Lecture 5
Local Search

Thanapon Noraset
Faculty of ICT, Mahidol University

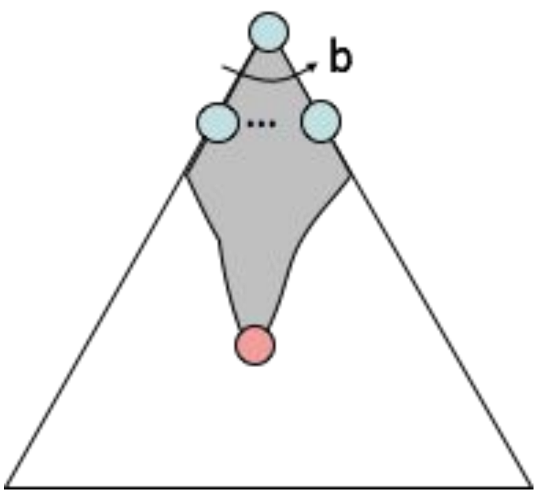
Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

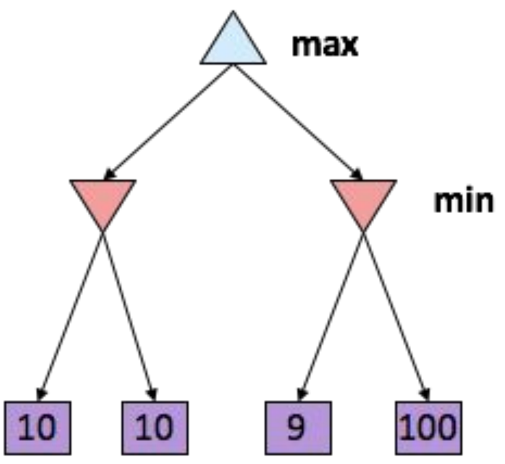
Agenda

- Local Search
  - Hill Climbing
  - Random restart
  - Local Beam Search
- Simulated Annealing
- Genetic Algorithms

Exploring the state space

- Search algorithms so far explores the state space systematically
- They also keep track of multiple alternative paths (i.e. frontiers, legal moves)





Local Search

Advantages of Local Search

- Work with problems where a solution is single state, not a path.

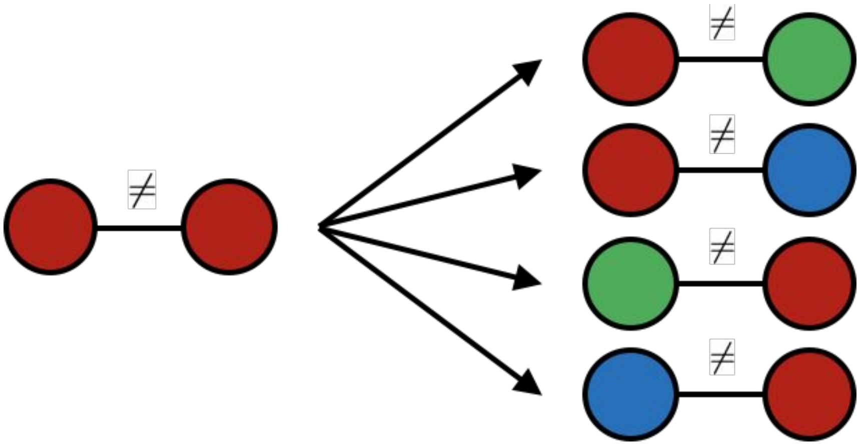
- Use very little memory (usually a constant).

- Often find reasonable solutions in a large or infinite state space (continuous).

Local Search: General Idea

From a current node

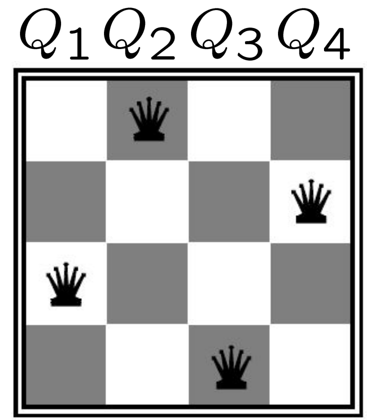
- Check local changes ➔ Neighbors in state space
- Select a better neighbor until you cannot find any better



Objective Functions

- Local search algorithms need to compare "scores" of different options
- Objective function or heuristic function provide these score
- The goal states have the maximum (or minimum) score
- For example
  - 8-Queens: number of attacking pairs
  - Eval function: average root mean square (vs actual minimax values)

Example: N-Queens

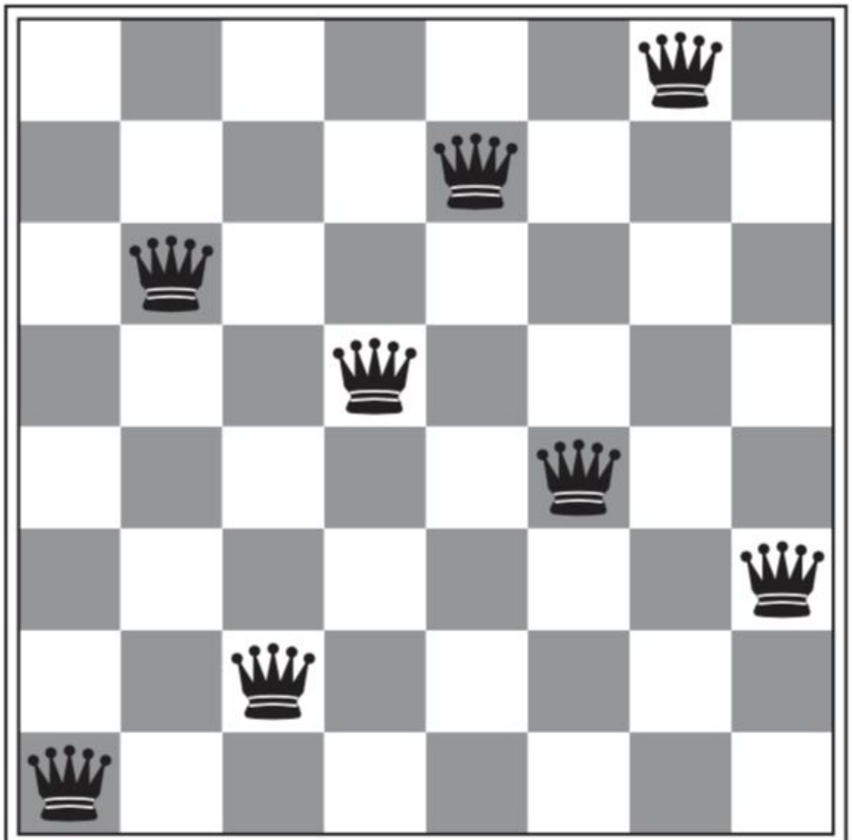


- Start with all the queens
- Neighbor states = all possible States to move a single queen
  - How many for N queens?

- Heuristic function: the number of pairs of queens that are attacking each other

Example: N-Queens

How many pairs are attacking each other?



Hill Climbing Search

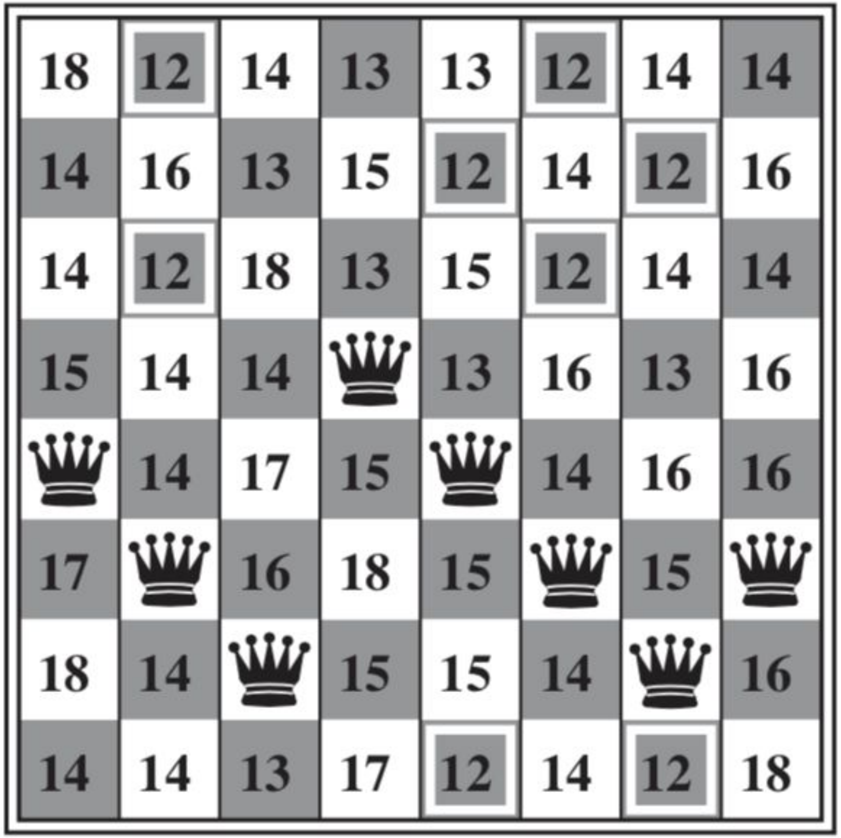
- General Idea:
  - Start at a random state
  - Keep moving to the best neighbor
  - If no better neighbors, exit
- Advantage:
  - Simple and memory efficient

---

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

Example: N-Queens

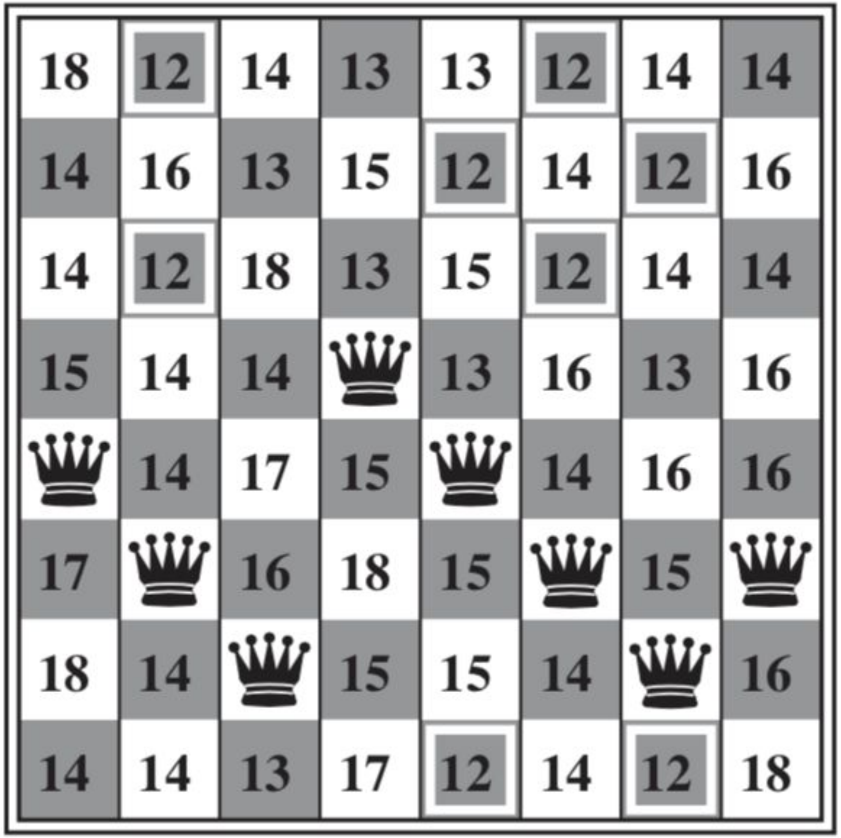


Current cost = ?

Neighbor states and their heuristic costs

Best move cost = ?

Example: N-Queens



Current cost = 17

Neighbor states and their heuristic costs

Best move cost = 12

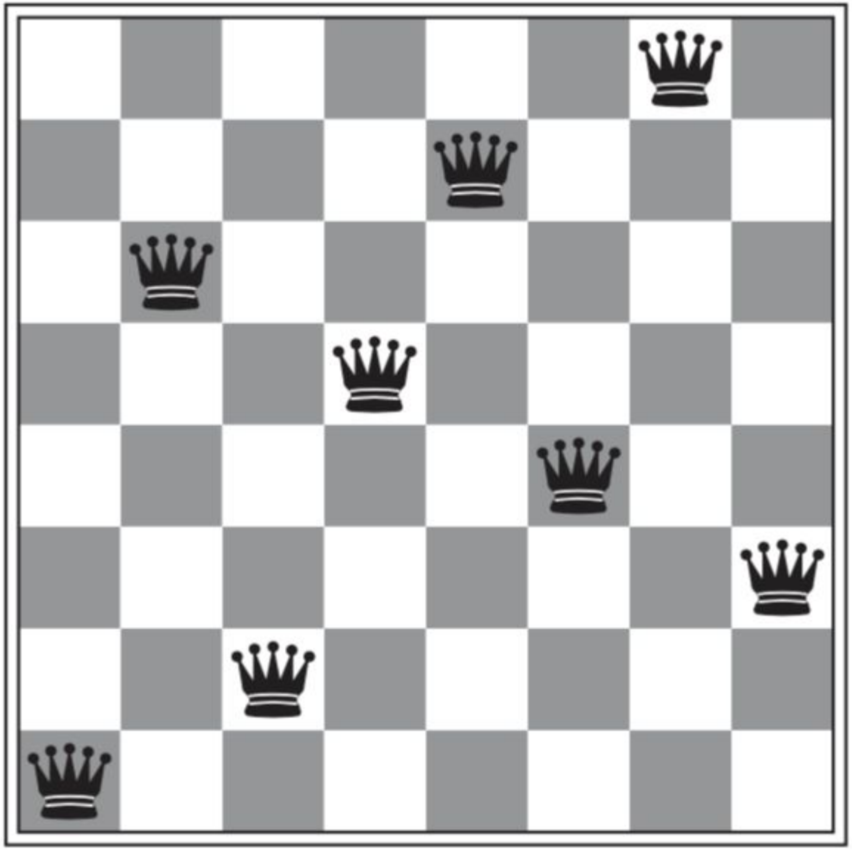
Hill Climbing Search

- Not complete nor optimal
- Get stuck in local optimal states and plateaus

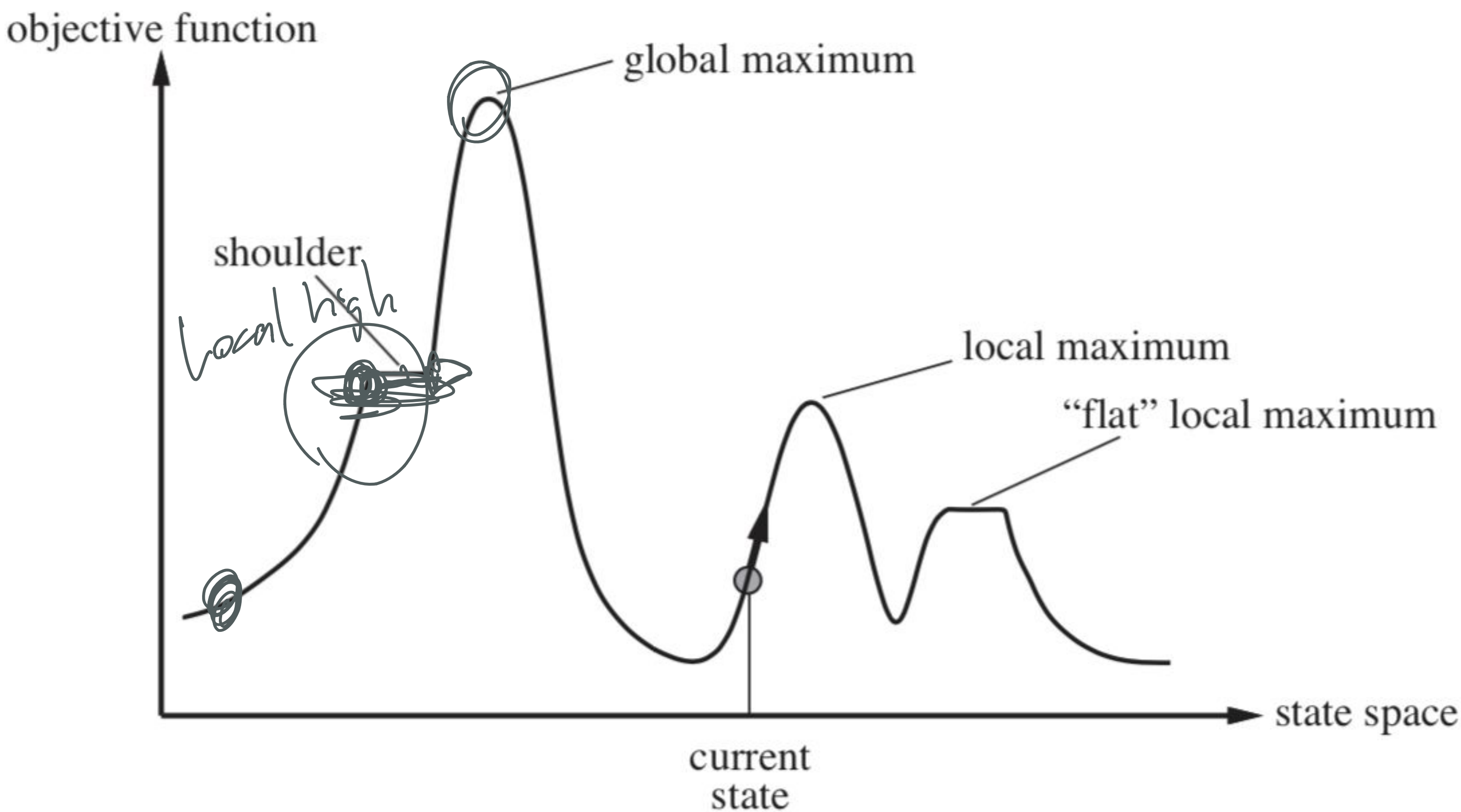
- Only solve 14% of 8-Queen starting states
- Very fast, on average it takes
  - 4 steps when solvable
  - 3 steps when stuck

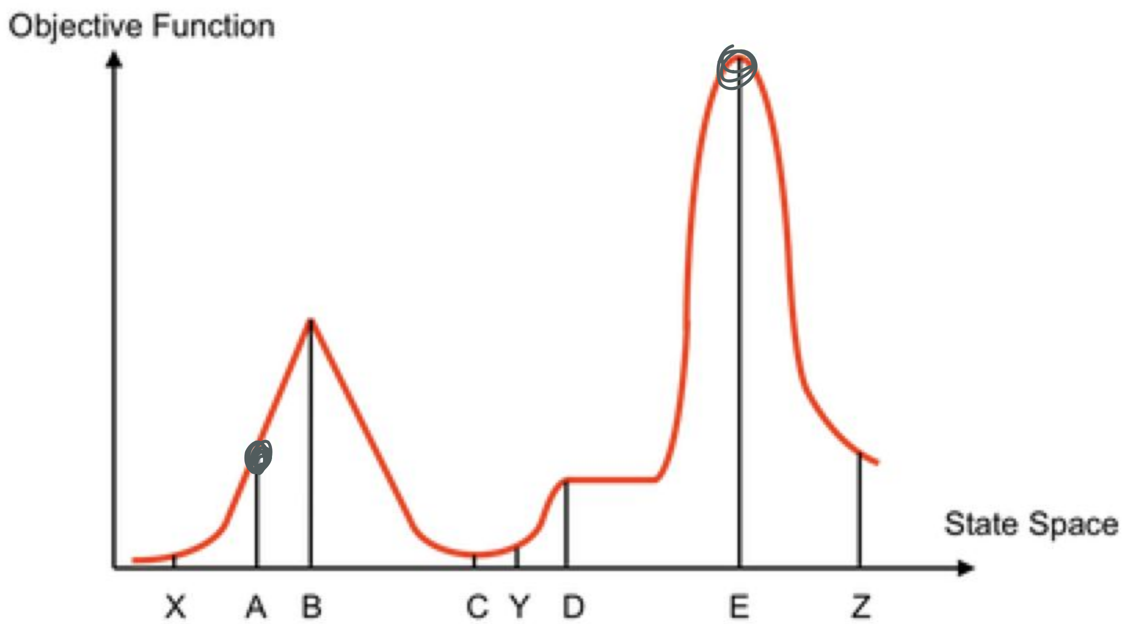
Hill Climbing Search: Stuck

Every neighbor has higher cost



State-space Landscape





Where do you end up if you start from:

- X, Y, or Z

B, E : local optimal – best among it's neighbor
some are global

stop B D E

Hill Climbing Search: Improvement (1 / 3)

Sideway: Keep moving on on a plateau

- Need to limit how many sideway steps
  - Why?
- 8-Queens (limit = 100 moves):
  - Improve from 14% to 94% success rate
  - Increase average steps to 21 / 64

Hill Climbing Search: Improvement (2 / 3)

Randomness comes to rescue!

- Stochastic choice:
  Randomly choose from random uphill moves

- Random-restart:
  “If at first you don’t succeed, try, and try again”
  - On average, it takes 7 restarts to solve 8-Queens

Hill Climbing Search: Improvement (3 / 3)

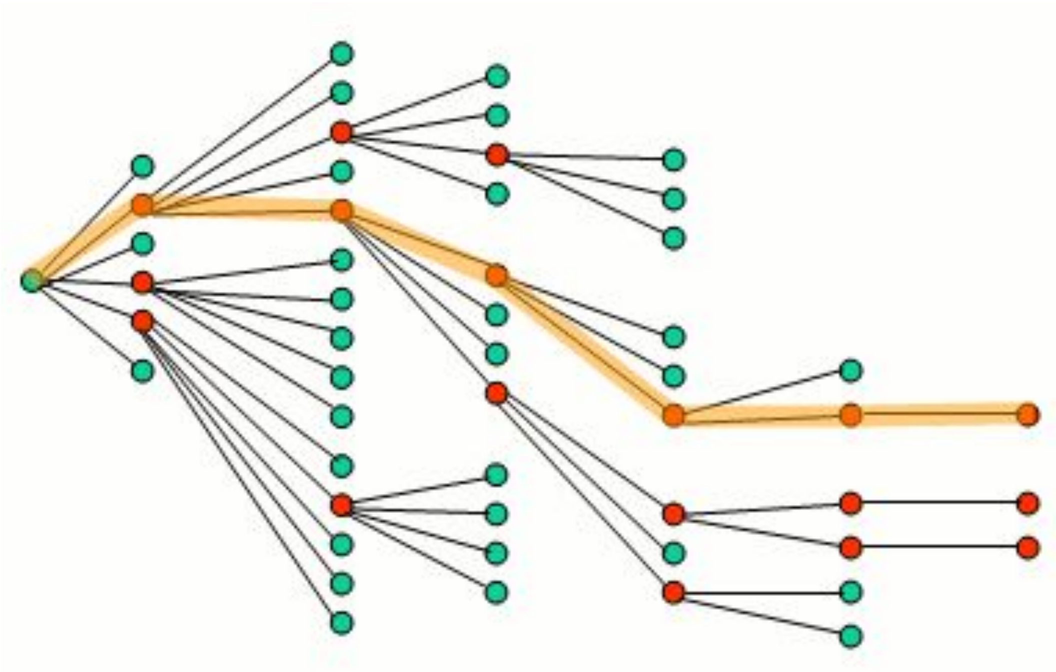
Local Beam Search

: bit complicated
no need to do in hw

- Instead of a single current state, we can keep track *k* best states at once
- A each step pick the best *k* neighbors (regardless of the branch)

- Commonly used in speech recognition and machine translation

Local Beam Search: Example



"Come over here, the grass is greener!"

Image from https://edux.pjwstk.edu.pl/mat/2144/lec/wyklad6/w6.htm

Simulated Annealing

Simulated Annealing

- Hill climbing search gets stuck because it *never* makes downhill moves

- Moving randomly is complete (if we wait long enough), it is too inefficient

- Simulated Annealing combines both
  - Escape local maxima by allowing downhill moves
  - But make less downhill moves as time goes on

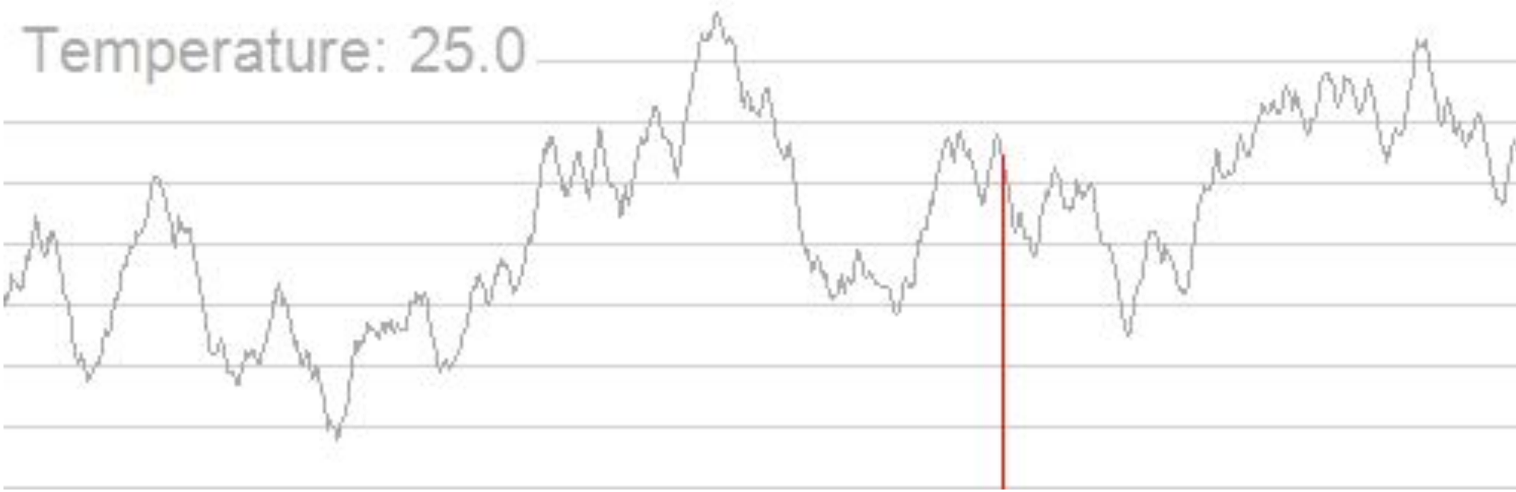
Simulated Annealing: Algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to “temperature”

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

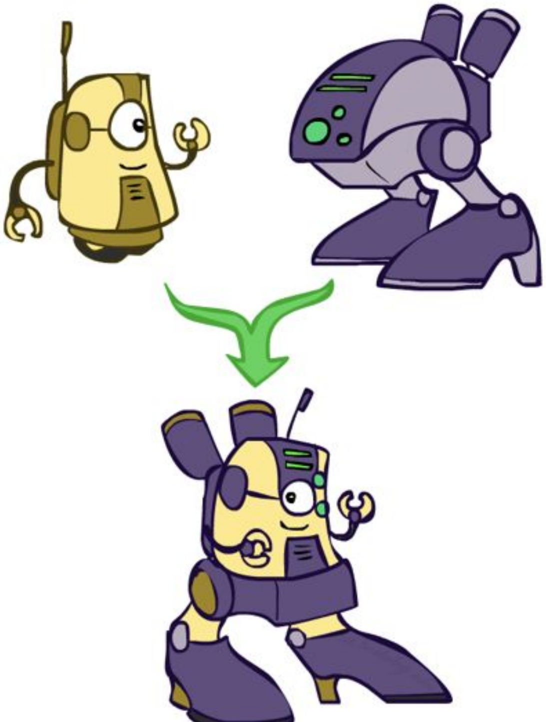
Simulated Annealing

- Lower temperature ⇒ Fewer "downhill" moves
- If T decreased slowly enough, it will converge to optimal state!
- ... theoretically ... In practice, it can stuck too :-(



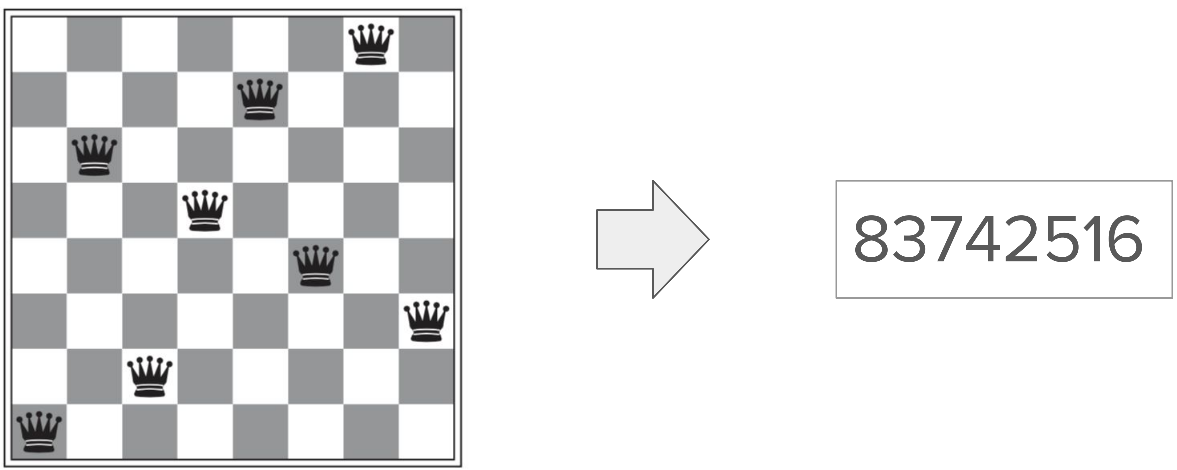
https://en.wikipedia.org/wiki/Simulated_annealing

Genetic Algorithm

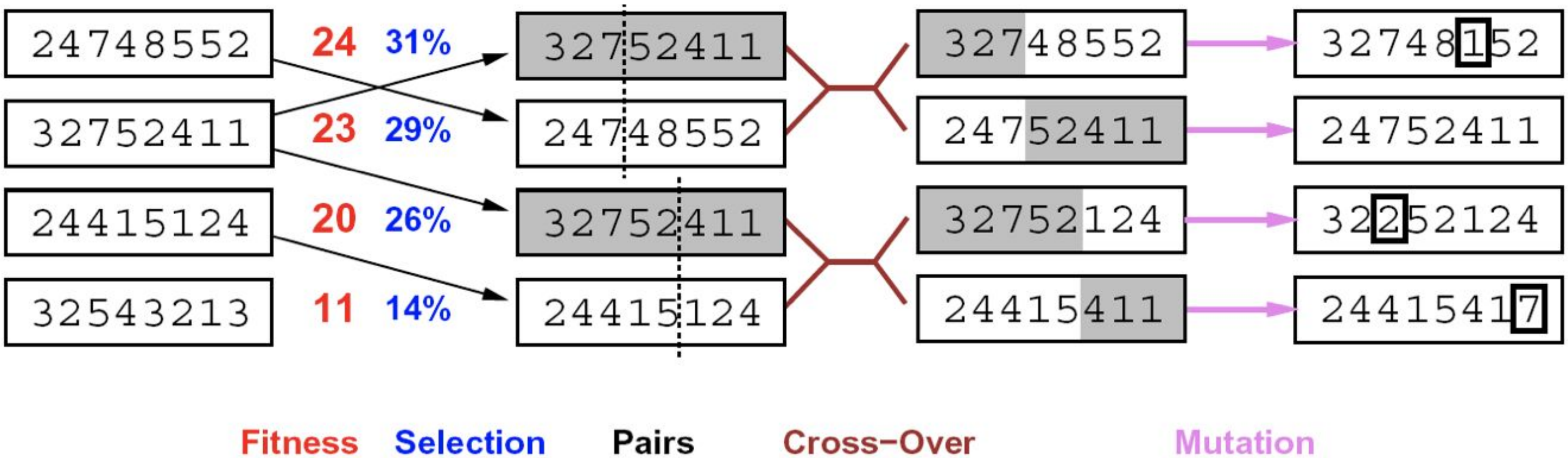


Genetic Algorithm

- Sometimes it is helpful to move far away, rather than jittering around the same place.
- Genetic algorithm defines a way to “jump” around using “natural selection” analogy
- Need to encode a state into a sequence



Genetic Algorithm: Idea



- Start with a random population (fix size)
    - State ➔ Individual (gene)
- States with higher fitness function will have higher chance to reproduce
- Reproduction by crossover and mutation

Genetic Algorithm

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

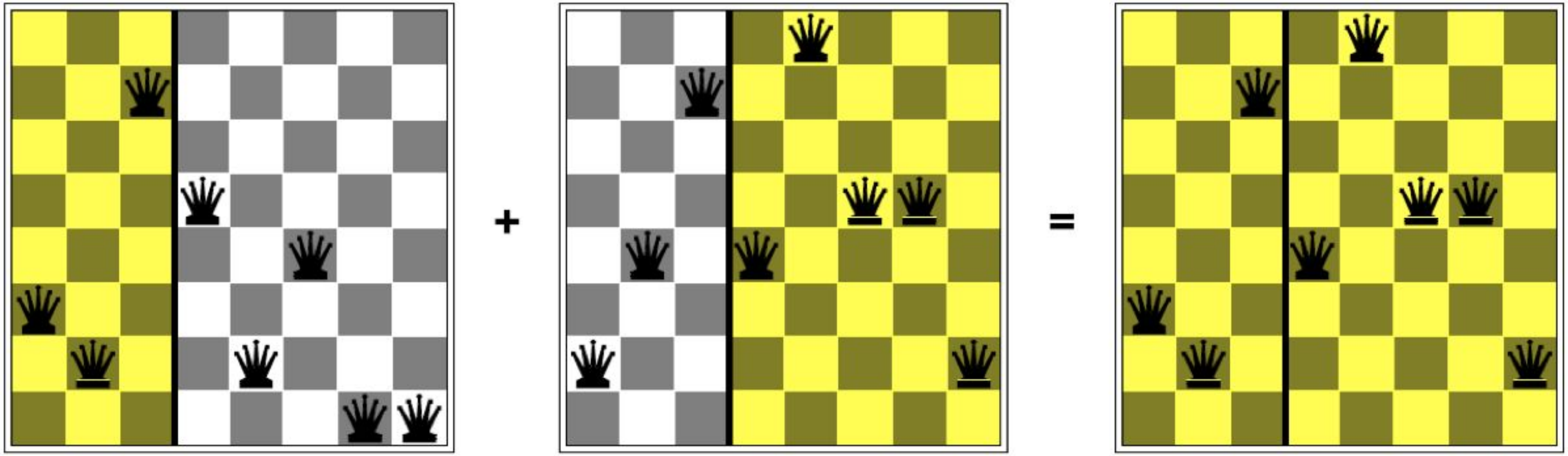
  repeat
    new_population ← empty set
    for i = 1 **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNESS-FN)
      y ← RANDOM-SELECTION(*population*, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *new_population*
    population ← *new_population*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to FITNESS-FN

---

function REPRODUCE(x, y) **returns** an individual
  inputs: x, y, parent individuals

  $n \leftarrow$ LENGTH(x); $c \leftarrow$ random number from 1 to n
  return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c + 1, n$))

Genetic Algorithm: N-Queens



Genetic Algorithm

- Similar to stochastic local beam search + crossover operation
- But sometimes crossover provides no advantage at all
  - Crossover only makes sense when we can make independent improvement on a block of the gene code

Summary

- Local search algorithms
  - Used when a solution is a state
  - Start from a random state and move to a neighbor state
- Algorithms:
  - Hill Climbing Search + its variants
  - Simulated Annealing
  - Genetic Algorithms

Quiz

See MyCourse