Local Search

Today's Plan

- Project 1 comments and Q&A
- Iterative Improvement for CSPs → Local Search

Project 1 Tips

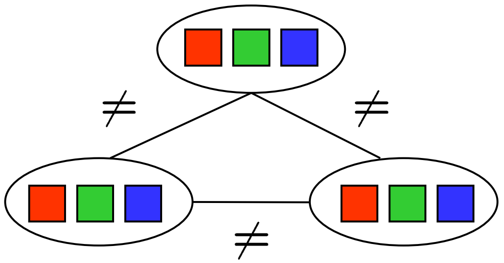
- Use Piazza, read FAQ before posting questions: https://piazza.com/class/ixql4613j9k223?cid=8
- **Questions 1-4**: if you develop a correct solution for DFS, the rest will be easy modifications
- **Run autograder** after *every* question. Until you pass all the test cases, assume your code has bugs.
- It's OK to (re) submit multiple times, only latest attempt is graded.

Tips for Project 1 (cont'd)

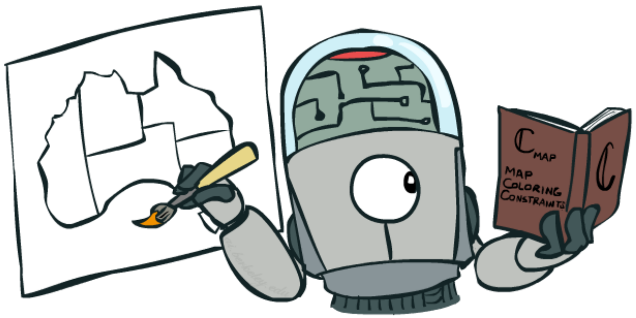
- Problems 5-8 <u>depend on code in 1-4</u>. Get that right (and tested) first, before moving on!
- P5/Corners problem: must visit all corners in *single* path
  - Implications for search tree, state info to update
- Heuristics for p6-8: start simple. For extra credit, think back to graph traversal algorithms from cs323.

Reminder: CSPs

- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a list of the legal tuples)
    - Unary / Binary / N-ary



- Goals:
  - Here: find any solution
  - Also: find all, find best, etc.



Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

Why more efficient to use a recursive alg (not iterative + Fringe like regular search)?

Improving Backtracking

- General-purpose ideas give huge gains in speed
  - … but it's all still NP-hard

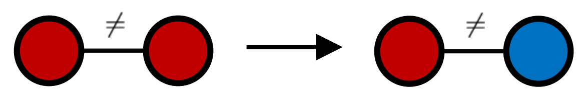
- **Filtering**: Can we detect inevitable failure early?
- **Ordering**:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)

- Many other ideas (e.g., problem structure)

<u>**Iterative**</u> Algorithm for CSPs: <u>MinConflicts</u>

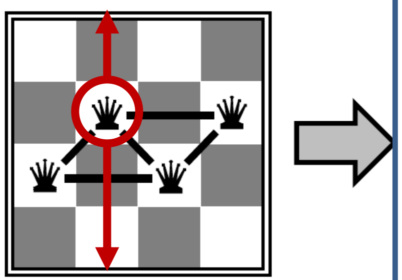
- Local search methods typically work with "complete" states, i.e., all variables assigned

- To apply to CSPs:
  - Take an assignment with unsatisfied constraints
  - Operators *reassign* variable values
  - **No fringe**! Live on the edge.



- <u>Greedy algorithm</u>: While not solved,
  - Variable selection: randomly select **any** conflicted variable
  - Value selection: <u>**min-conflicts**</u> heuristic:
    - Choose a value that violates the fewest constraints
    - I.e., **hill climb** with h(n) = <u>total number of violated constraints</u>

Example: 4-Queens



h = 5

[Board]

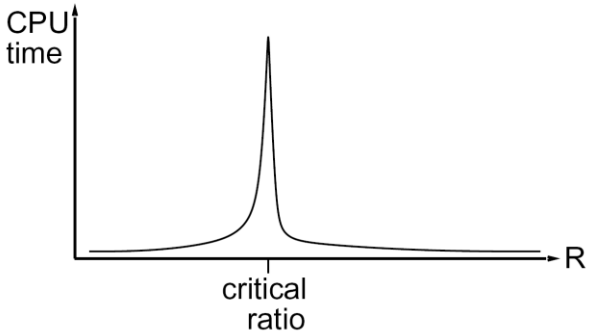
- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

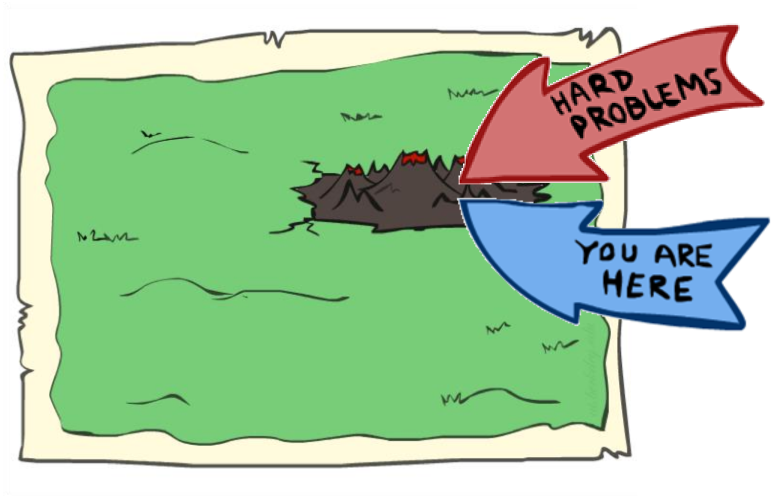
Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

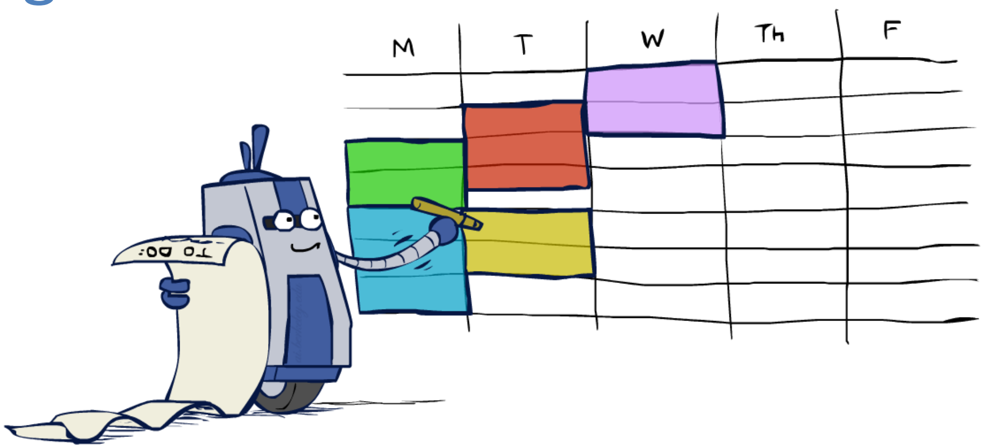
$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$





Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by
- Basic solution: backtracking se
- Speed-ups:
  - Ordering
  - Filtering



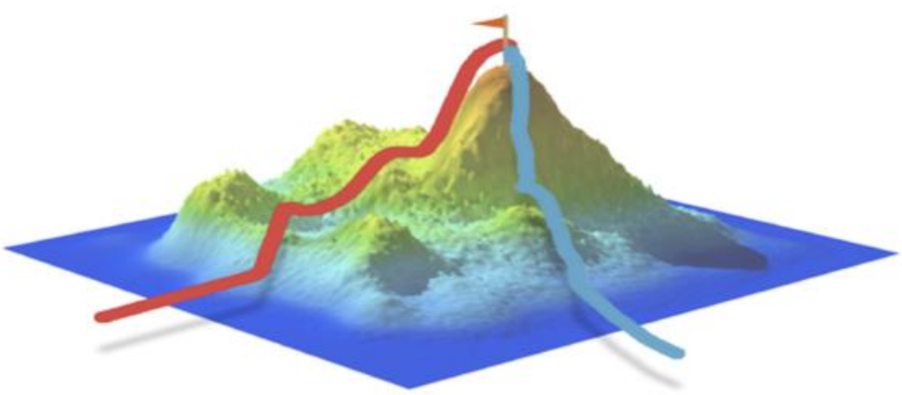
- Iterative min-conflicts is often effective in practice

Google's Deep Mind Wins at Go

- http://deepmind.com/alpha-go.html
- "Simple" game, but difficult to master
- Orders of magnitude more states than chess (x "googol")
- Traditional search (A*) not feasible
- Google solution:
  - ➢ Local search w/ restarts (this lecture)
  - – "deep learning" to estimate state values (instead of heuristics) ← observing human players
  - – https://googleblog.blogspot.com/2016/01/alphago-machine-learning-game-go.html

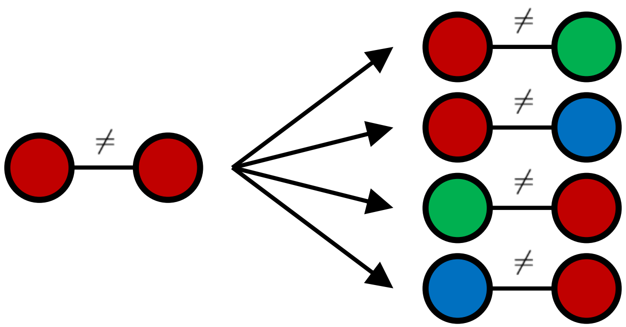
Searching Large (or Infinite) Spaces

- Too many states to explore using A* or variants
  - Large or infinite branching factor (continuous)
- “Reasonable” solution is good enough
- Local search idea: **start with initial guess and incrementally improve**



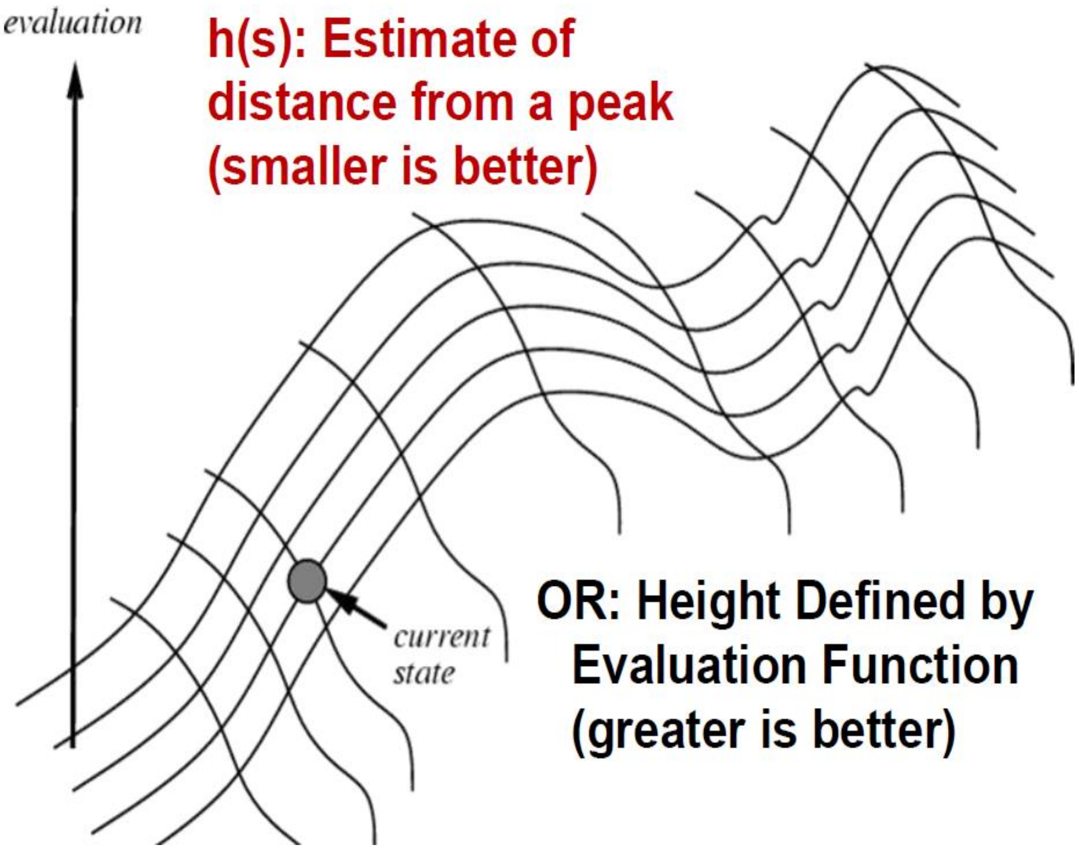
Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can’t make it better (no fringe!)
- New successor function: local changes



- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing



h(s): Estimate of distance from a peak (smaller is better)

OR: Height Defined by Evaluation Function (greater is better)

Hill Climbing: Trade-offs

- What's bad about this approach?
  - Complete?
  - Optimal?
- What's good about it?

Hill Climbing: Algorithm

I. **While ($\exists$ uphill points):**
   - Move in the direction of increasing evaluation function ***f***

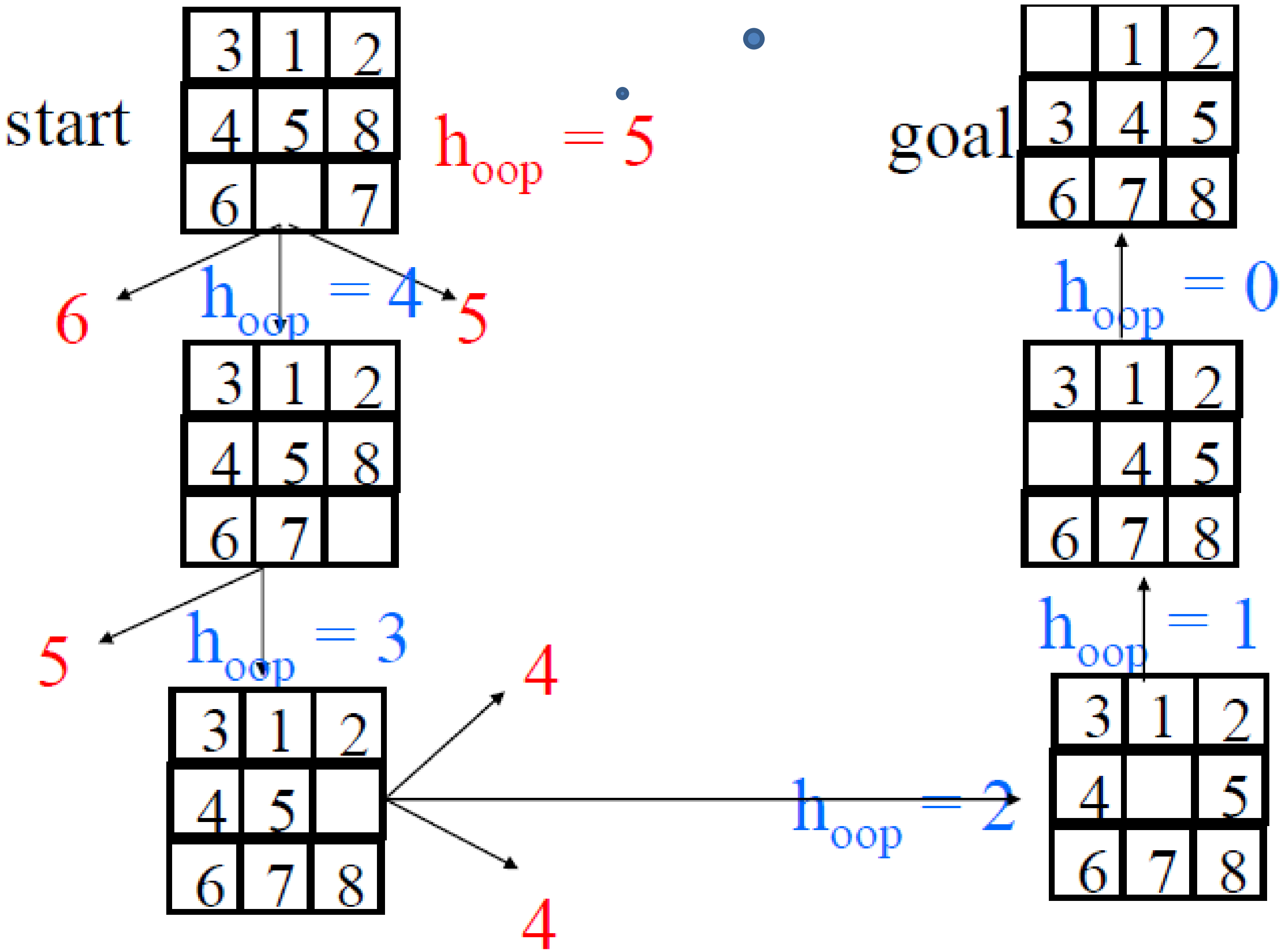
II. **Let $s_{next} = \arg\max_{s} f(s)$, s a successor state to the current state n**
   - If $f(n) < f(s)$ then move to s
   - Otherwise halt at n

- **Properties:**
  - Terminates when a peak is reached.
  - Does not look ahead of the immediate neighbors of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Doesn't *backtrack*, since it doesn't remember where it's been
- **a.k.a. *greedy local search***

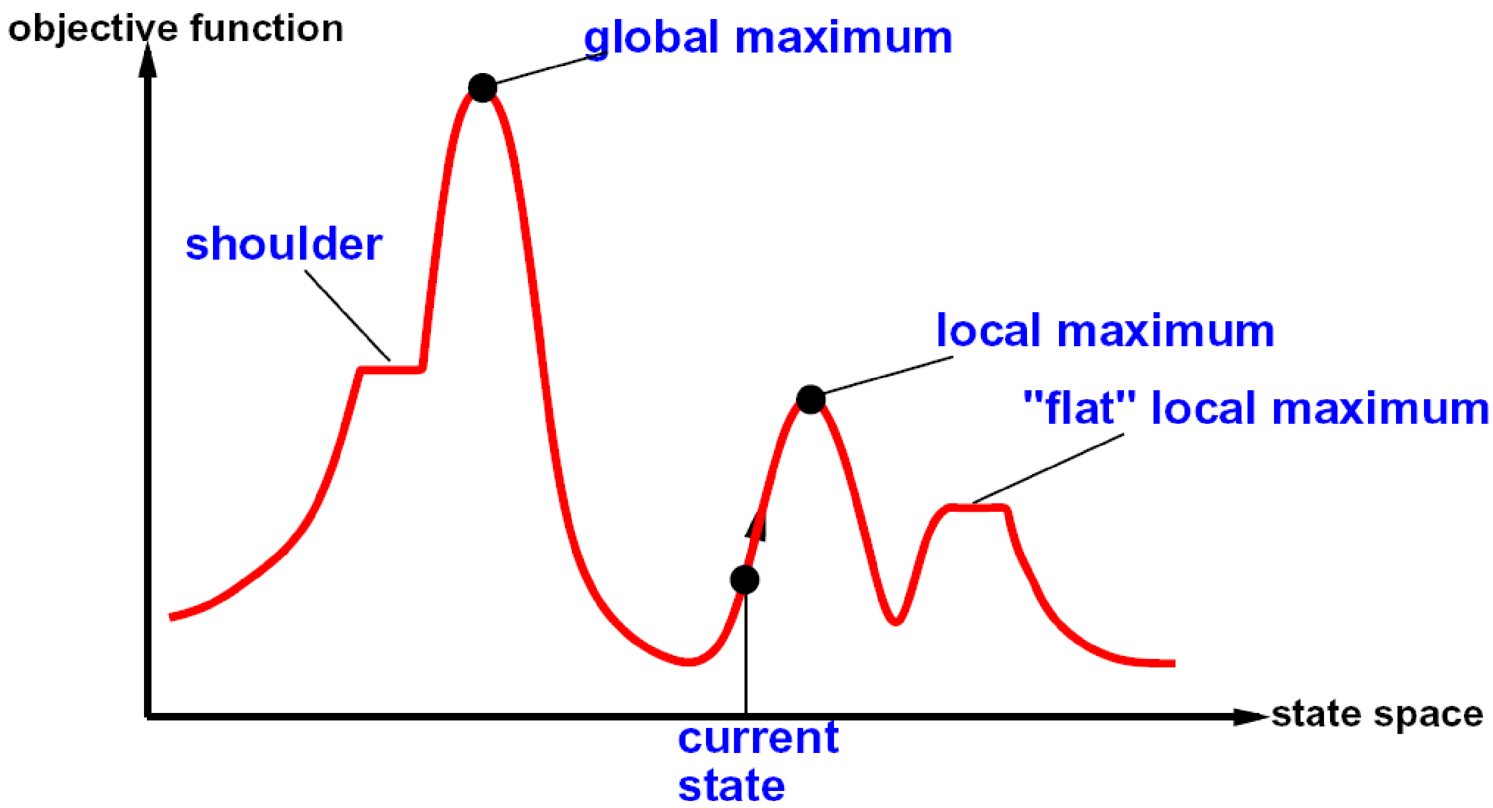
"Like climbing Everest in thick fog with amnesia"

Toy Example: Tiles

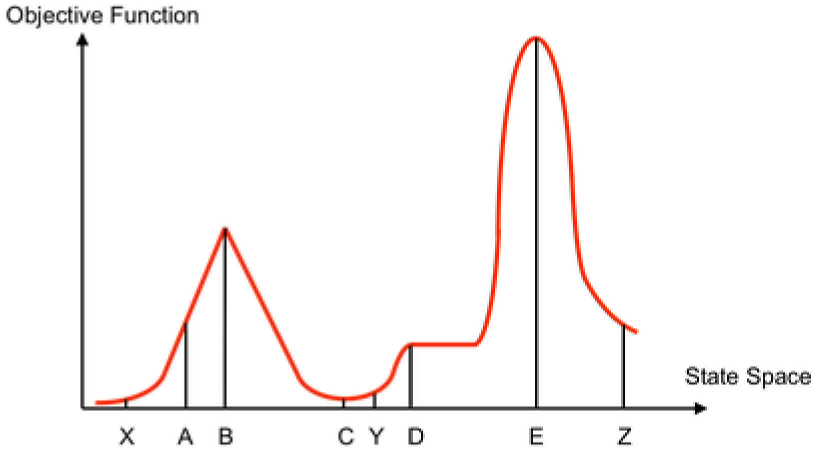
h= out of place (oop) tiles



Hill Climbing Diagram



Hill Climbing Quiz



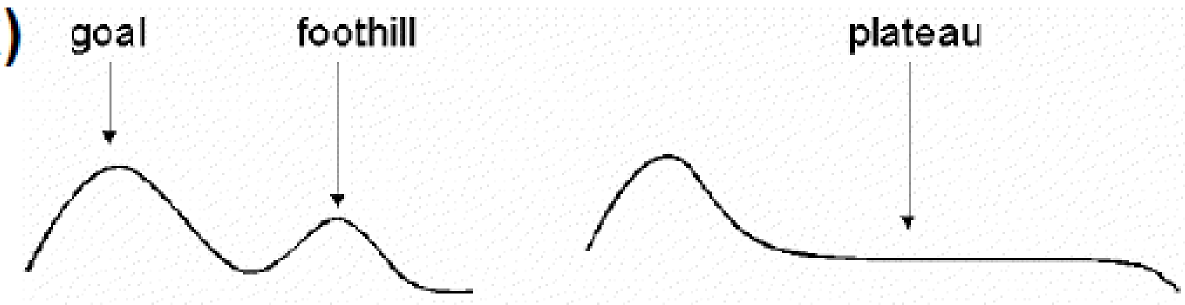
Starting from X, where do you end up ?

Starting from Y, where do you end up ?

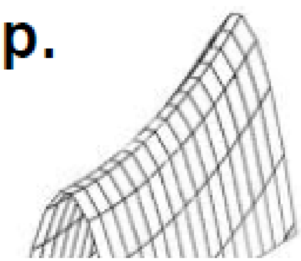
Starting from Z, where do you end up ?

Drawbacks of Hill Climbing

- Local Maxima: peaks that aren't the highest point in the space
- Plateaus: the space has a broad flat region that gives the search algorithm no direction (random walk)

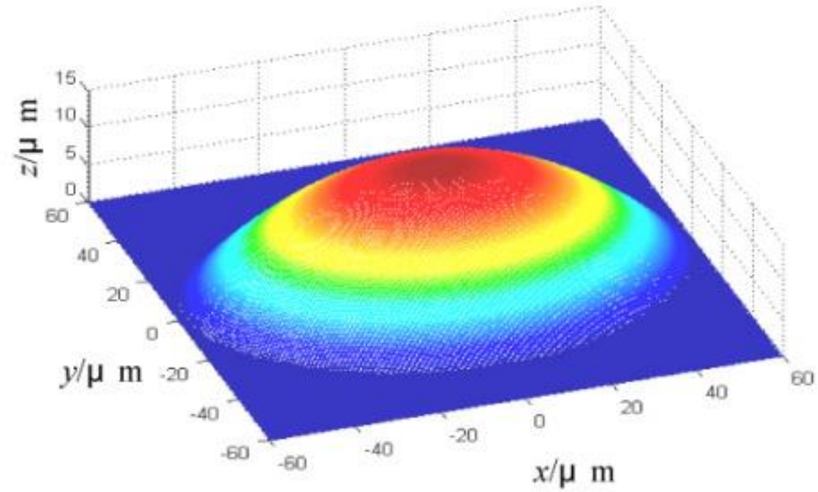


- Ridges: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

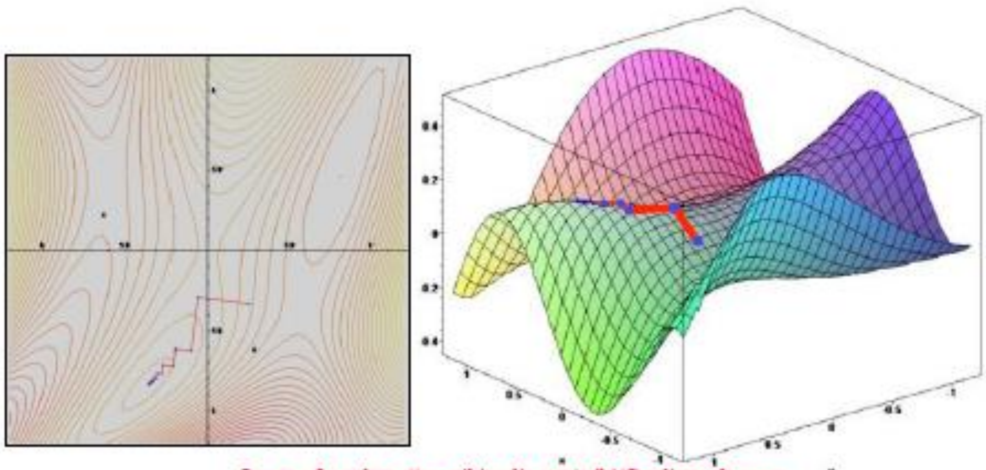


“Easy” Problems: Convex Surface

- No local maxima (only 1 peak)
- Hill climbing works great
- Can we make it faster?



Gradient Descent (Steepest Descent)



Images from http://en.wikipedia.org/wiki/Gradient_descent

- Gradient descent procedure for finding the $arg_x\ min\ \ f(x)$
  - choose initial x_0 randomly
  - repeat
    - $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
  - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size η (eta) is small (perhaps 0.1 or 0.05)

http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/

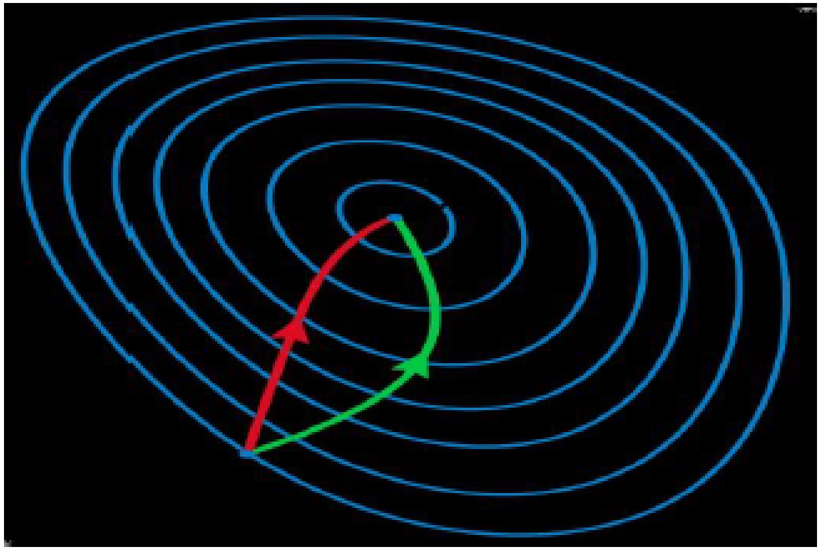
Gradient Ascent vs. Newton-Ralphston

- A reminder of Newton's method from Calculus:

  $x_{i+1} \leftarrow x_i - \eta f'(x_i) / f''(x_i)$

- Newton,s method uses 2nd order information (the second derivative, or, curvature) to take a more direct route to the minimum.

- The second-order information is more expensive to compute, but converges quicker.



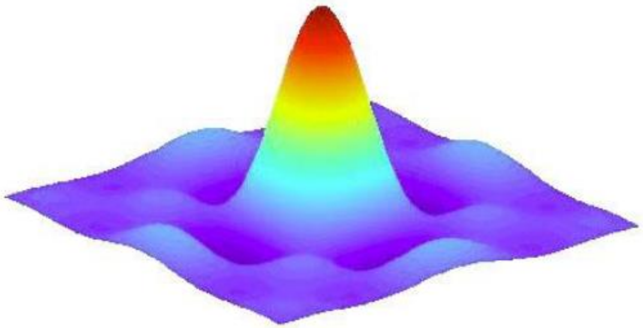
Contour lines of a function
Gradient descent (green)
Newton,s method (red)

Image from http://en.wikipedia.org/wiki/Newton's_method_in_optimization

(this and previous slide from Eric Eaton)

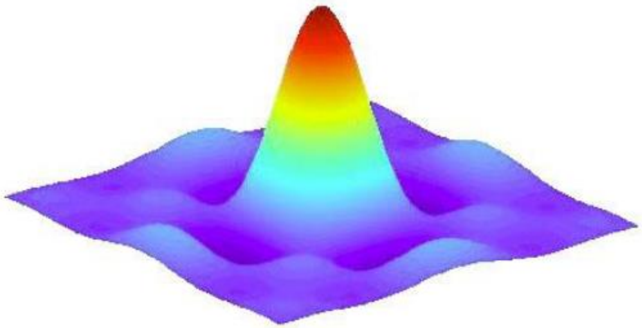
Problem: Non-Convex Surfaces

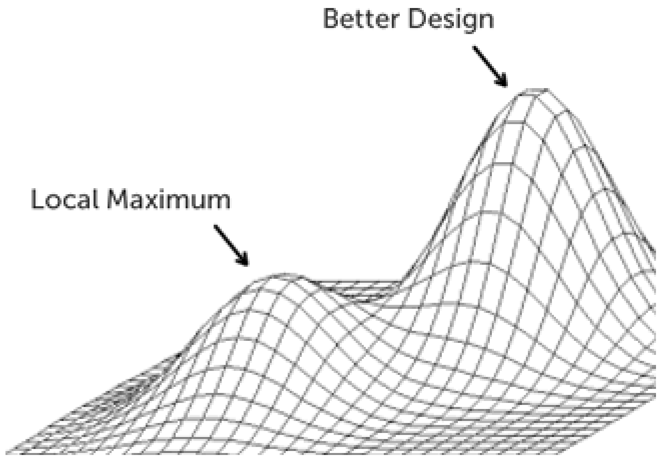
- Realistic problems:
  - Many local suboptimal maxima
  - Easy to get trapped

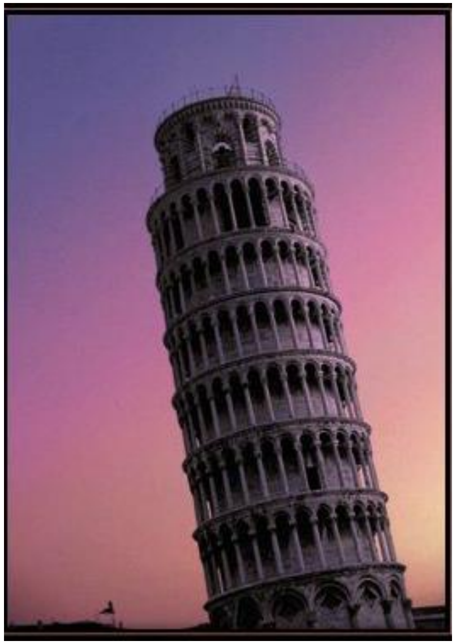


Problem: Non-Convex Surfaces

- Realistic problems:
  - Many local suboptimal maxima
  - Easy to get trapped
- Examples:







Solving the Problems

- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

Random Restart (Monte-Carlo methods)

- Idea: restart hill climbing algorithm from random start configurations
- Repeat N times.
- If reasonable sampling of space, w high prob will find global max

Monte Carlo Descent

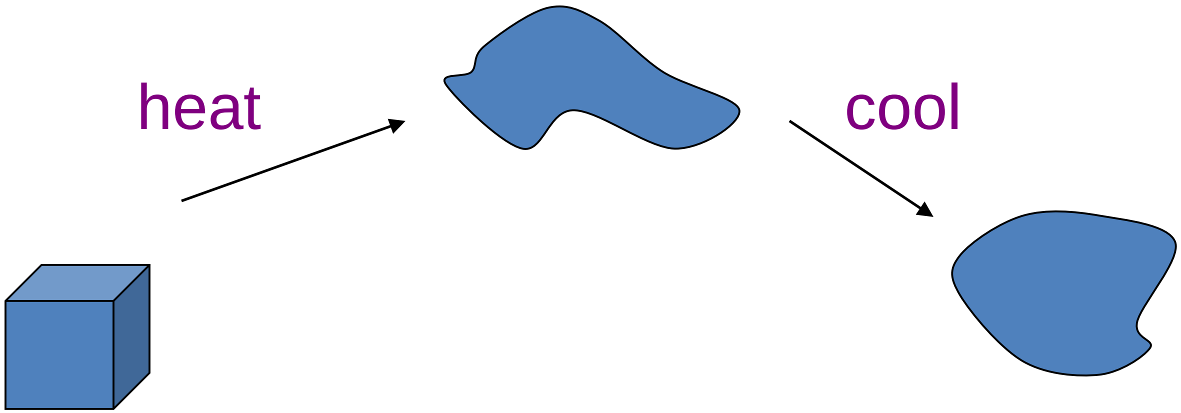
1) S ← initial state
2) Repeat k times:
   a) If GOAL?(S) then return S
   b) S' ← successor of S picked at random
   c) if $h(S') \leq h(S)$ then S ← S'
   d) else
      - Dh = h(S')-h(S)
      - with probability ~ $\exp(-Dh/T)$, where T is called the "temperature", do: S ← S' [Metropolis criterion]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

Simulated Annealing (2)

- Variant of hill climbing (maximize value)
- Tries to explore enough of the search space early on, so that the optimal solution is less sensitive to the start state
- May make some downhill moves before finding a good way to move uphill.

Simulated Annealing (2)

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

  $p = e^{(-\Delta E/kT)}$

  where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

Simulated Annealing: Intuition

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - ΔE/kT gets bigger
  - (-ΔE/kT) gets smaller
  - e^(-ΔE/kT) gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.
- ΔE is the change in the value of the objective function.
- Need an annealing schedule, which is a sequence of values of T: T_0, T_1, T_2, ...

Simulated Annealing Algorithm

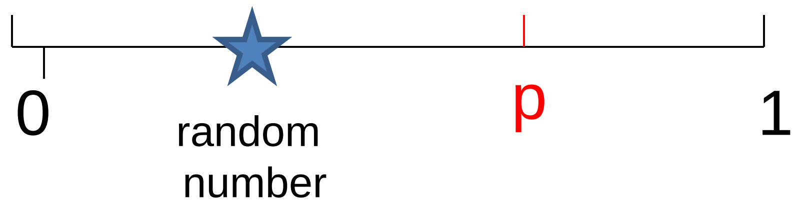
- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

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 $e^{\Delta E/T}$

Implementation: "Select successor with probability p"

- Select *next* with probability p



- Generate a random number
- If it's <= p, select *next*

Simulated Annealing Properties

- 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 a local optimum, the less likely you are to ever make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

Simulated Annealing Properties

- At a fixed “temperature” T, state occupation probability reaches the Boltzman distribution: https://en.wikipedia.org/wiki/Boltzmann_distribution

$$p_i = \frac{e^{-\varepsilon_i/kT}}{\sum_{j=1}^{M} e^{-\varepsilon_j/kT}}$$

- If T is decreased **slowly enough** (very slowly), the procedure will reach the best state.

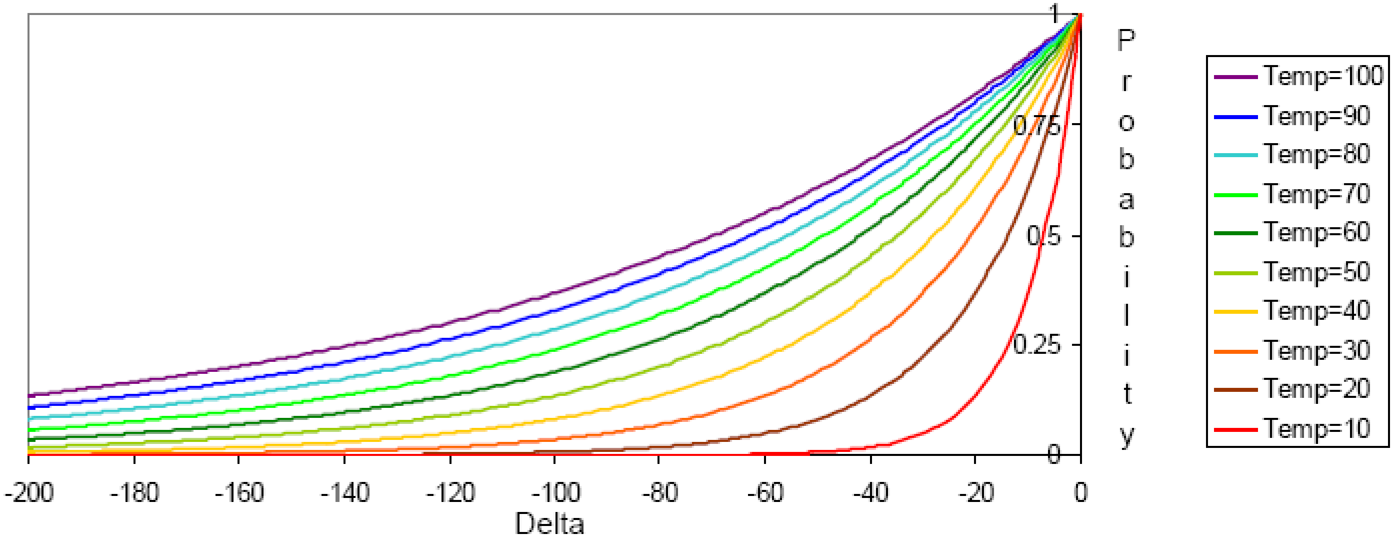
- **Slowly enough** has proven too slow for some researchers who have developed alternate schedules.

Simulated Annealing Schedules

- Acceptance criterion and cooling schedule

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)

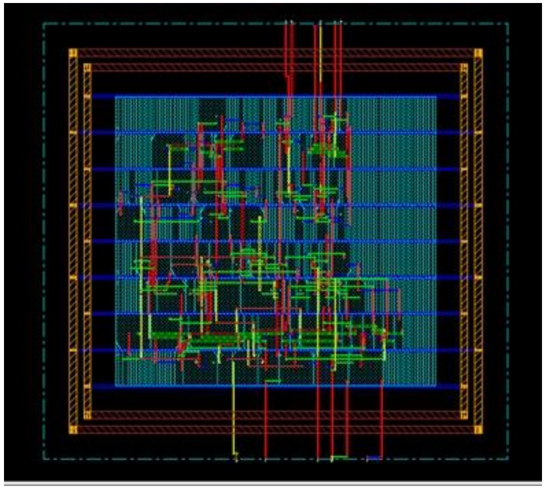
Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily "quench" the system

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory
  - Chemistry: molecular structure:

http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/



http://www.sciencedirect.com/science/article/pii/S0166128097001954

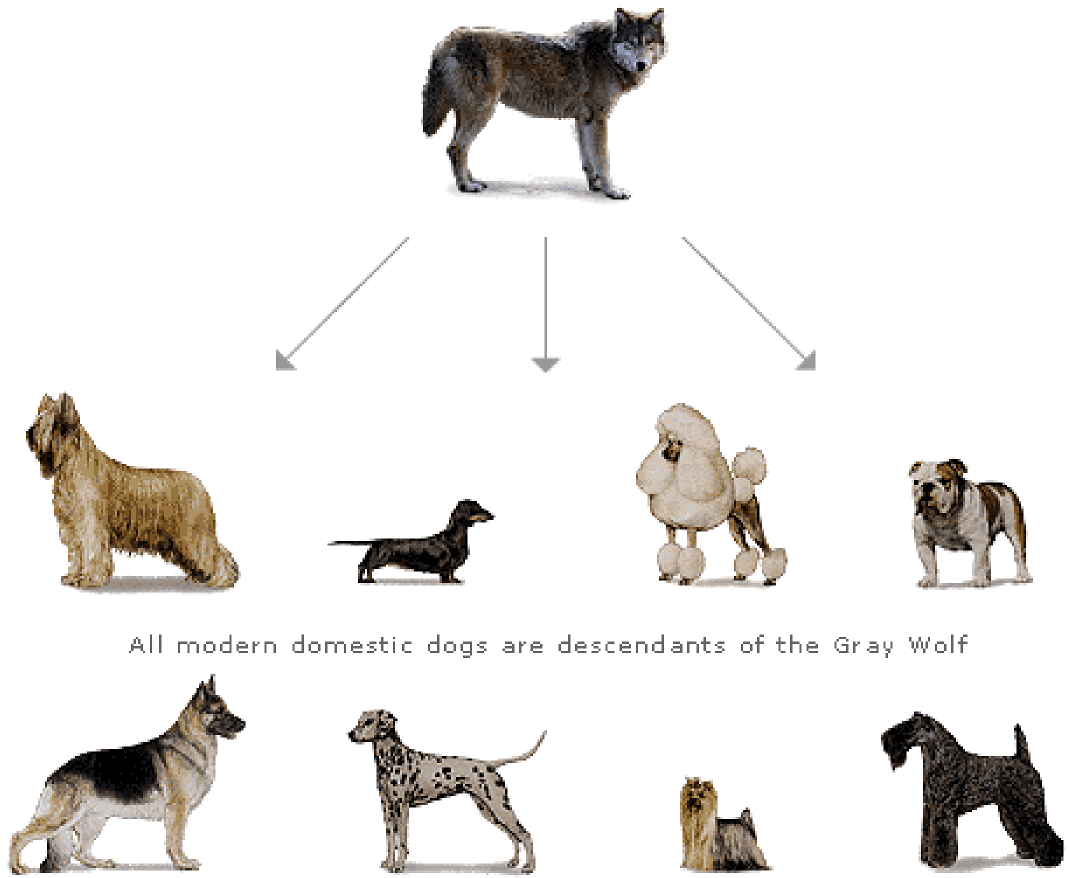
Local Beam Search

- Keeps more previous states in memory
  - Simulated annealing just kept one previous state in memory.
  - This search keeps k states in memory.

    - randomly generate k initial states
    - if any state is a goal, terminate
    - else, generate all successors and select best k
    - repeat

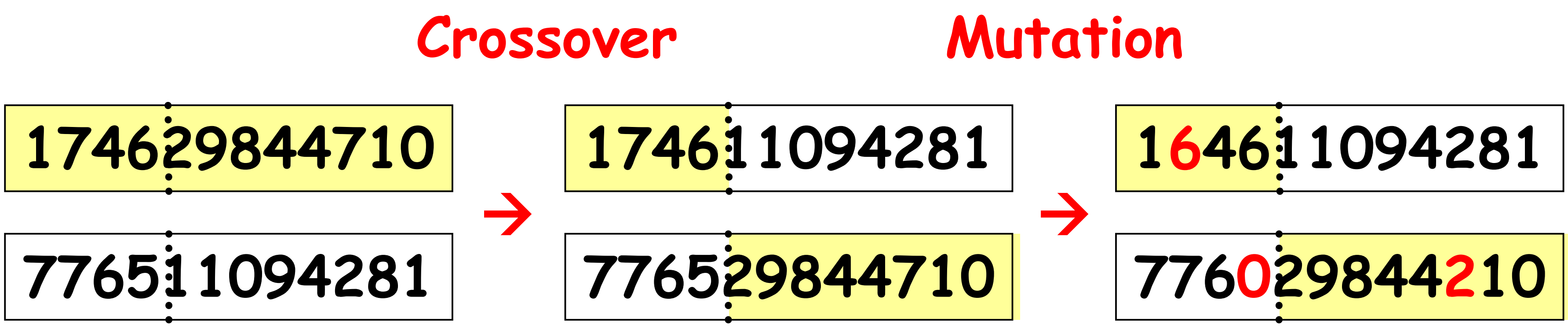
Explore Multiple Moves in Parallel?

- Evolution-inspired computation



Genetic Algorithms

- **Start with k random states (the *initial population)***
- **2.New states are generated by either**
  - **1.“*Mutation*” of a single state or**
  - **2.“*Sexual Reproduction*”: (combining) two *parent states* (selected proportionally to their *fitness) – crossover***
- **Encoding used for the “*genome*” of an individual strongly affects the behavior of the search**
- Similar (in some ways) to stochastic beam search



Genetic Algorithm

- Given: population P and fitness-function f
- repeat
  - newP ← empty set
  - for *i* = 1 to size(P)

    x ← RandomSelection(P,f)

    y ← RandomSelection(P,f)

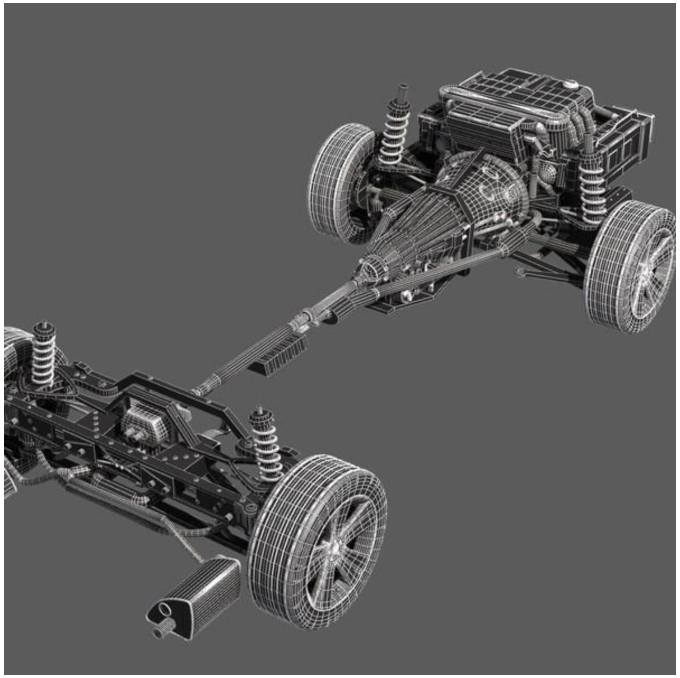
    child ← Reproduce(*x*,*y*)

    if (small random probability) then child ← Mutate(child)

    add *child* to newP
  - P ← newP
- until some individual is fit enough or enough time has elapsed
- return the best individual in P according to f

Example: Drive Train Design

- **Genes for:**
  - Number of Cylinders
  - **RPM: 1st-> 2nd**
  - **RPM 2nd-> 3rd**
  - **RPM 3rd-> Drive**
  - Rear end gear ratio
  - Size of wheels



- **A chromosome specifies a full drive train design**

GA: Reproduction

- **Reproduction by *crossover* selects genes from two parent chromosomes and creates two new offspring.**
- **To do this, randomly choose a crossover point (perhaps none).**
- **For child 1, everything before this point comes from the first parent and everything after from the second parent.**
- **Crossover looks like this (| is the crossover point):**

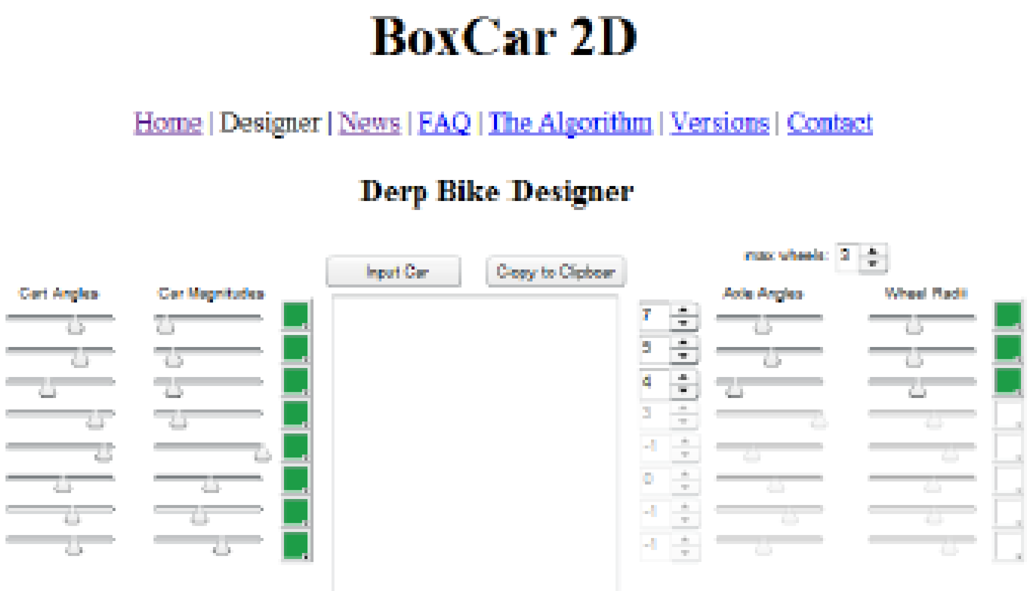
Chromosome 1	11001 | 00100110110
Chromosome 2	10011 | 11000011110
Offspring 1	11001 | 11000011110
Offspring 2	10011 | 00100110110

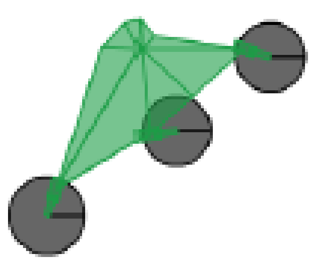
GA: Mutation

- **Mutation randomly changes genes in the new offspring.**
- **For binary encoding we can switch randomly chosen bits from 1 to 0 or from 0 to 1.**

Original offspring	1101111000011110
Mutated offspring	1100111000001110

GA Example: Cont'd





http://boxcar2d.com/

Summary: Search Techniques So Far

Search problems

Blind search

Heuristic search:
best-first and A*

Construction of heuristics

Variants of A*

Local search

Quick Review/Quiz

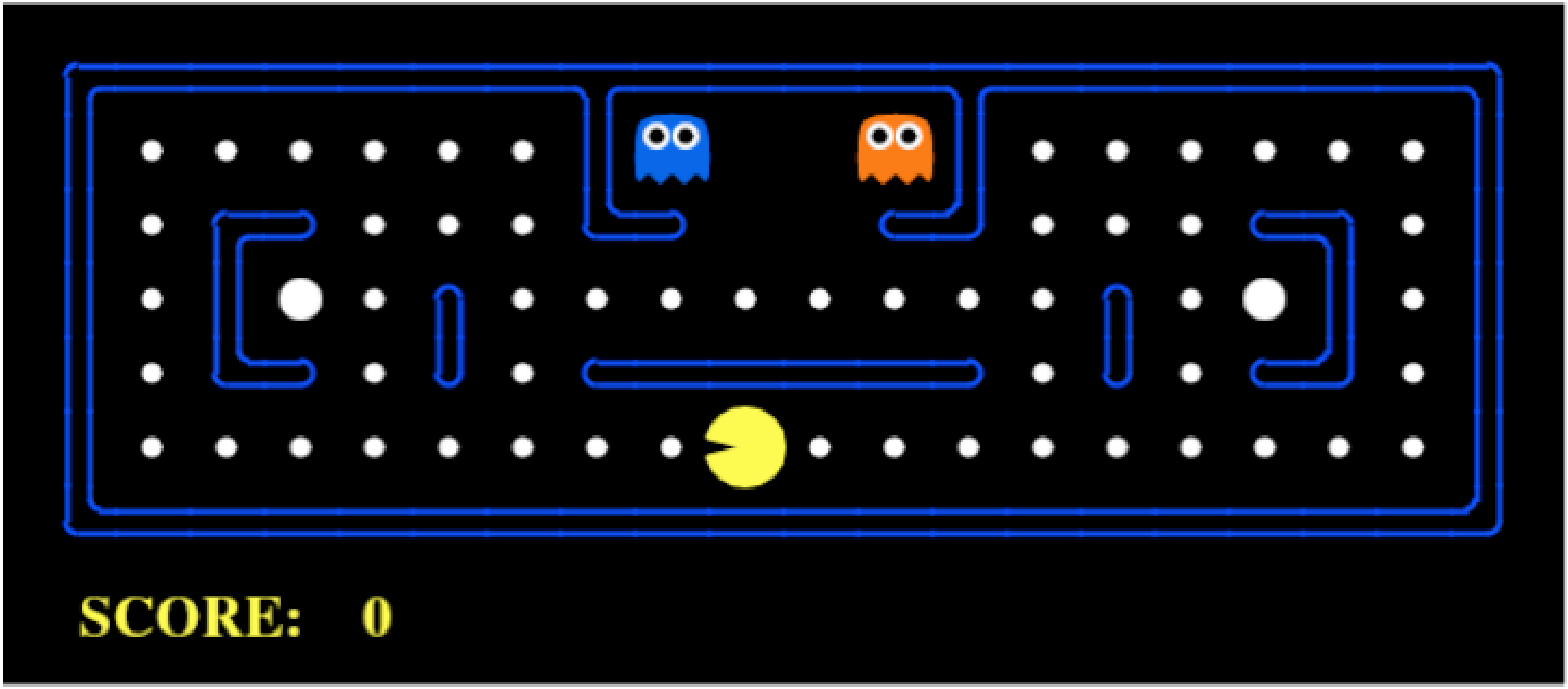
- What is a difference between a game state and search node?
- What do we lose if our A* heuristic not admissible?
- Why can't we use A* for "large" problems?

When to Use Search Techniques?

1) The search space is small, and
   - No other technique is available, or
   - Developing a more efficient technique is not worth the effort

2) The search space is large, and
   - No other available technique is available, and
   - There exist “good” heuristics

Thursday: Adversarial Search!



Announcements

- Reminder: **project 1 due Tomorrow (Wed) at 8pm**
  - Submit on Blackboard (enabled last week)
- Normal office hours tomorrow (Wed) 3-4, and Thursday 4-5.
- Thursday: start adversarial search;
  Project 2 will be assigned, due in ~1.5 weeks
- Next week: Instructor away next Tuesday, Rafi Haque will continue adversarial search.