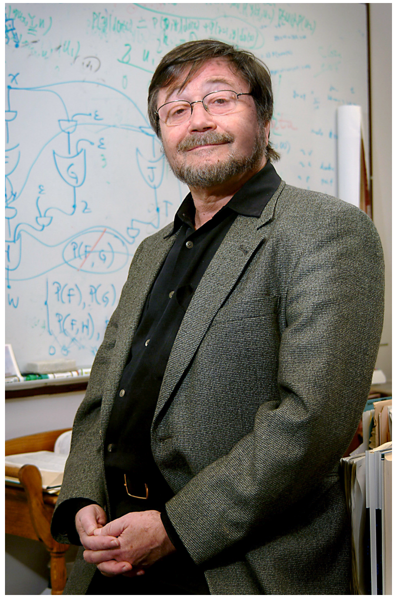


Artificial Intelligence

Local Search & Optimization

CS 444 – Spring 2021

Dr. Kevin Molloy

Department of Computer Science

James Madison University

JAMES MADISON UNIVERSITY.

Much of this lecture is taken from
Dan Klein and Pieter Abbeel AI class at UC Berkeley

Announcements

- HW 2 is due tonight
- PA 1 is due this Monday, Feb 22
- First quiz is released today after class. It is due tomorrow by 5:00 pm tomorrow Friday).

Learning Objectives for Today

- Local Search
  - Hill Climbers
  - Evolutionary Algorithms
  - Beam Search

JAMES MADISON UNIVERSITY

Review of Search Problems/Methods so Far

- Uninformed and Informed Search
  - Systematic search
  - Assume finite state space (not always the case)
  - Environment may not be fully observable

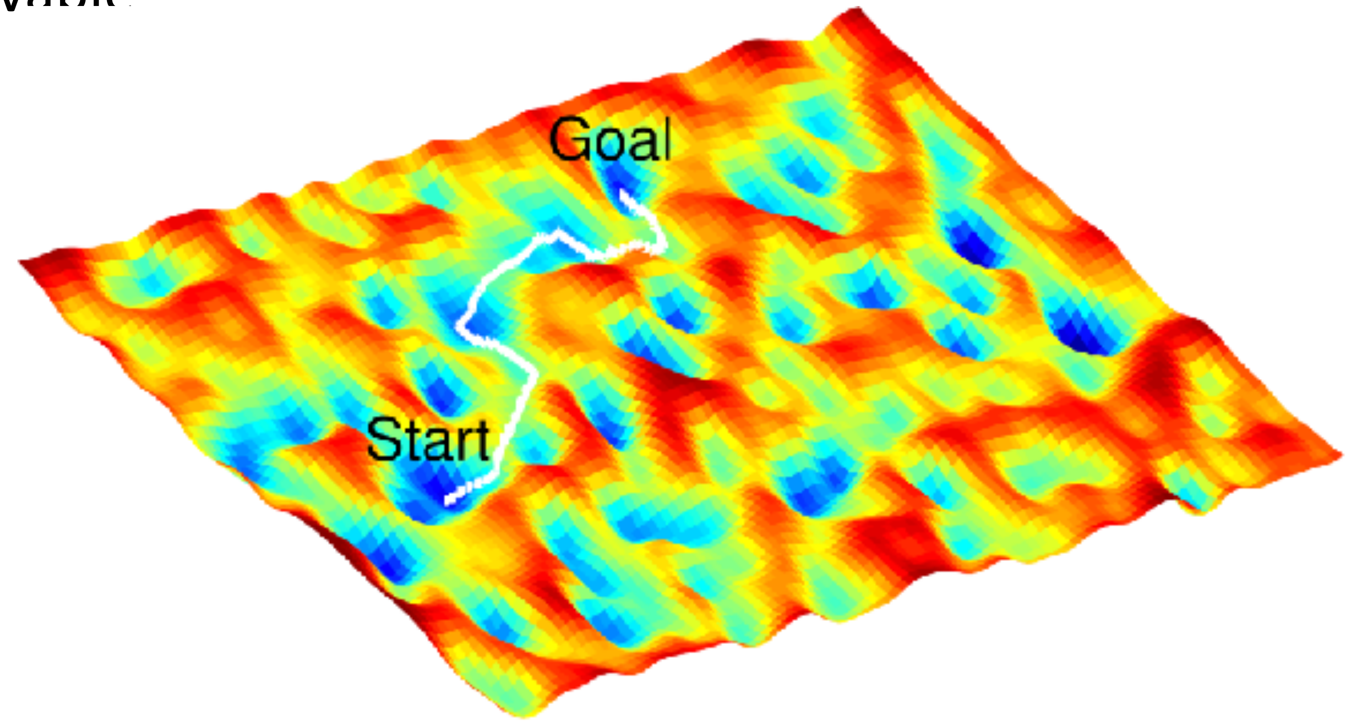


Figure from Berkley AI

Local Search

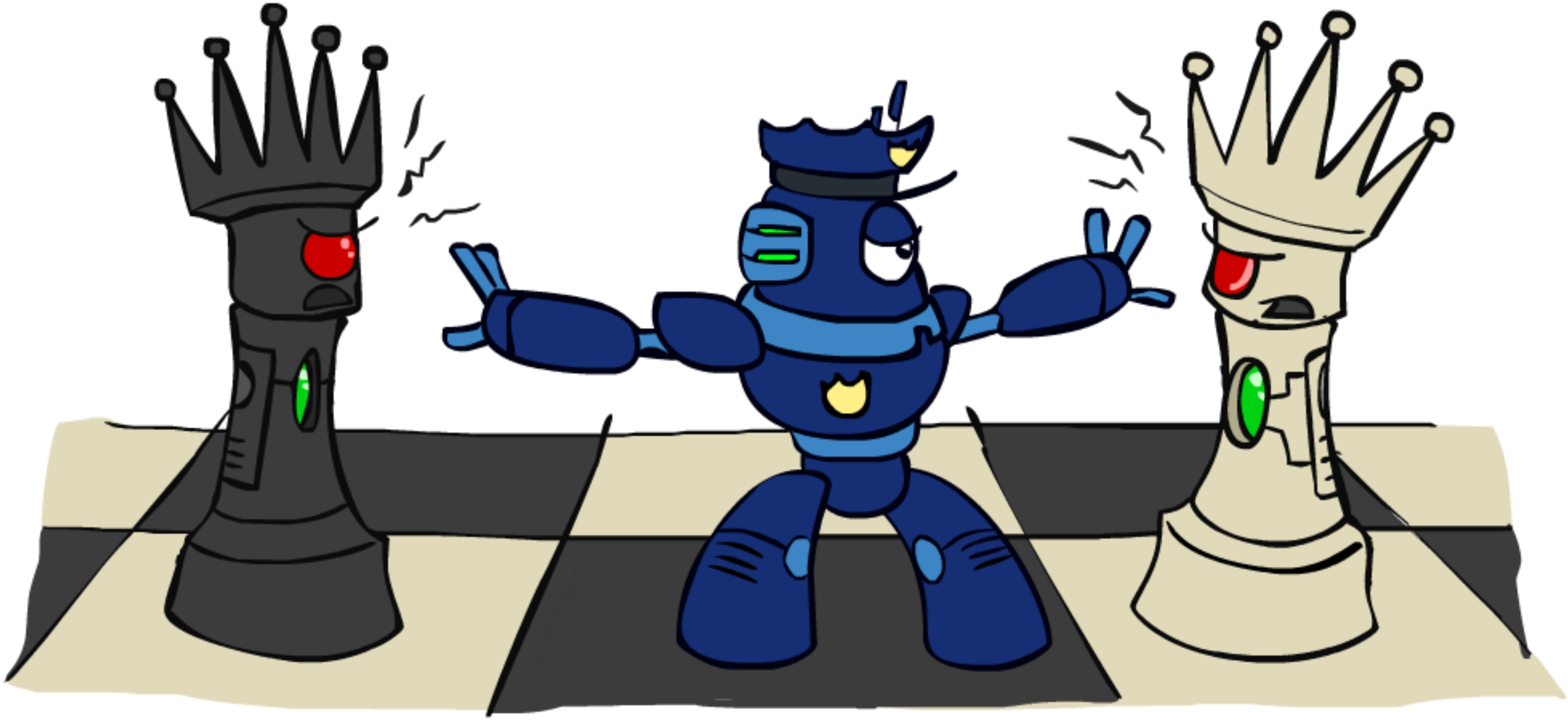
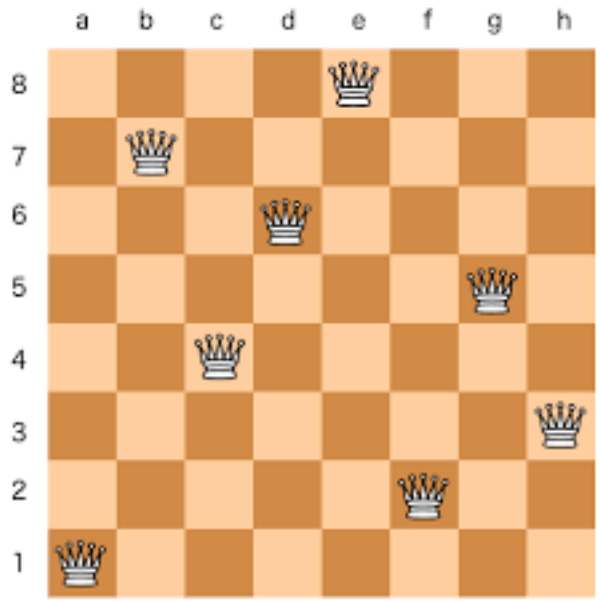
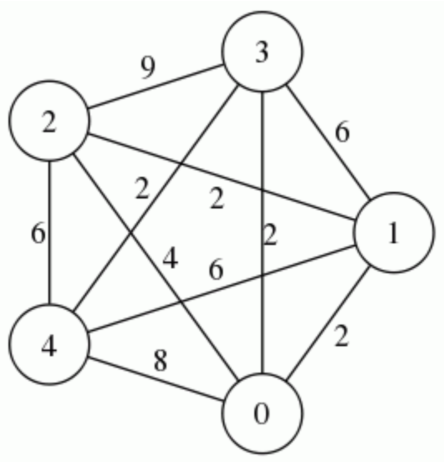


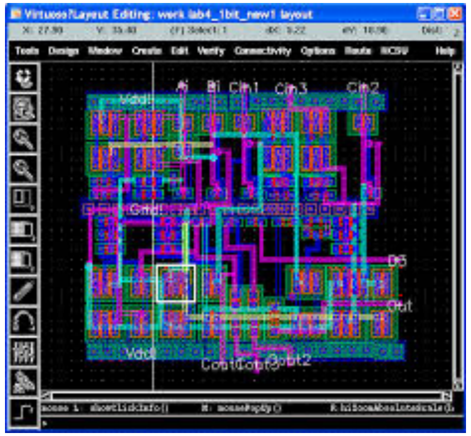
Figure from Berkley AI

Other Problems to Solve?

- Traveling Salemans Problem
- Placing N-queens on a chessboard so no queens can "attack" each other
- Design the layout of a circuit board
- Protein structure prediction







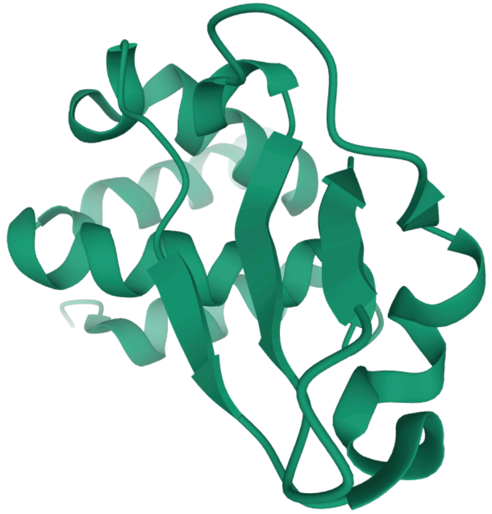


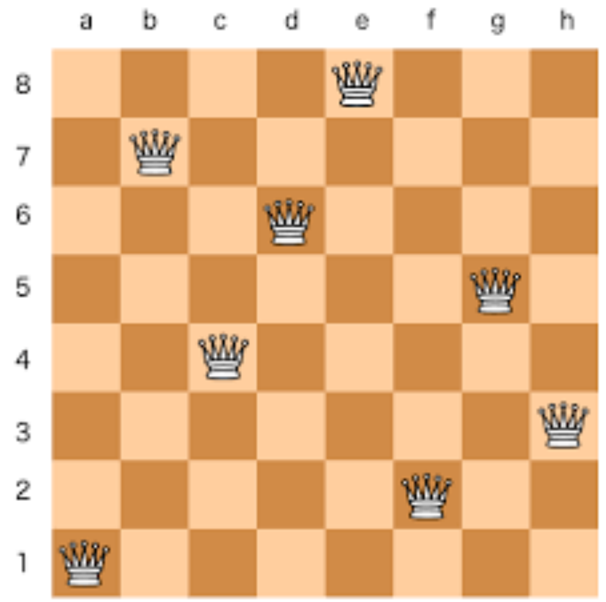
Figure from Berkley AI

Optimization Problems

Key Difference from problems so far?

- The goal itself is the solution. **No path required**
- The state space is a set of **complete** solutions.

Find the optimal configuration.



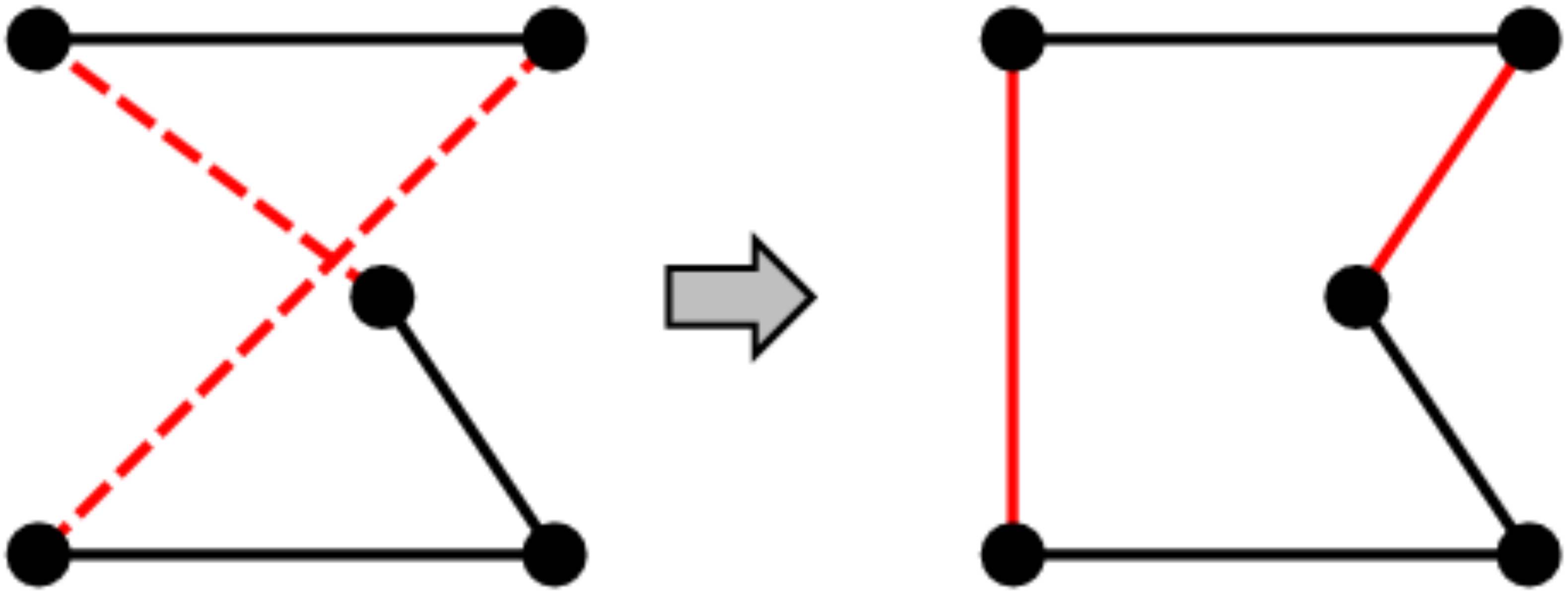
Key idea: **Iterative improvement**

- Keep a single "current" state, try to improve it. That is, no memory of what has been found so far, hence, sometimes called (memory-less) local search
- **Iterative** refers to iterating between states
- **Improvement** refers to later states improving some objective/goal function or satisfying more

Figure from Berkley AI

Example: Traveling Salesman Problem (TSP)

Start with any complete tour, perform pairwise exchanges

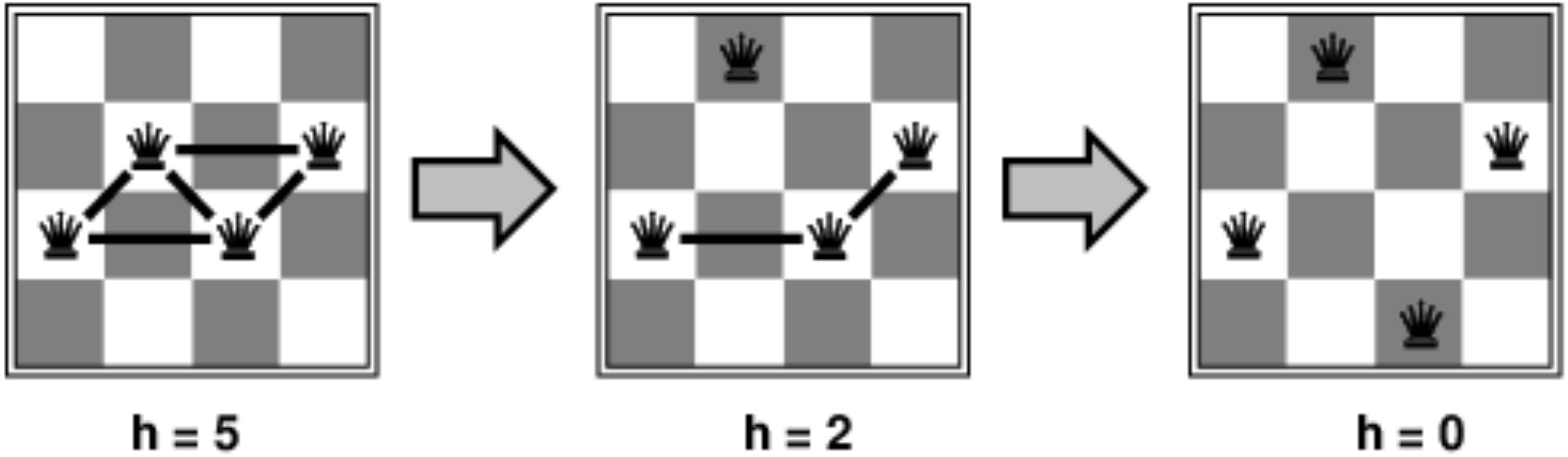


Variants of this approach get within **1%** of the optimal solution very quickly (even with thousands of cities)

Example: n-queens

Put *n* queens on an n x n board with no two queens on the same row, column, or diagonal.

Move a queen to reduce number of conflicts.



Local search techniques can solve this problem almost instantaneously for very large n (n = 1 million) (recall an 8x8 board has 8^8 states (≈ 17 million states).

Hill Climbing Algorithm

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

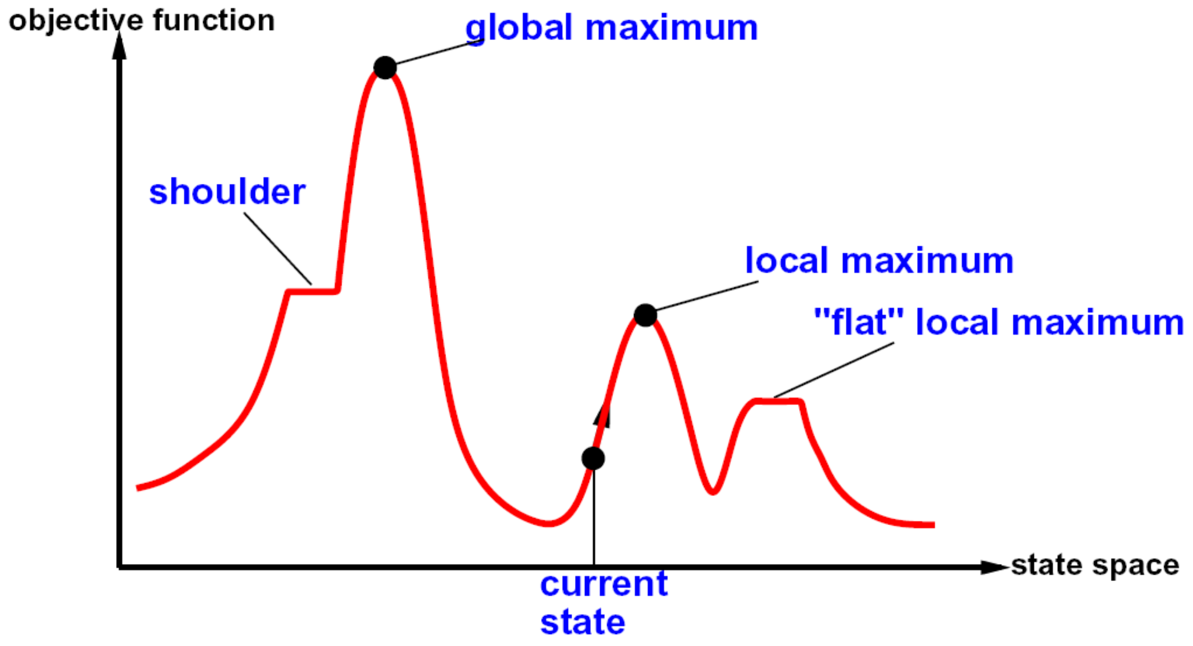
- What's bad about this approach?
  - Complete?
  - Optimal?

- What's good about it?

Figure from Berkley AI

Hill Climbing

function Hill-Climbing(problem) **returns** a state (local optimum)
    inputs: problem, a problem
    local variables: *current (a node)*
        neighbor (a node)
    current ← MAKE-NODE(INITIAL-STATE [problem])
    loop do
        neighbor ← a successor of current
        If Value[neighbor] **is not better than** Value[current]
            then return State ← [current]
        current ← neighbor
    end



Hill Climbing Generating Neighboring States

How is the neighbor of a current state generated? **Varies with approach…**

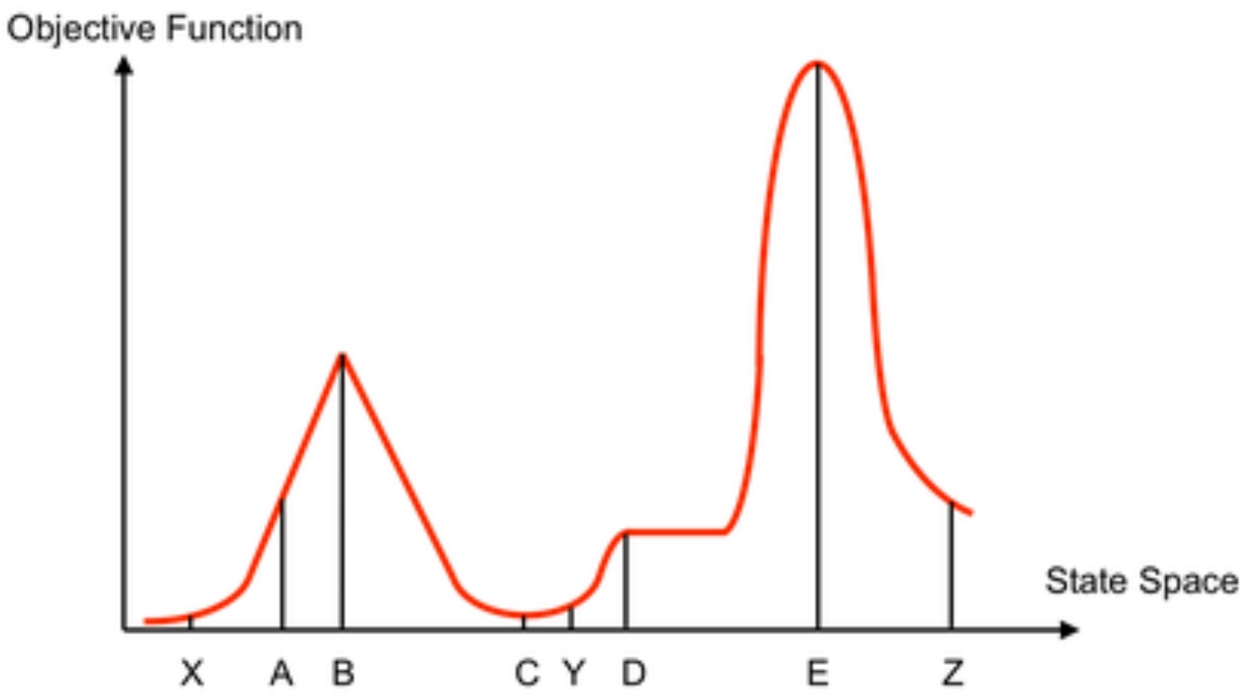
If state space is discrete and neighbor list is finite, all neighbors of a current state can be considered:

- **Steepest hill climbing**: compare best neighbor to current
- **First-choice hill climbers** use the first choice that is improves on current

What if neighbors cannot be enumerated? What if state space is continuous?

- **Stochastic hill climbing**: generate neighbor at random (continuous spaces, perform a small perturbation to generate neighbor)
- **Gradient-based variants**: for continuous state spaces
  - (Conjugate) Gradient Descent/Ascent
  - Other numerical optimization algorithms (beyond scope of CS 444)

Hill Climbing Quiz



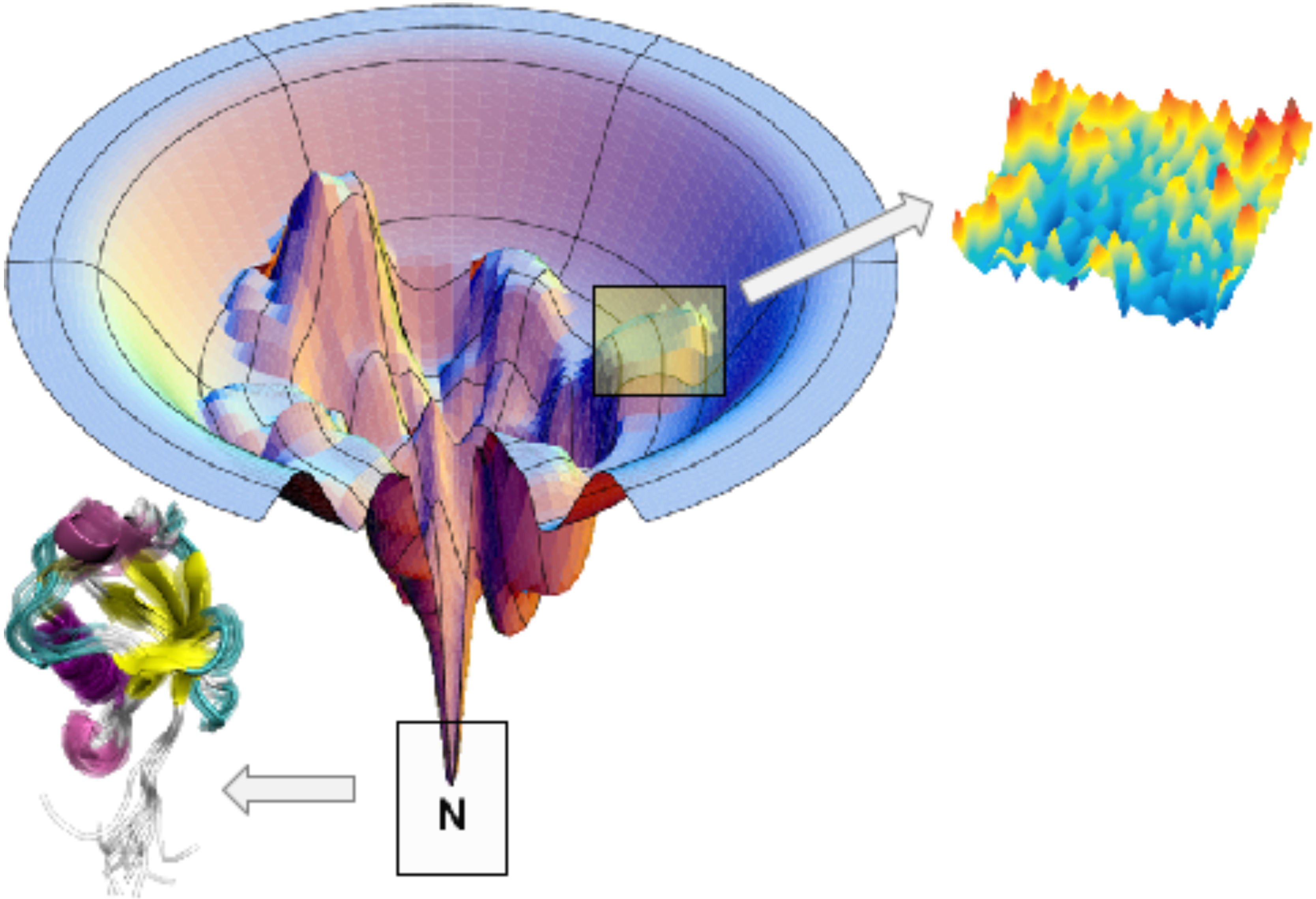
Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Figure from Berkley AI

Challenging Hill Climbing Landscape



Funnell landscape from Ken Dill

Dealing with Local Optima

Randomization:

- Random/multi restart allows embarrassing parallelization
- Iterated Local Search (ILS)

Memory-less randomized/stochastic search optimization:

Monte Carlo search

Simulated Annealing Monte Carlo

Memory-based randomized search:

- Memory via search structure
  - List: tabu search
  - Tree-/graph based search
- Memory via population
  - Evolutionary search strategies
  - Evolutionary Algorithms (Eas)
  - Genetic Algorithms (GA)

Figure from Berkley AI

Random-Restart Hill Climbers

Idea: Launch multiple hill climbers from different initial states/configurations.

Bonus: Amenable to embarrassing parallelization.

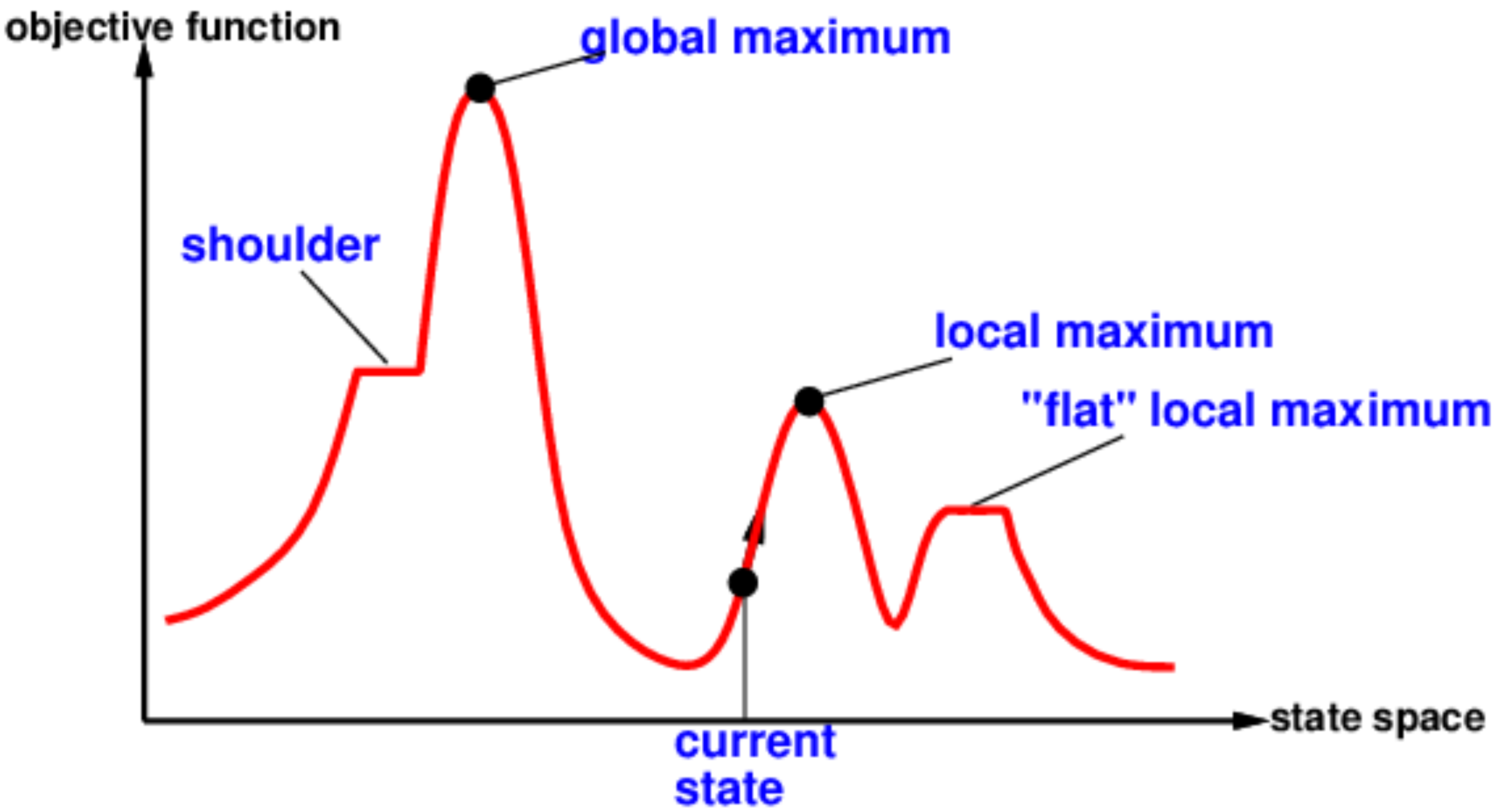
Take-away: It is often better to spend CPU time exploring the space, then carefully optimizing from an initial condition.

Why?

Repeated restarts give a global view of the state space (instead of just the local one provided by each climber).

Drawback? The hill climbers do not talk to one another.

Escaping a local maximum/minimum



How to escape from a local minimum?

Make a random move – this is what we call **Iterated Local Search (ILS)**

Figure from Berkley AI

Local Beam Search

Idea: Don't keep just a single state, keep *k* states.

Not the same a *k* searches in parallel!

Search that finds good states recruits other searches to join them

Generate *k* starting states at random.

REPEAT

For each state *k* generate a successor state

Pick the best *k* states from the set of 2*k states (the originals and the "offspring"*

Issues/Problems?

Quite often, all k states end up on some local "hill"

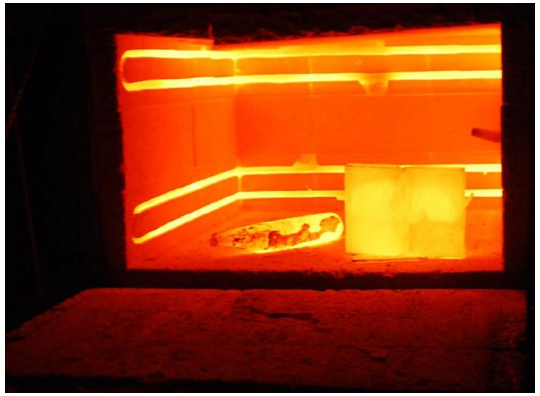
Solution: choose k successors randomly (biased towards "good" states". This is call Monte Carlo sampling (robotics/computer vision use this in particle filters).

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 ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```



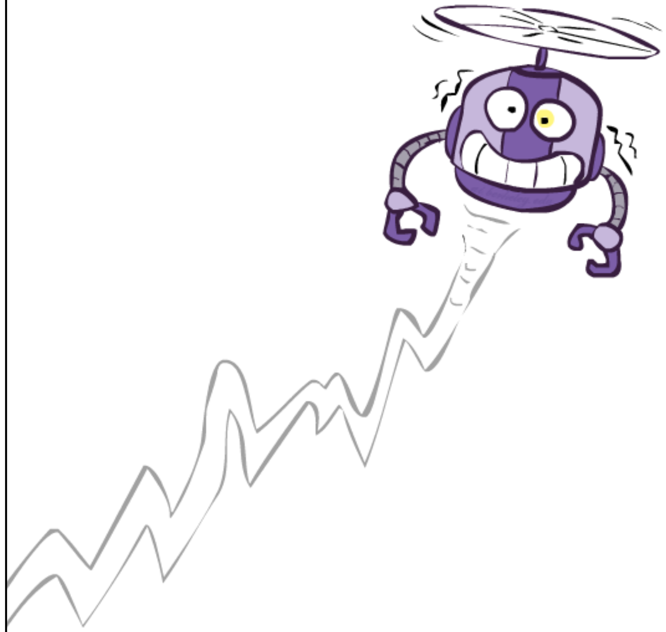


Figure from Berkley AI

JAMES MADISON UNIVERSITY

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 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

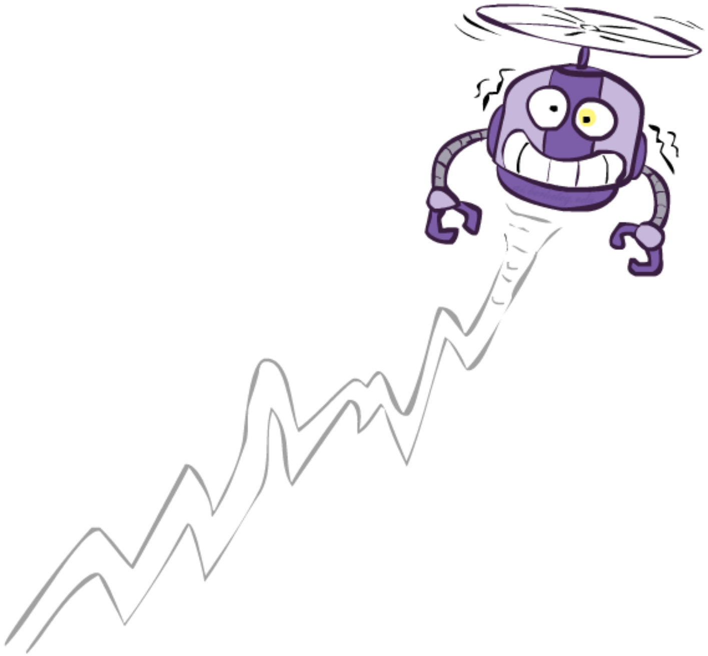
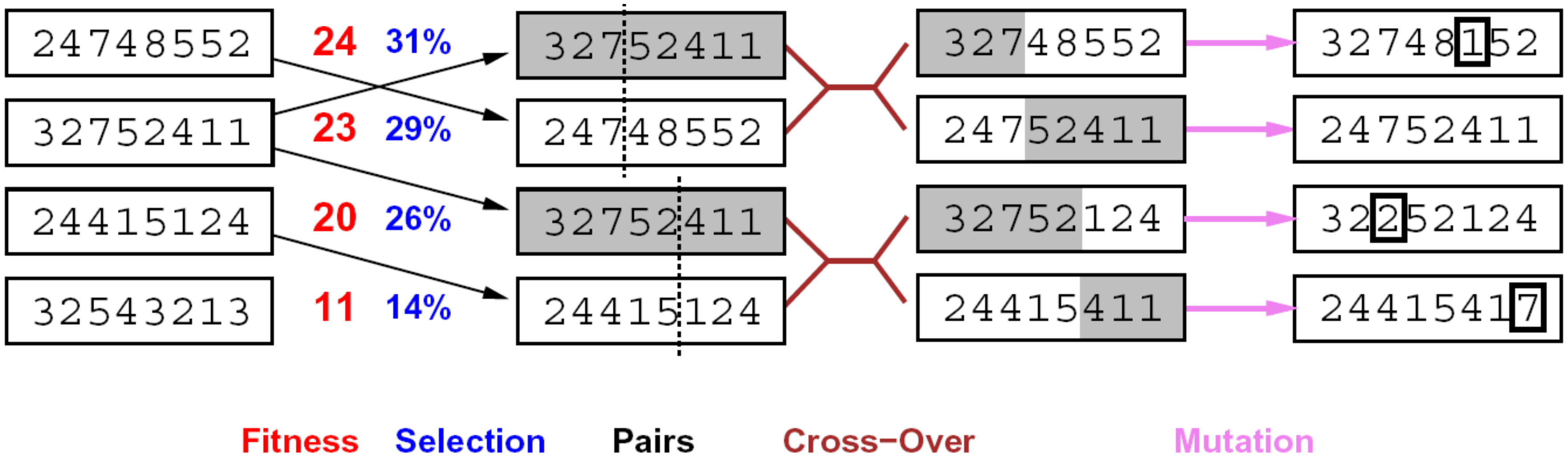
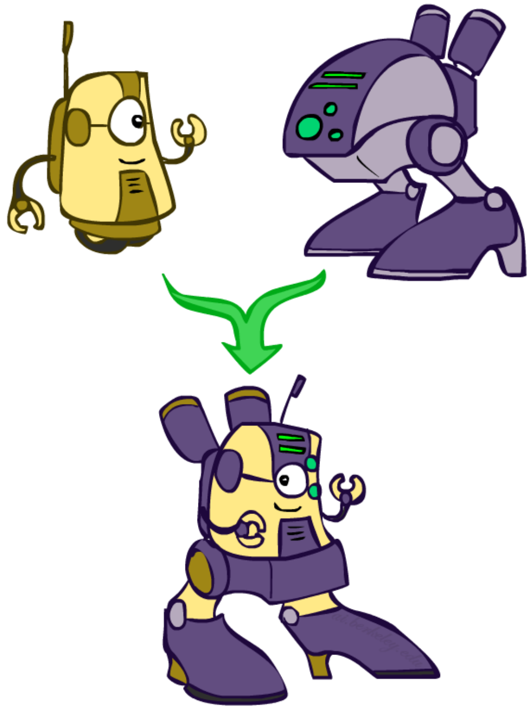


Figure from Berkley AI

Genetic Algorithms





- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around