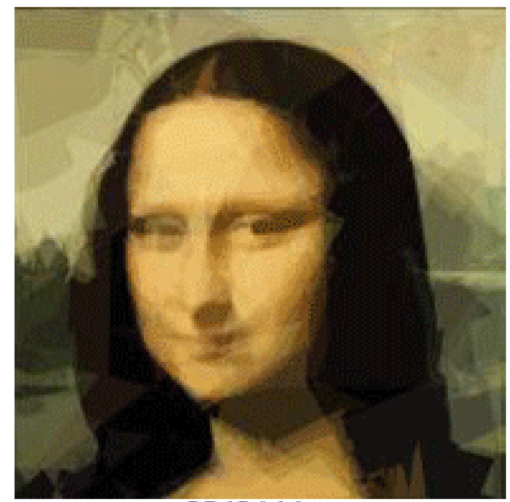
Quiz

- Hill climbing on this graph:



Hill climbing Mona Lisa

Could the computer paint a replica of the Mona Lisa using only 50 semi transparent polygons?



904314.jpg

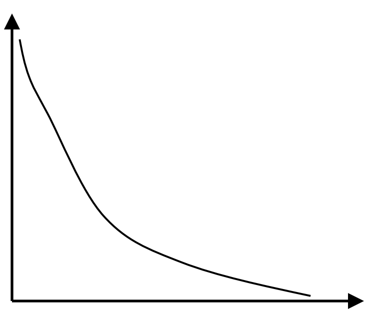
- http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

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

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



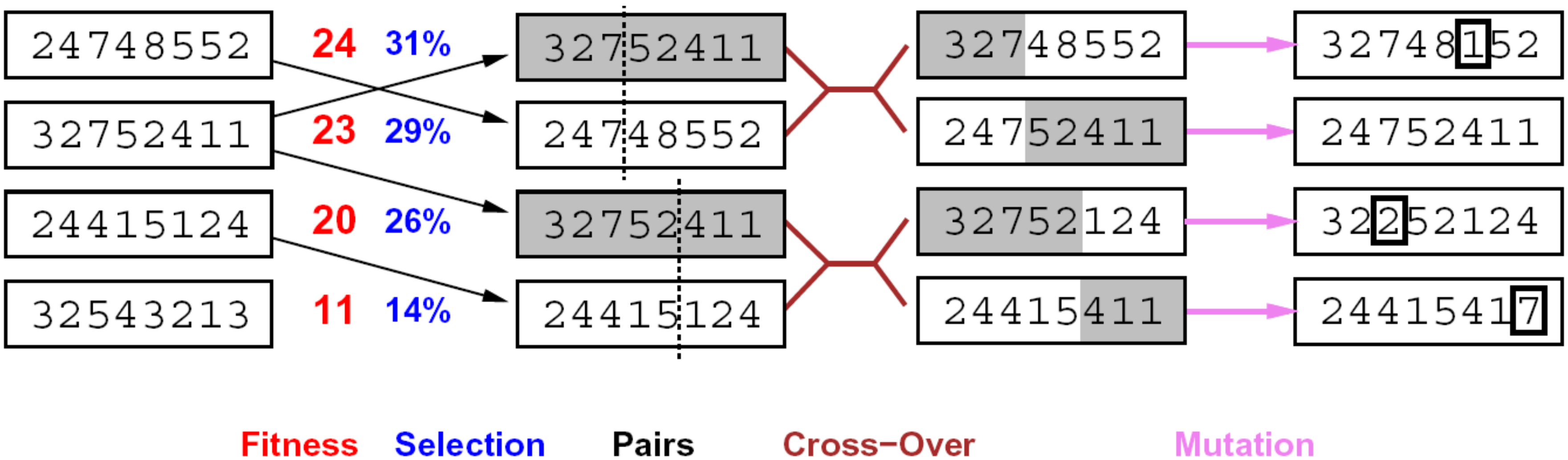
Beam Search

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

Greedy Search | Beam Search

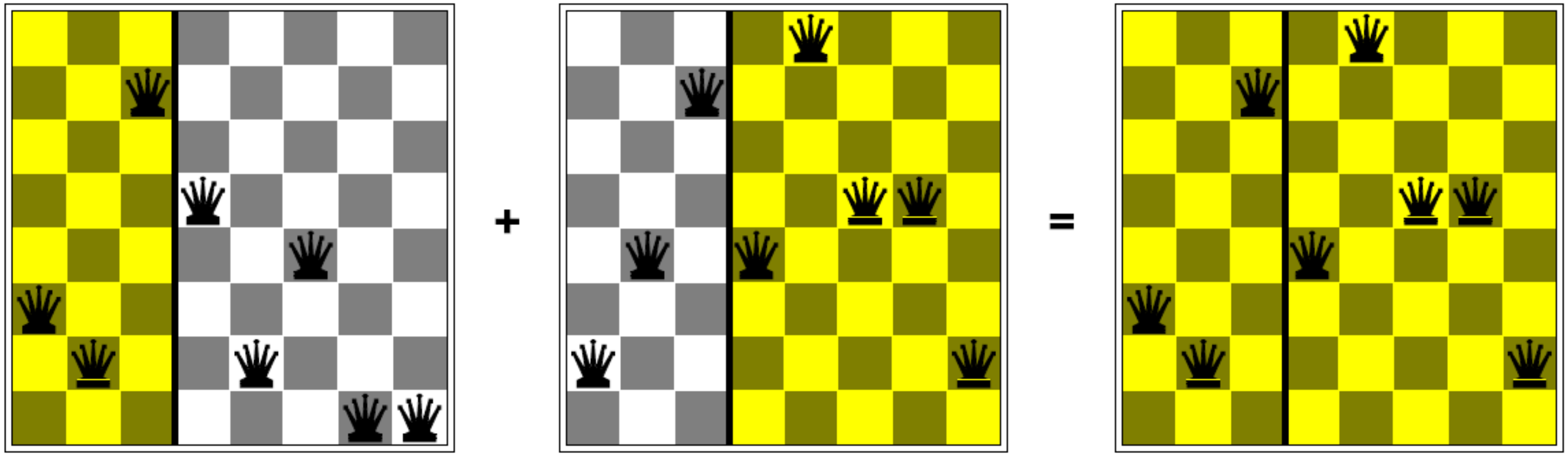
- Variables: beam size, encourage diversity?
- The best choice in many practical settings

Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
- Like beam search (selection), but also have pairwise crossover operators, with optional mutation

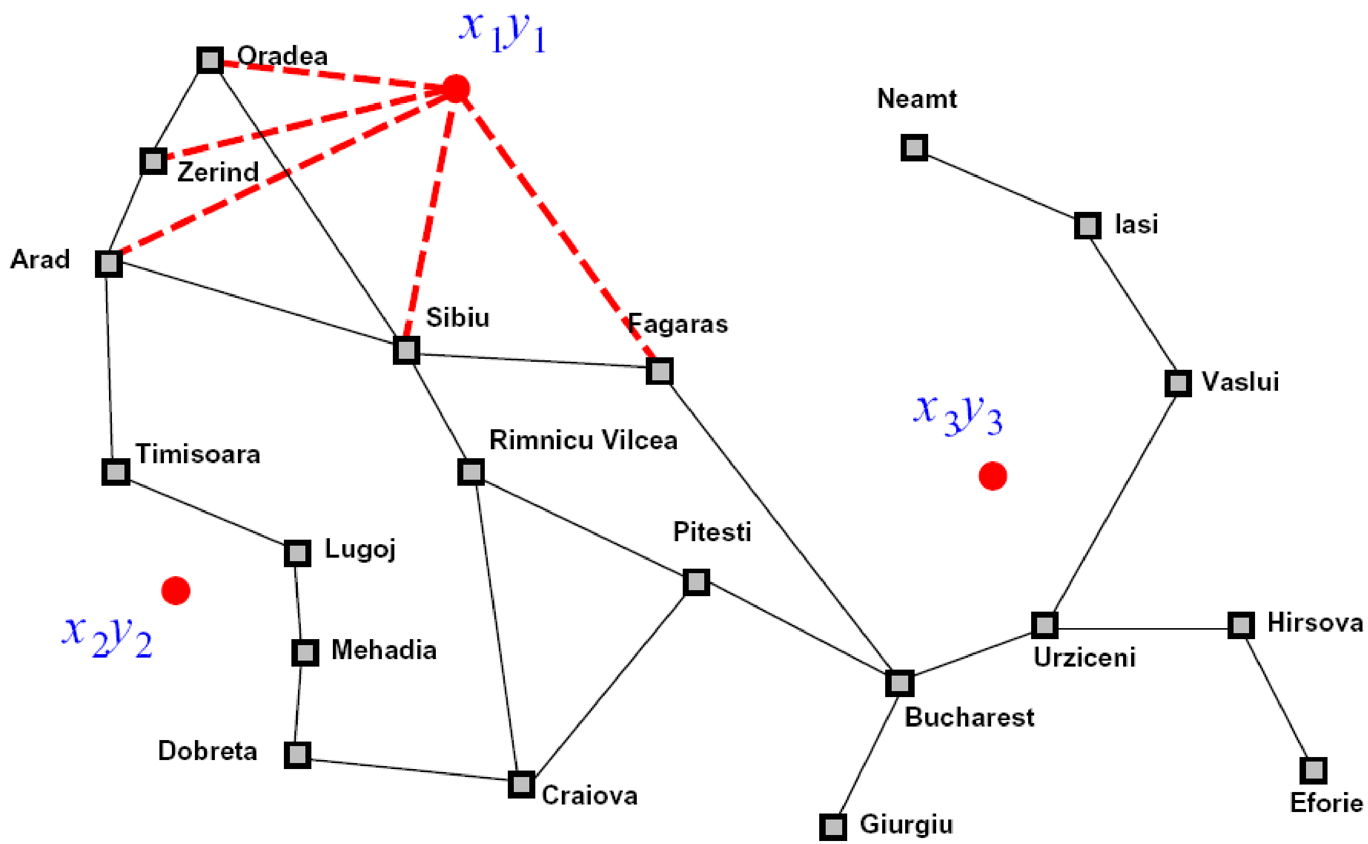
Example: N-Queens



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

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

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

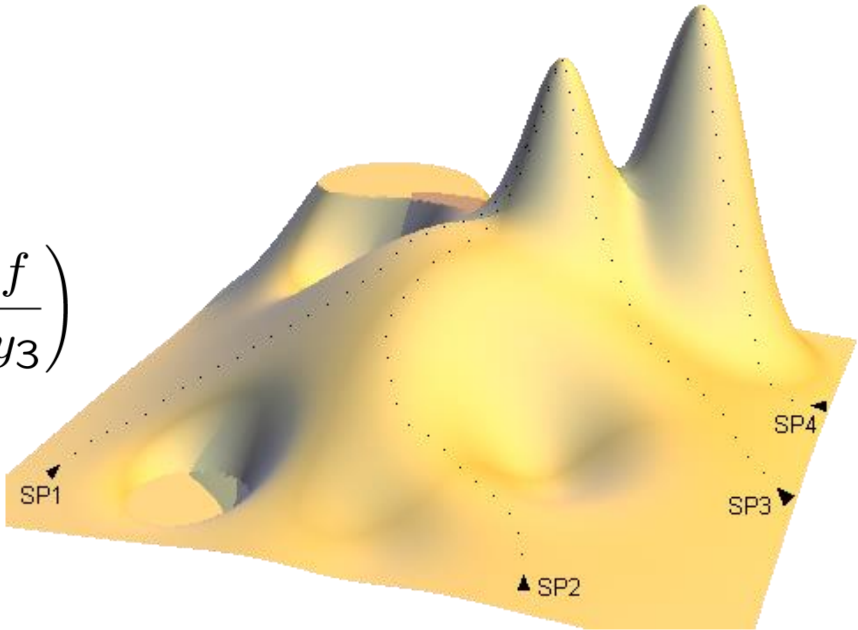


Image from vias.org

Example: Continuous local search

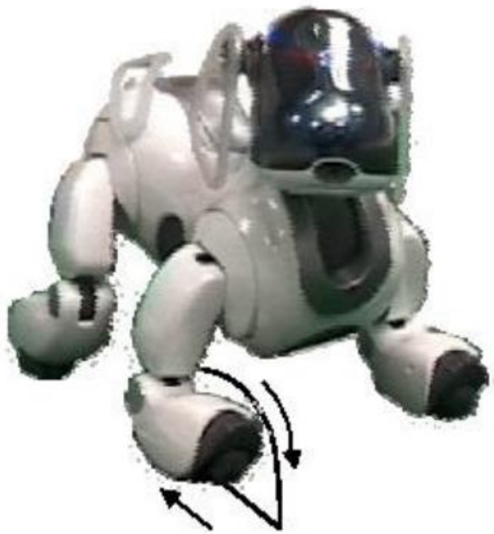
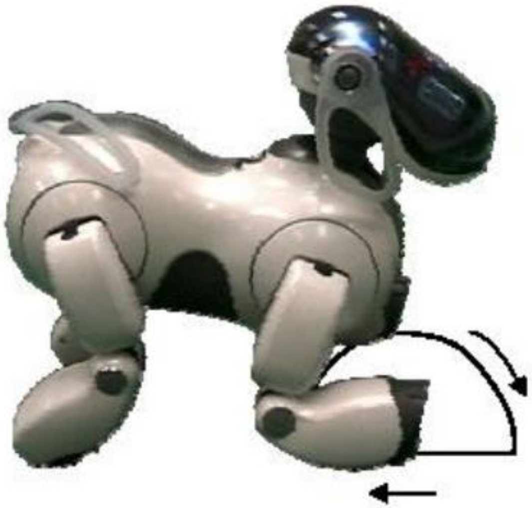
Goal: Enable an Aibo to walk as fast as possible

- Start with a **parameterized walk**
- **Learn** fastest possible parameters
- **No simulator** available:
  - Learn entirely on robots
  - Minimal human intervention

Slide credit: Peter Stone

A parameterized walk

- Trot gait with elliptical locus on each leg
- 12 continuous parameters (ellipse length, height, position, body height, etc)

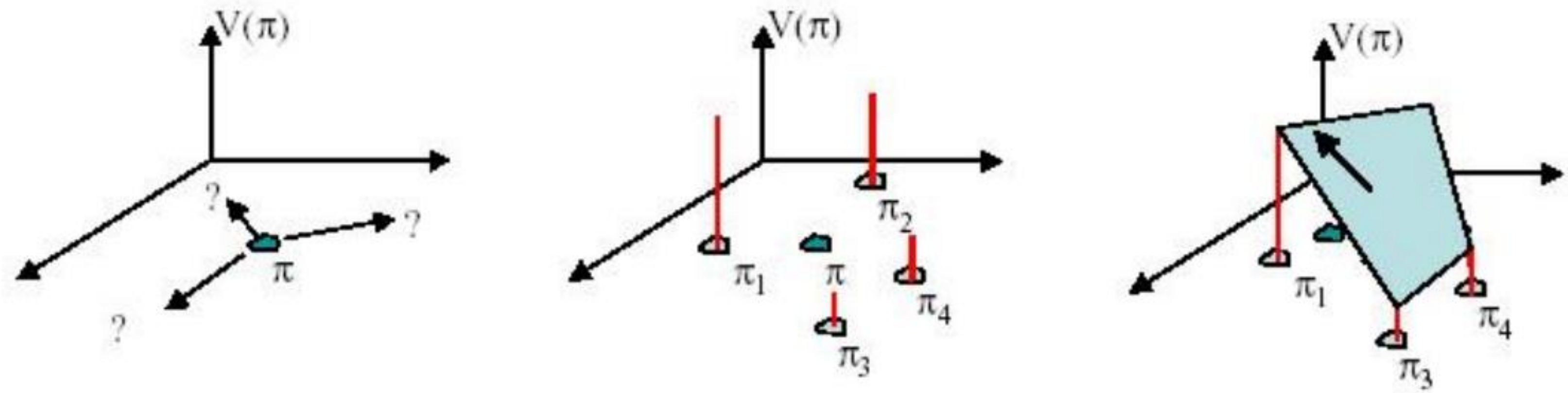
Slide credit: Peter Stone

Experimental setup

- Policy $\pi = \{\theta_1, \ldots, \theta_{12}\}$, $V(\pi)$ = walk **speed** when using π
- Training Scenario
  - Robots **time themselves** traversing fixed distance
  - Multiple traversals (3) per policy to account for **noise**
  - **Multiple robots** evaluate policies simultaneously
  - Off-board computer **collects results**, **assigns policies**

Policy gradient reinforcement learning

- From π want to move in direction of **gradient** of $V(\pi)$
  - Can't compute $\frac{\partial V(\pi)}{\partial \theta_i}$ directly: **estimate** empirically
- Evaluate **neighboring policies** to estimate gradient
- Each trial randomly varies **every parameter**



Slide credit: Peter Stone

Summary

- Graph search
  - Keep closed set, avoid redundant work
- A* graph search
  - Optimal if h is consistent
- Local search: Improve current state
  - Avoid local min traps (simulated annealing, crossover, beam search)