

Announcements

Assignments:

- HW3 (online)
 - Due Tue 9/19, 10 pm
- P1: Search and Games
 - Due Mon 9/18, 10 pm
 - Submit to Gradescope early and as often as you like

Recitation:

- Last week to “shop around”
- Stay tuned to Piazza for informal recitation switch form

Outlook: HW4 due 9/26, Exam 1 9/28

Plan

Last Time

- Constraint Satisfaction Problems

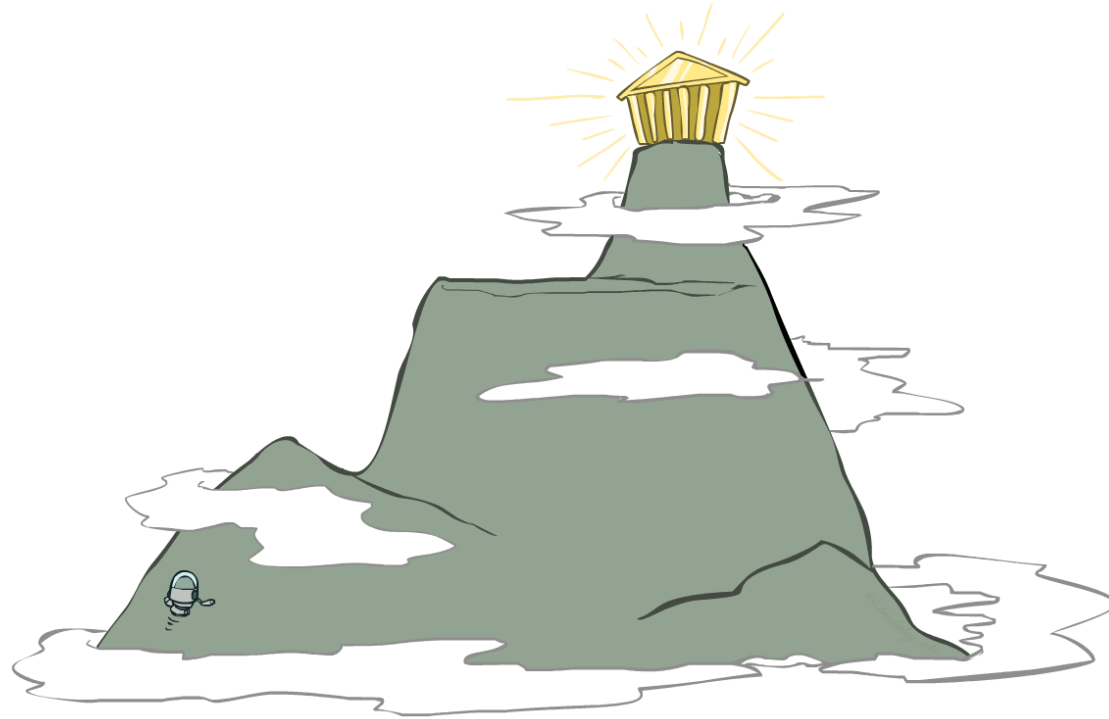
Today

- CSPs continued (MRV, LCV)
- Local Search

Back to CSPs Lecture

AI: Representation and Problem Solving

Local Search



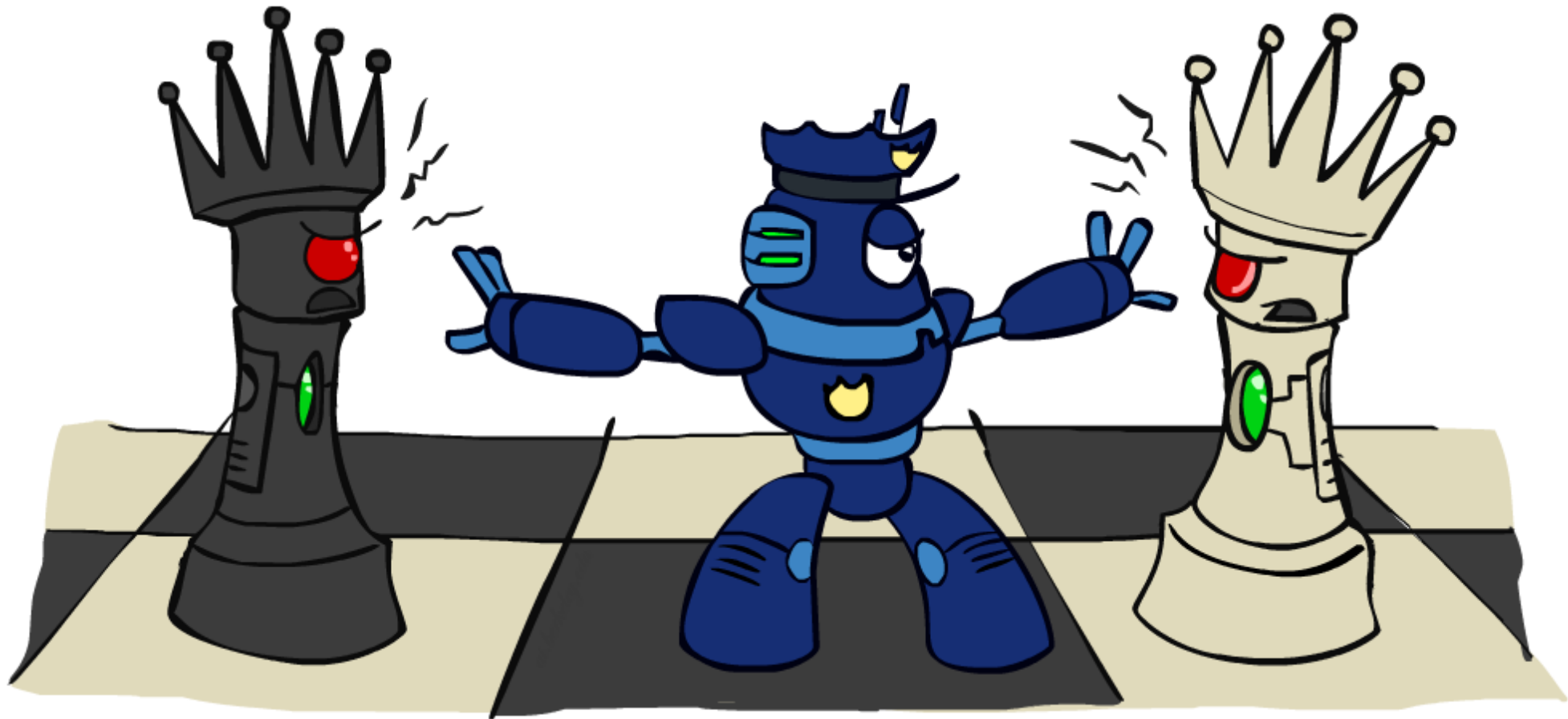
Instructors: Vincent Conitzer and Aditi Raghunathan

Slide credits: CMU AI, <http://ai.berkeley.edu>

Local Search

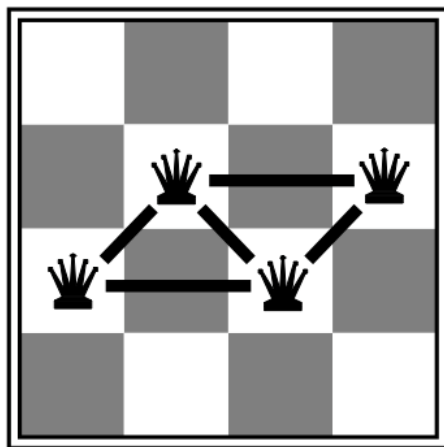
- Can be applied to identification problems (e.g., CSPs), as well as some planning and optimization problems
- For identification problems, we use a **complete-state formulation**, e.g., all variables assigned in a CSP (may not satisfy all the constraints)
- For planning problems, typically we make local decisions. e.g., not a plan all the way to the goal or not a deep search

Iterative Improvement for CSPs

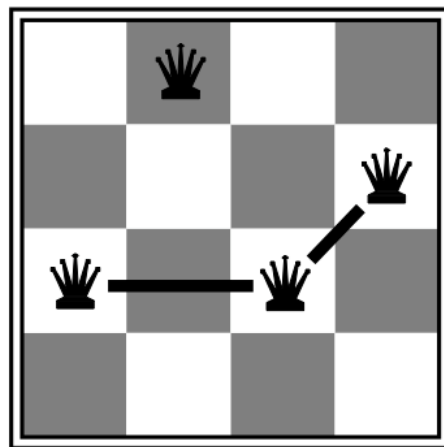
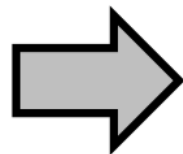


Iterative Improvement for CSPs

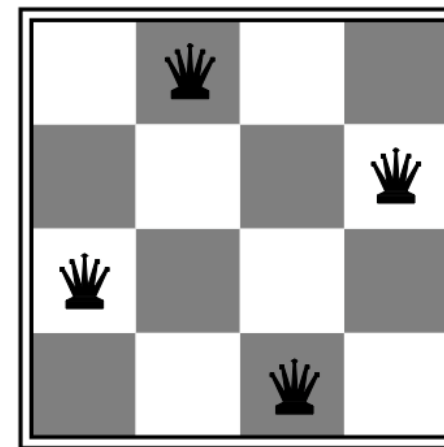
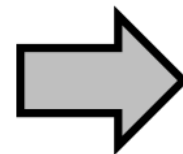
- Start with an arbitrary assignment, iteratively *reassign* variable values
- While not solved,
 - Variable selection: randomly select a conflicted variable
 - Value selection with **min-conflicts heuristic h** : Choose a value that violates the fewest constraints (break tie randomly)
- For n -Queens: Variables $x_i \in \{1..n\}$; Constraints $x_i \neq x_j, |x_i - x_j| \neq |i - j|, \forall i \neq j$



$h = 5$



$h = 2$

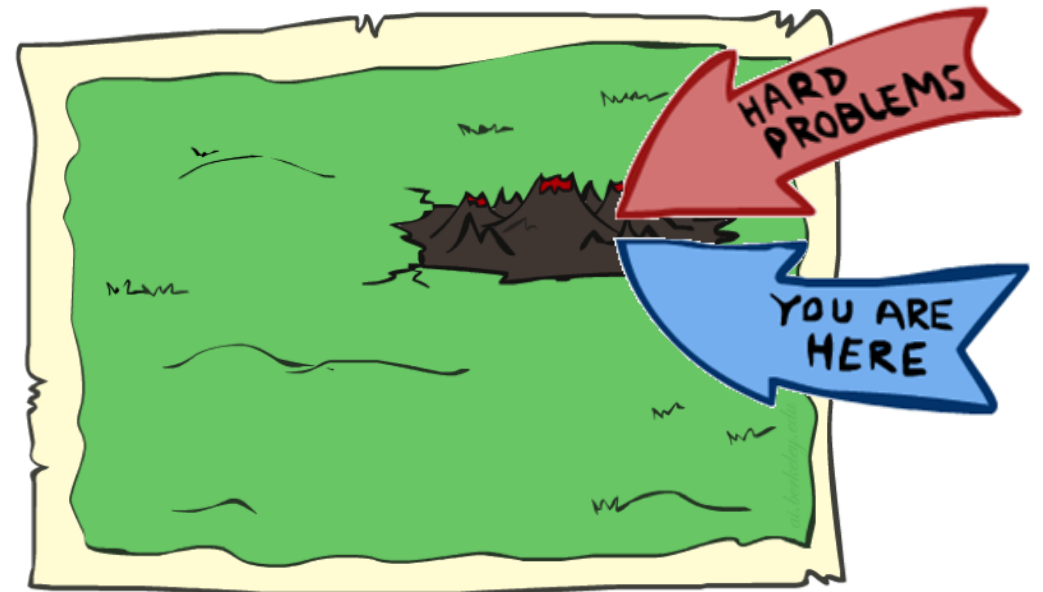
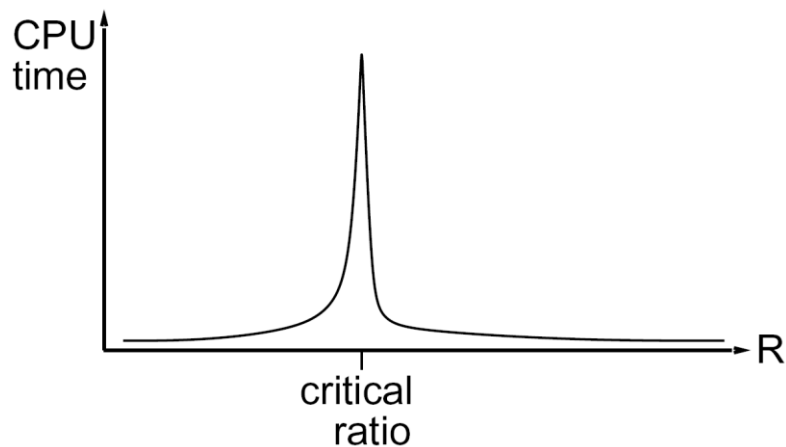


$h = 0$

Iterative Improvement for CSPs

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!
- Same for any randomly-generated CSP *except* in a narrow range of the ratio

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



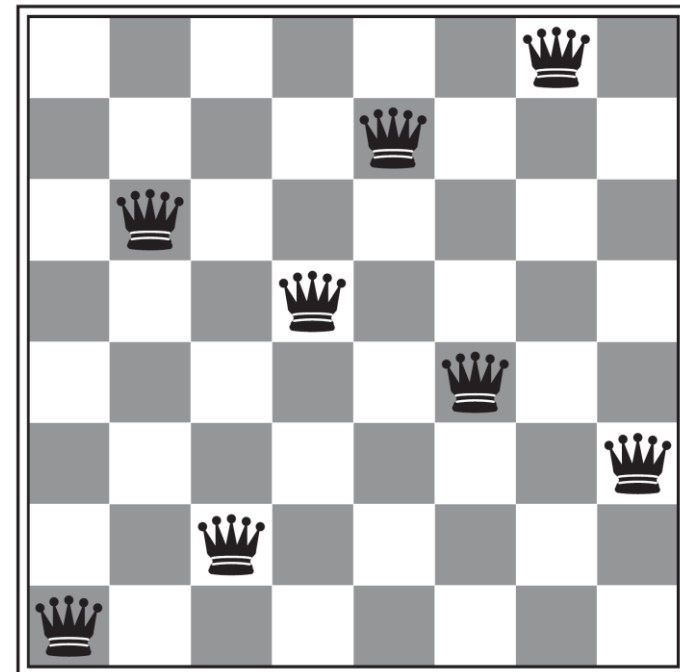
Local Search

- A local search algorithm is...
 - **Optimal** if it always finds a global minimum/maximum heuristic value

Will an iterative improvement algorithm for CSPs always find a solution?

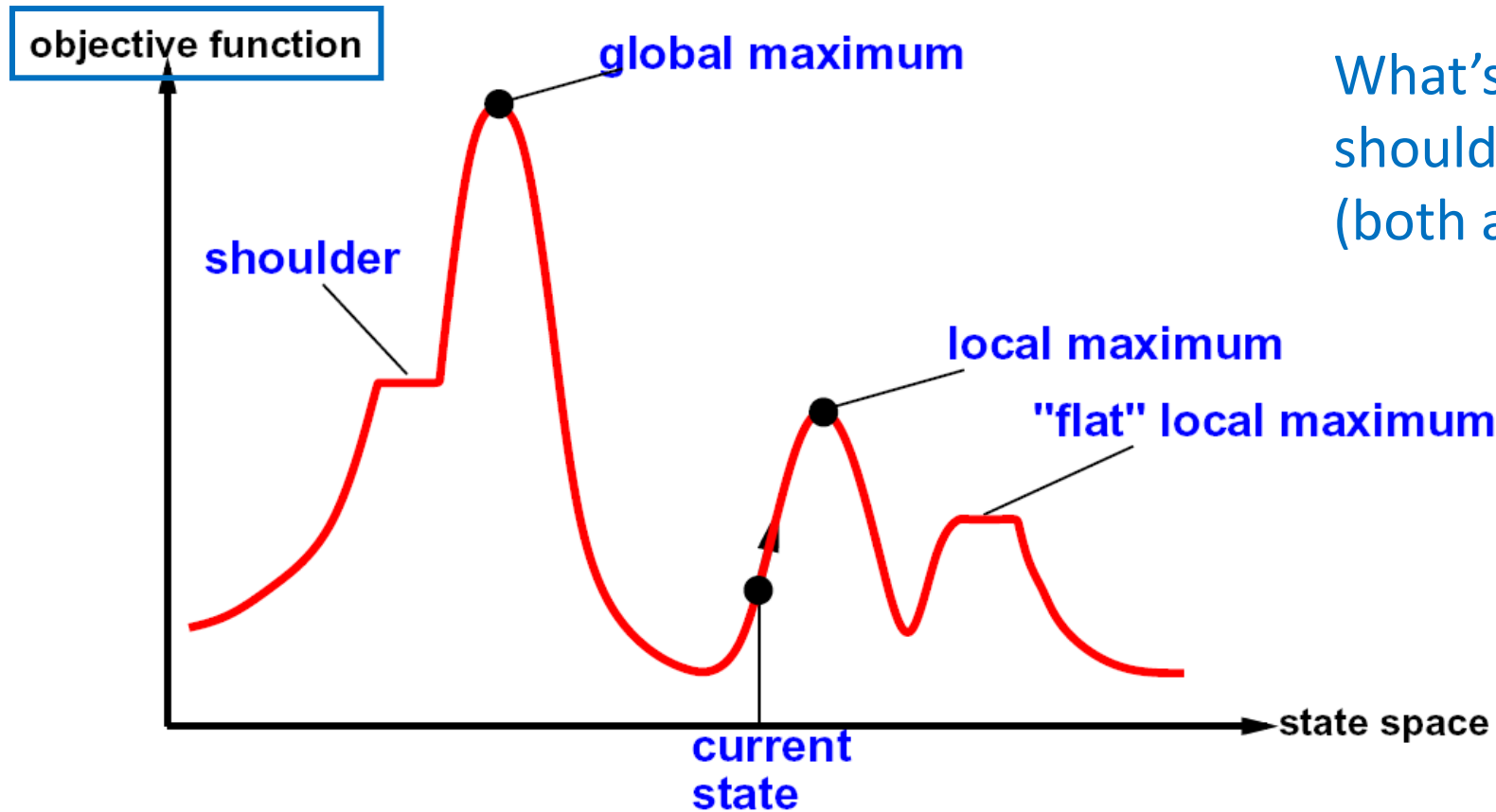
No! May get stuck in a local optimum

$h = 1$



State-Space Landscape

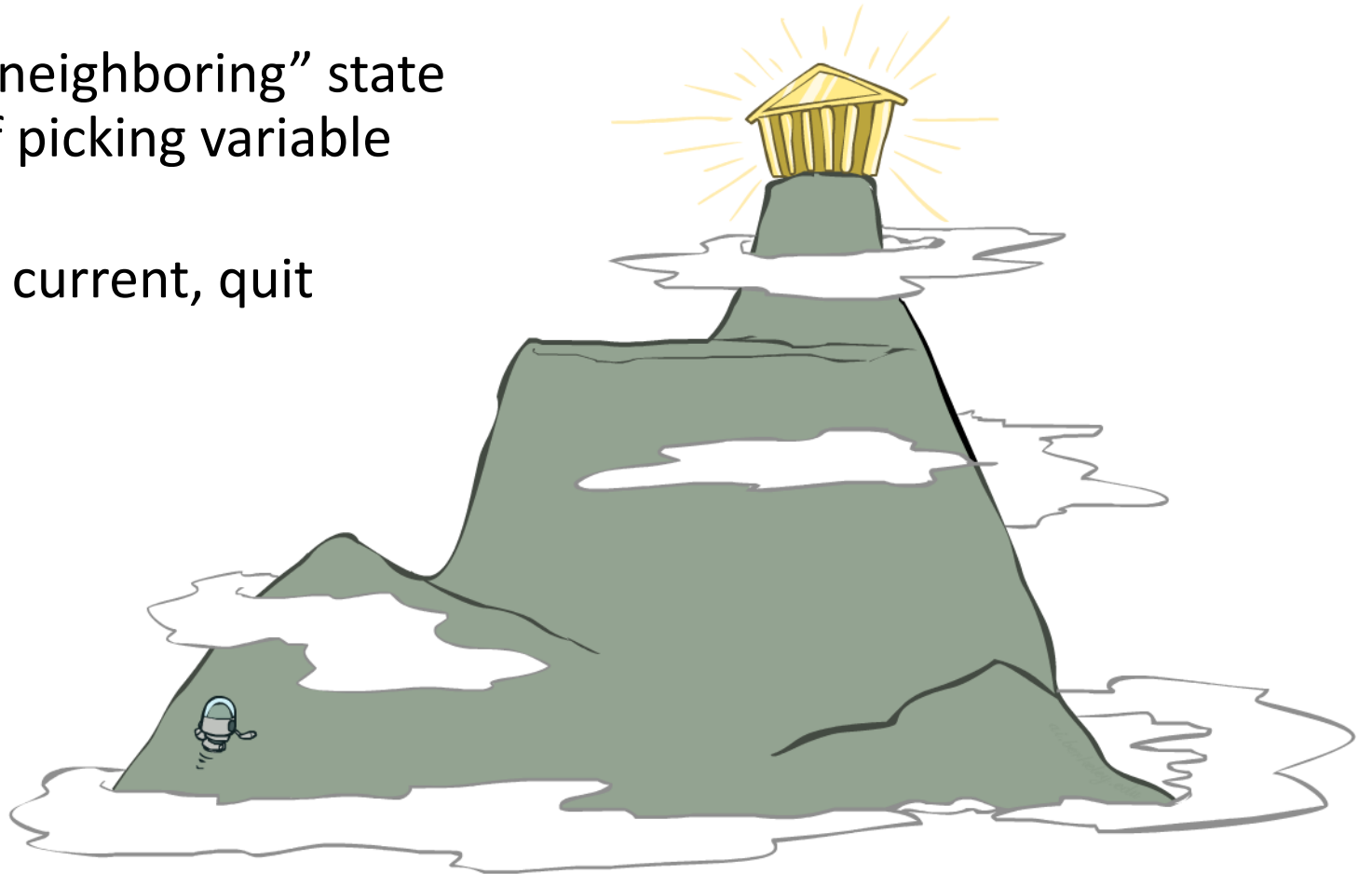
In identification problems, could be a function measuring how close you are to a valid solution, e.g., $-1 \times \text{\#conflicts}$ in n-Queens/CSP



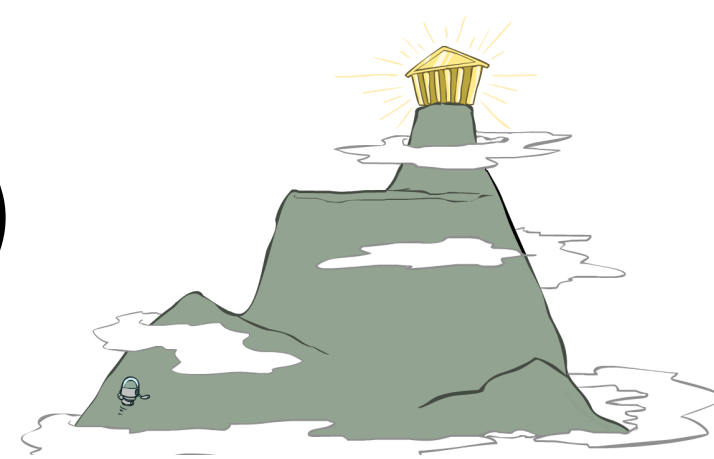
What's the difference between shoulder and flat local maximum (both are plateaux)?

Hill Climbing (Greedy Local Search)

- Simple, general idea:
 - Start wherever
 - Repeat: move to the best “neighboring” state (successor state) instead of picking variable randomly
 - If no neighbors better than current, quit



Hill Climbing (Greedy Local Search)



function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

current ← MAKE-NODE(*problem*.INITIAL-STATE)

loop do

neighbor ← a highest-valued successor of *current*

What if there is a tie?

Typically break ties randomly

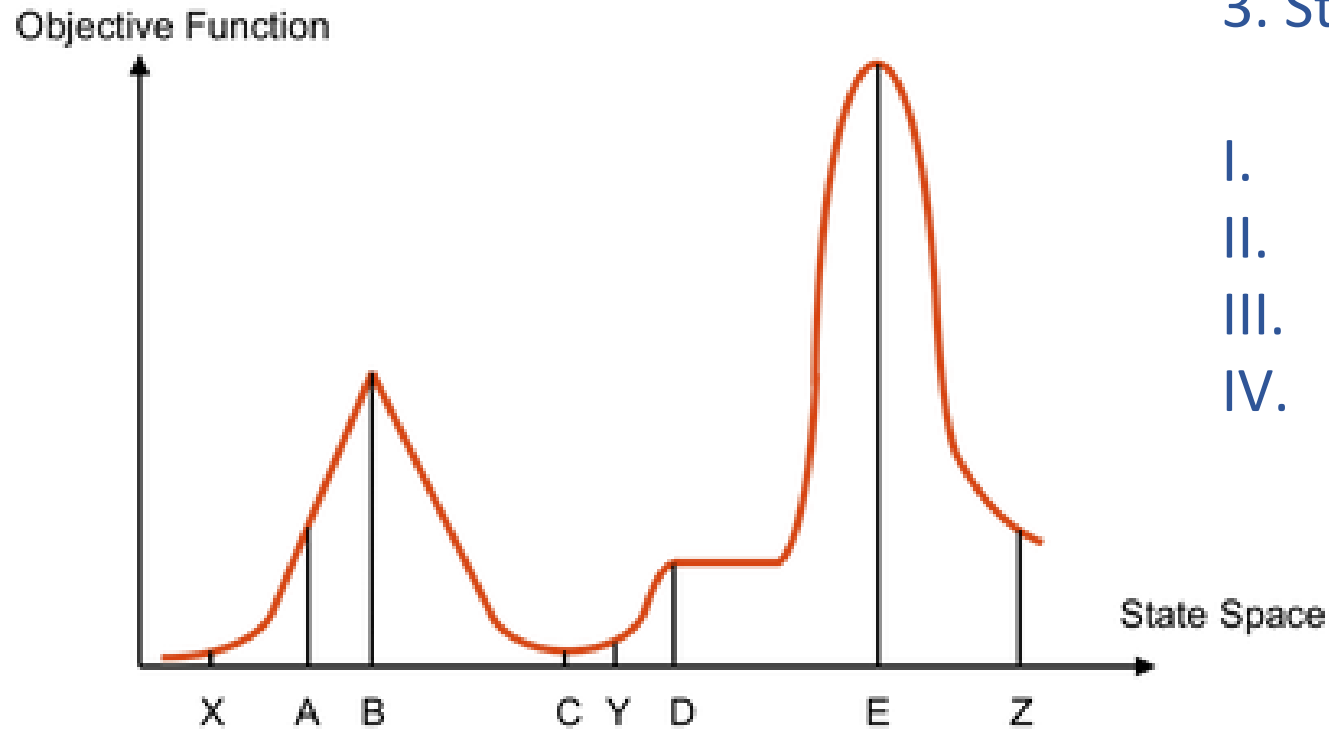
if *neighbor*.VALUE ≤ *current*.VALUE **then return** *current*.STATE

current ← *neighbor*

What if we do not stop here? Make a sideways move if “=”

- In 8-Queens, steepest-ascent hill climbing solves 14% of problem instances
 - Takes 4 steps on average when it succeeds, and 3 steps when it fails
- When allow for ≤100 consecutive sideways moves, solves 94% of problem instances
 - Takes 21 steps on average when it succeeds, and 64 steps when it fails

Poll 1: Hill Climbing



1. Starting from X, where do you end up?
2. Starting from Y, where do you end up?
3. Starting from Z, where do you end up?

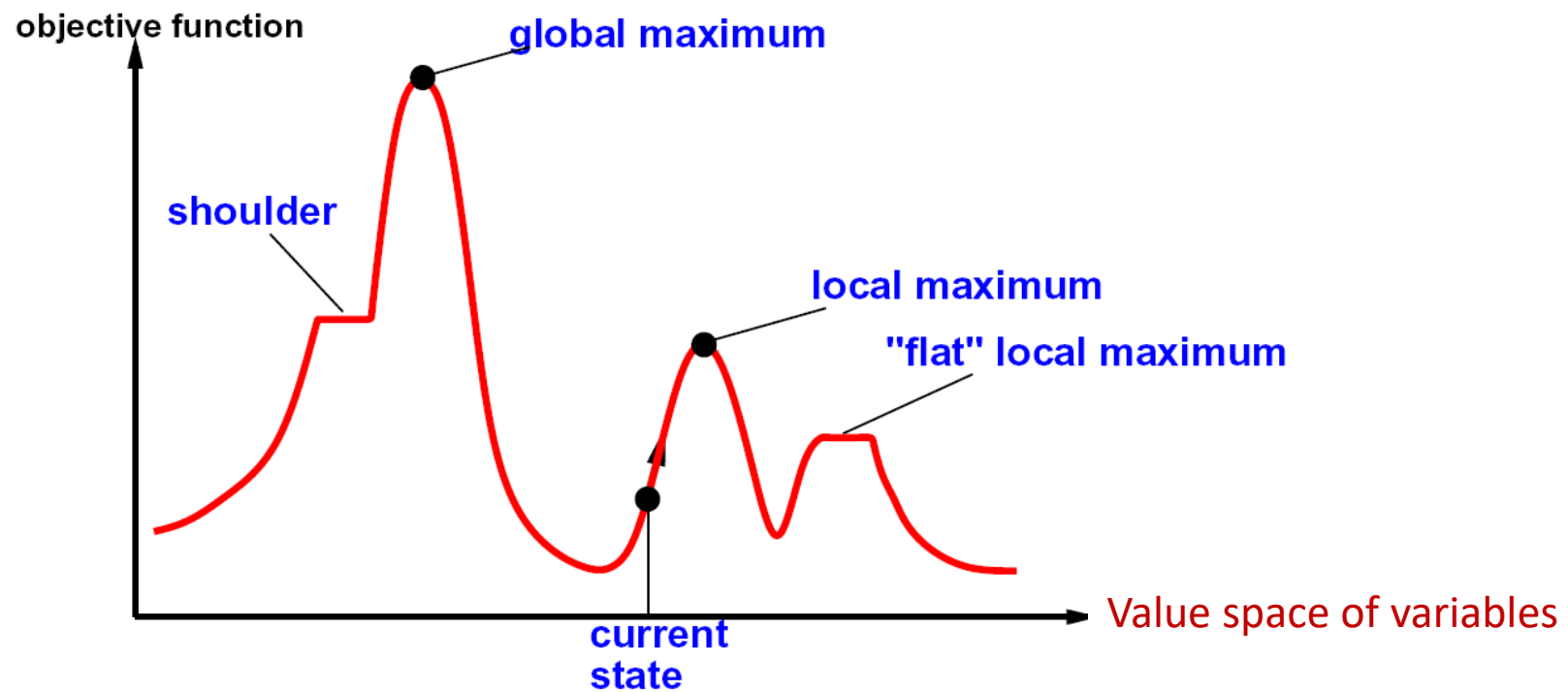
- I. $X \rightarrow A, Y \rightarrow D, Z \rightarrow E$
- II. $X \rightarrow B, Y \rightarrow D, Z \rightarrow E$
- III. $X \rightarrow B, Y \rightarrow E, Z \rightarrow E$
- IV. I don't know

Variants of Hill Climbing

- Random-restart hill climbing
 - “If at first you don’t succeed, try, try again.”
 - What kind of landscape will random-restarts hill climbing work the best?
- Stochastic hill climbing
 - Choose randomly from the uphill moves, with probability dependent on the “steepness” (i.e., amount of improvement)
 - Converge slower than steepest ascent, but may find better solutions
- First-choice hill climbing
 - Generate successors randomly (one by one) until a better one is found
 - Suitable when there are too many successors to enumerate

Variants of Hill Climbing

- What if variables are continuous, e.g. find $x \in [0,1]$ that maximizes $f(x)$?
 - Gradient ascent
 - Use gradient to find best direction
 - Use the magnitude of the gradient to determine how big a step you move

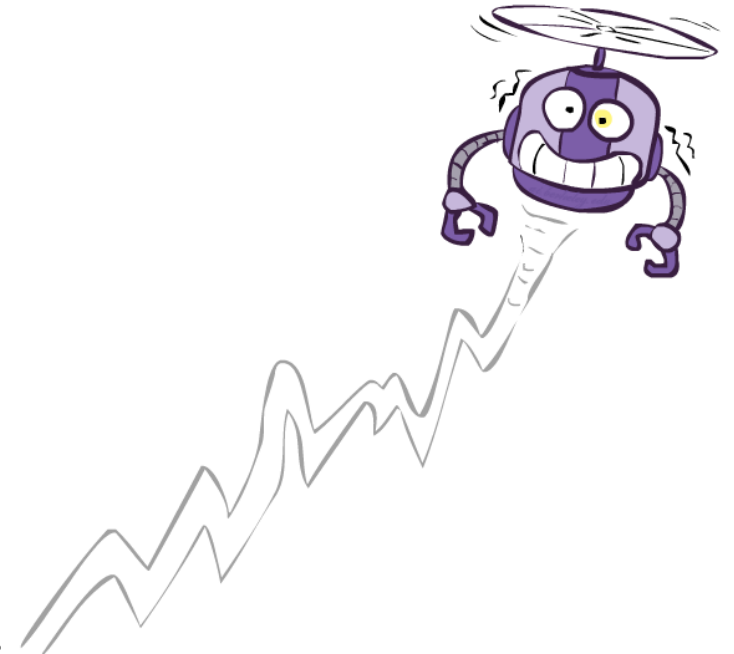


Random Walk

- Uniformly randomly choose a neighbor to move to
 - Save the best you've seen so far
 - Stop after K moves
-
- What happens to the solution as K increases?

Simulated Annealing

- Combines random walk and hill climbing
- Inspired by statistical physics
- Annealing – Metallurgy
 - Heating metal to high temperature then cooling
 - Reaching low energy state
- Simulated Annealing – Local Search
 - Allow for downhill moves and make them rarer as time goes on
 - Escape local maxima and reach global maxima



Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ *next*.VALUE $-$ *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Control the change of
temperature T (\downarrow over time)

Almost the same as hill climbing
except for a *random* successor

Unlike hill climbing, move
downhill with some prob.

Poll 2:

Which of the following will make it more likely that we'll take a downward step?

- A. Decrease T , decrease ΔE
- B. Decrease T , increase ΔE
- C. Increase T , decrease ΔE
- D. Increase T , increase ΔE

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Poll 2:

Which of the following will make it more likely that we'll take a downward step?

- A. Decrease T , decrease ΔE
- B. Decrease T , increase ΔE
- C. Increase T , decrease ΔE
- D. Increase T , increase ΔE

ΔE is negative but should be close to 0,
T should be big because of E's negative

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

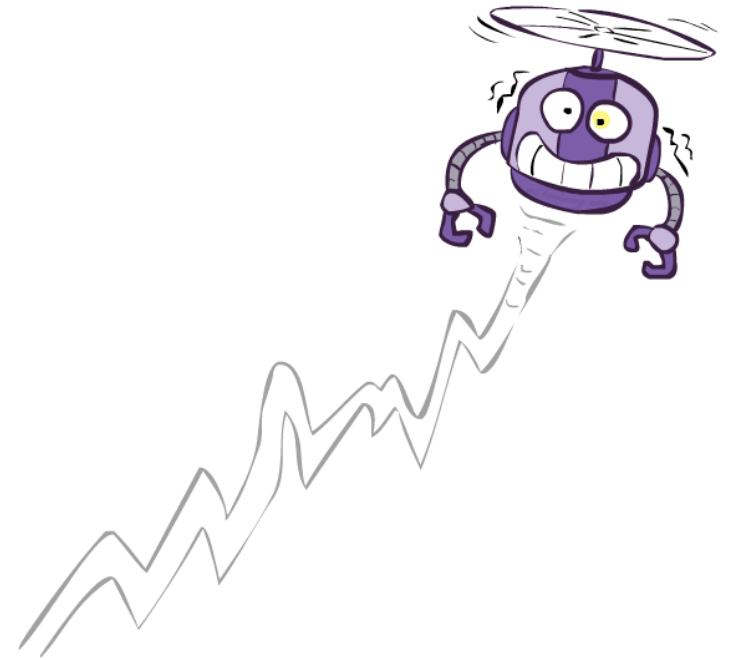
$\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Simulated Annealing

- $P[\text{move downhill}] = e^{\Delta E/T}$
 - Bad moves are more likely to be allowed when T is high (at the beginning of the algorithm)
 - Worse moves are less likely to be allowed
- Guarantee: If T decreased slowly enough, will converge to optimal state!
- But! In 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



Summary: Local Search

- Maintain a constant number of current nodes or states, and move to “neighbors” or generate “offspring” in each iteration
 - Do not maintain a search tree or multiple paths
 - Typically, do not retain the path to the node
- Advantages
 - Use little memory
 - Can potentially solve large-scale problems or get a reasonable (suboptimal or almost feasible) solution