Introduction to Artificial Intelligence

Local search

Instructor: Tatjana Petrov

University of Trieste, Italy

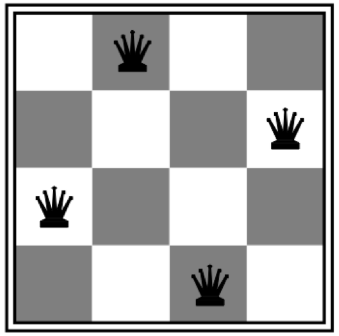
[slides adapted from Dan Klein, Pieter Abbeel, Stuart Russell, et al for CS188 Intro to AI at UC Berkeley. All materials available at http://ai.berkeley.edu.]

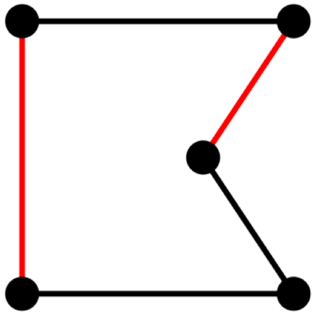
Today

- Local Search
  - In a discrete space
  - In a continuous space

Local search algorithms

- In many optimization problems, ***path*** is irrelevant; the goal state ***is*** the solution
- Then state space = set of "complete" configurations;
  find ***configuration satisfying constraints***, e.g., n-queens problem; or, find ***optimal configuration***, e.g., travelling salesperson problem





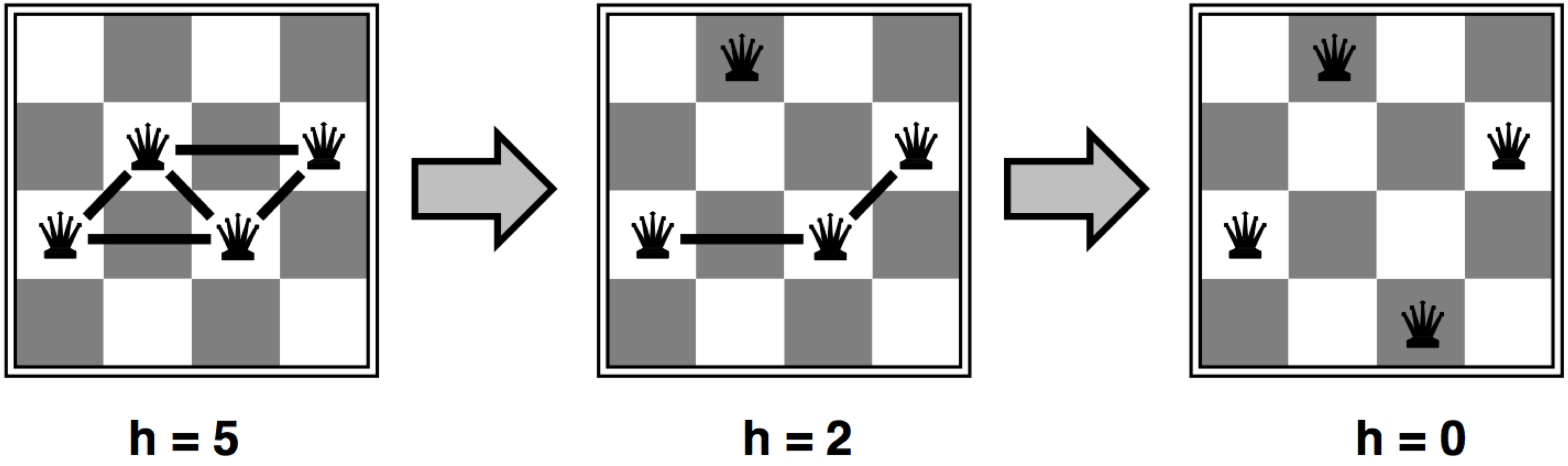
- In such cases, can use ***iterative improvement*** algorithms: keep a single "current" state, try to improve it
- Constant space, suitable for online as well as offline search
- More or less unavoidable if the "state" is yourself (i.e., learning)

Hill Climbing

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

Heuristic for *n*-queens problem

- Goal: n queens on board with no ***conflicts***, i.e., no queen attacking another
- States: n queens on board, one per column
- Actions: move a queen in its column
- Heuristic value function: number of conflicts

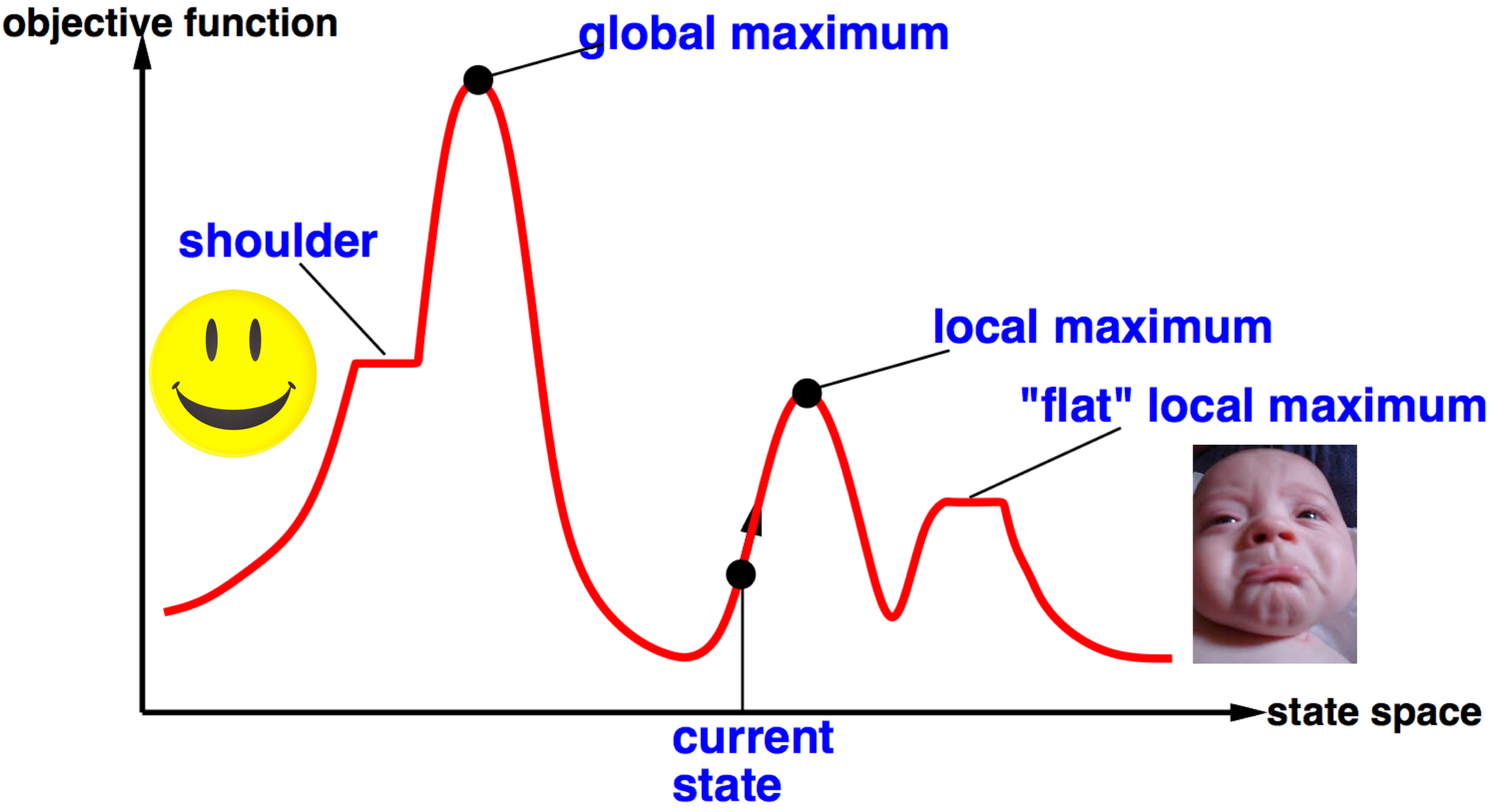


Hill-climbing algorithm

```
function HILL-CLIMBING(problem) returns a state
    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
```

"Like climbing Everest in thick fog with amnesia"

Global and local maxima



Random restarts

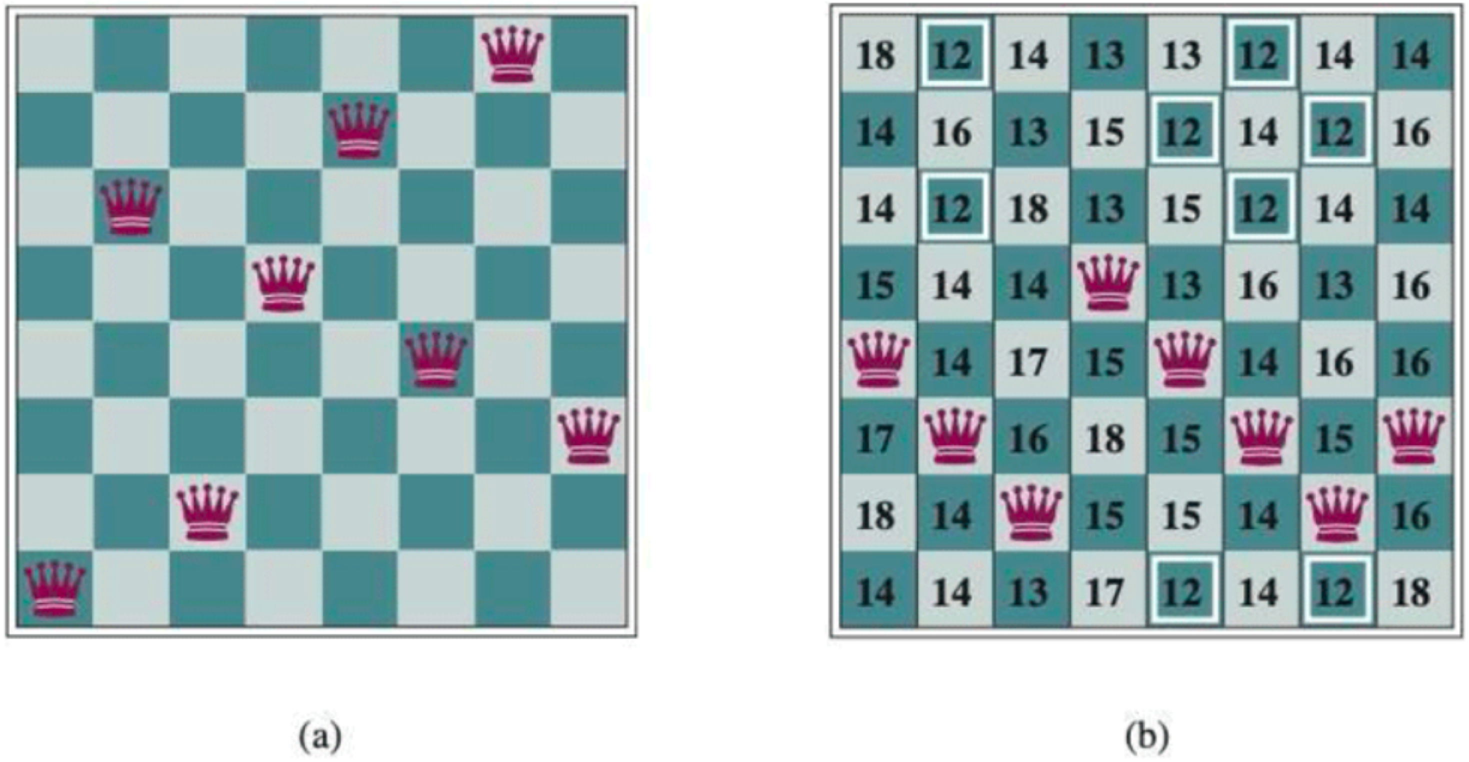
- find global optimum
- duh

Random sideways moves

- Escape from shoulders
- Loop forever on flat local maxima

Heuristic for *n*-queens problem

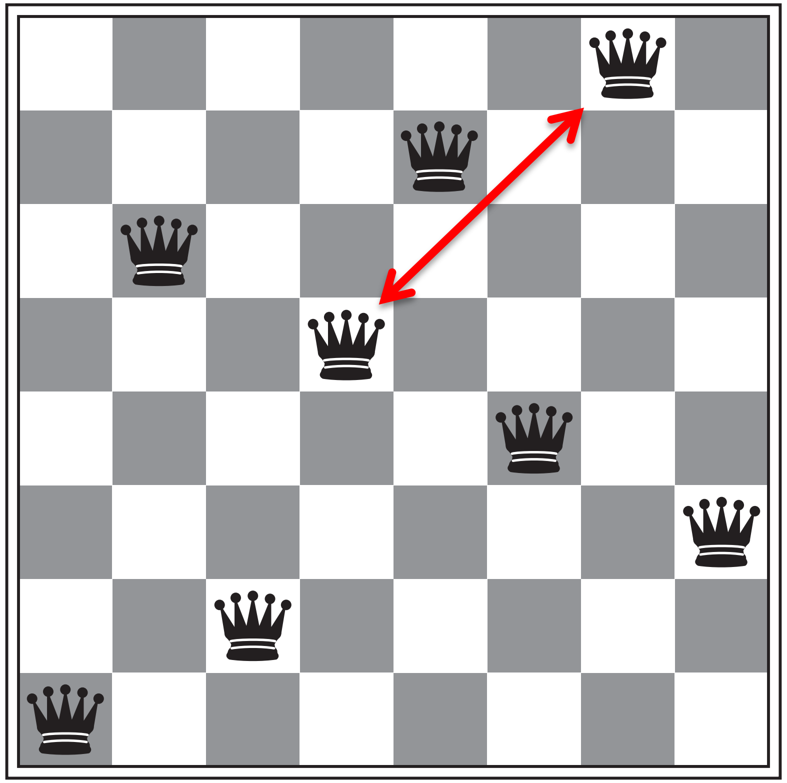
Figure 4.3



(a) The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) This position is almost a solution, except for the two queens in the fourth and seventh columns that attack each other along the diagonal. (b) An 8-queens state with heuristic cost estimate $h = 17$. The board shows the value of h for each possible successor obtained by moving a queen within its column. There are 8 moves that are tied for best, with $h = 12$. The hill-climbing algorithm will pick one of these.

Hill-climbing on the 8-queens problem

- No sideways moves:
  - Succeeds w/ prob. 0.14
  - Average number of moves per trial:
    - 4 when succeeding, 3 when getting stuck
  - Expected total number of moves needed:
    - 4 + 3(1-p)/p =~ 22 moves
- Allowing 100 sideways moves:
  - Succeeds w/ prob. 0.94
  - Average number of moves per trial:
    - 21 when succeeding, 65 when getting stuck
  - Expected total number of moves needed:
    - 21 + 65(1-p)/p =~ 25 moves



Moral: algorithms with knobs to twiddle are irritating

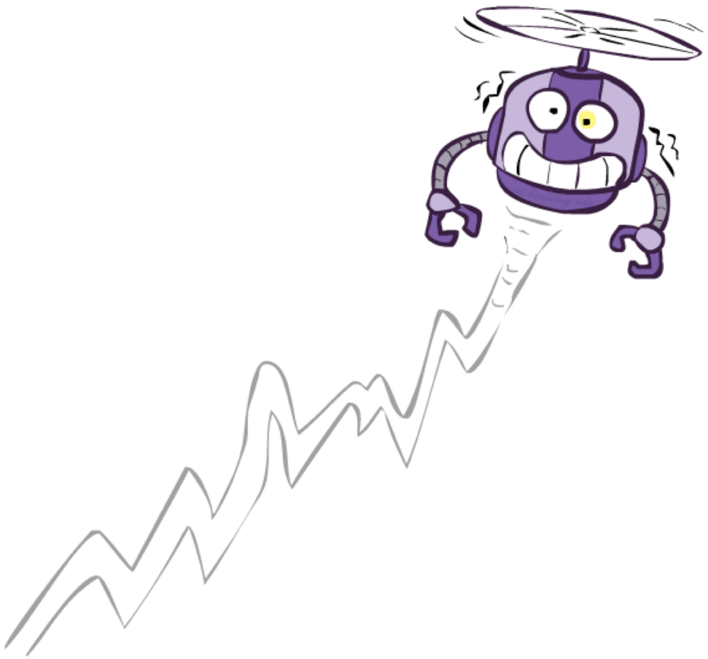
Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state
- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty instable, doesn’t it?

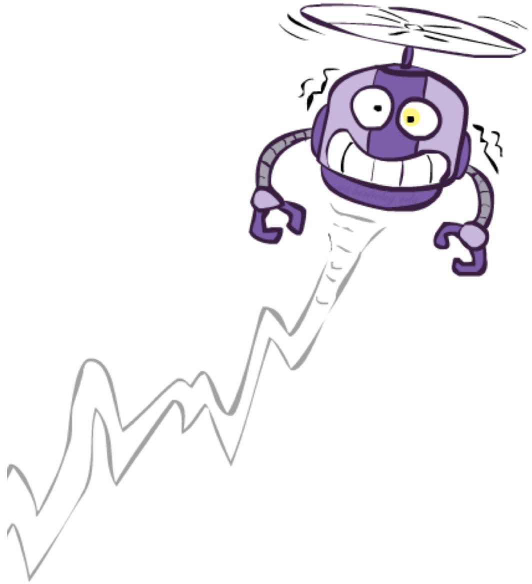
Simulated annealing algorithm

```
function SIMULATED-ANNEALING(problem,schedule) returns a  state
current ← problem.initial-state
for t = 1 to ∞ do
        T ←schedule(t)
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← next.value – current.value
        if ΔE > 0 then current ← next
                  else current ← next only with probability e^(ΔE/T)
```

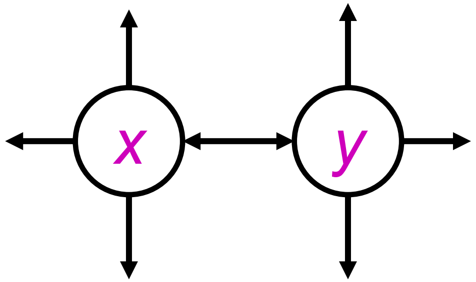
$T \to \infty \Rightarrow e^0 = 1$



Simulated Annealing



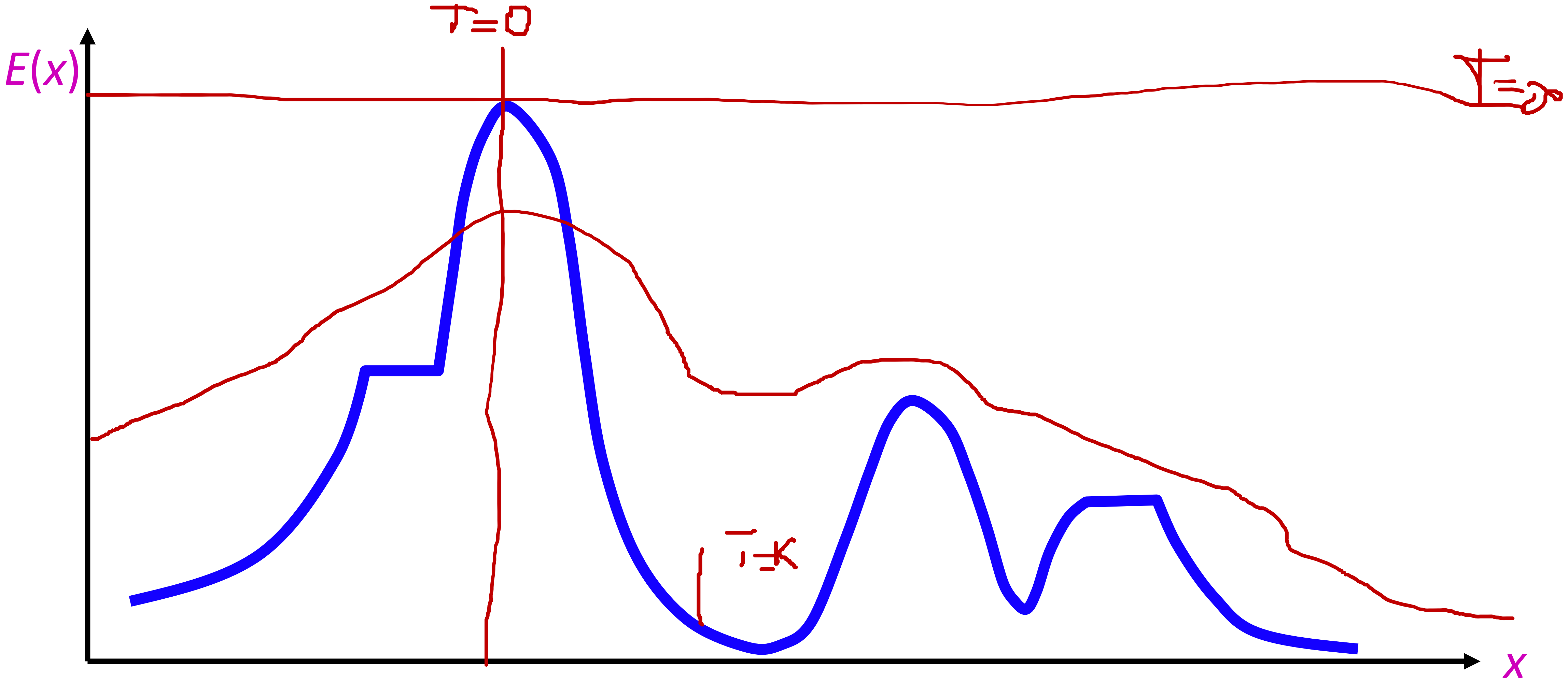
- Theoretical guarantee:
  - Stationary distribution (Boltzmann): $P(x) \alpha\, e^{E(x)/T}$
  - If T decreased slowly enough, will converge to optimal state!
- Proof sketch
  - Consider two adjacent states x, y with $E(y) > E(x)$ [high is good]
  - Assume $x \rightarrow y$ and $y \rightarrow x$ and outdegrees $D(x) = D(y) = D$
  - Let $P(x)$, $P(y)$ be the equilibrium occupancy probabilities at T
  - Let $P(x \rightarrow y)$ be the probability that state x transitions to state y



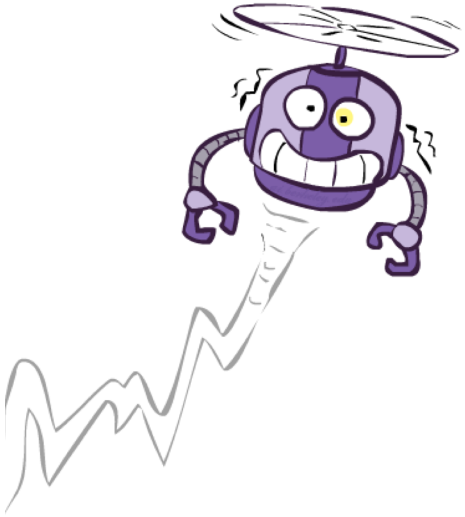
$$P(x)P(x \rightarrow y) = P(y)P(y \rightarrow x)$$

$$P(x)\frac{1}{D} = P(y)\frac{e^{\frac{E(x)-E(y)}{T}}}{D} \Rightarrow \frac{P(x)}{P(y)} = \frac{e^{E(x)/T}}{e^{E(y)/T}}$$

Occupation probability as a function of *T*



Simulated Annealing



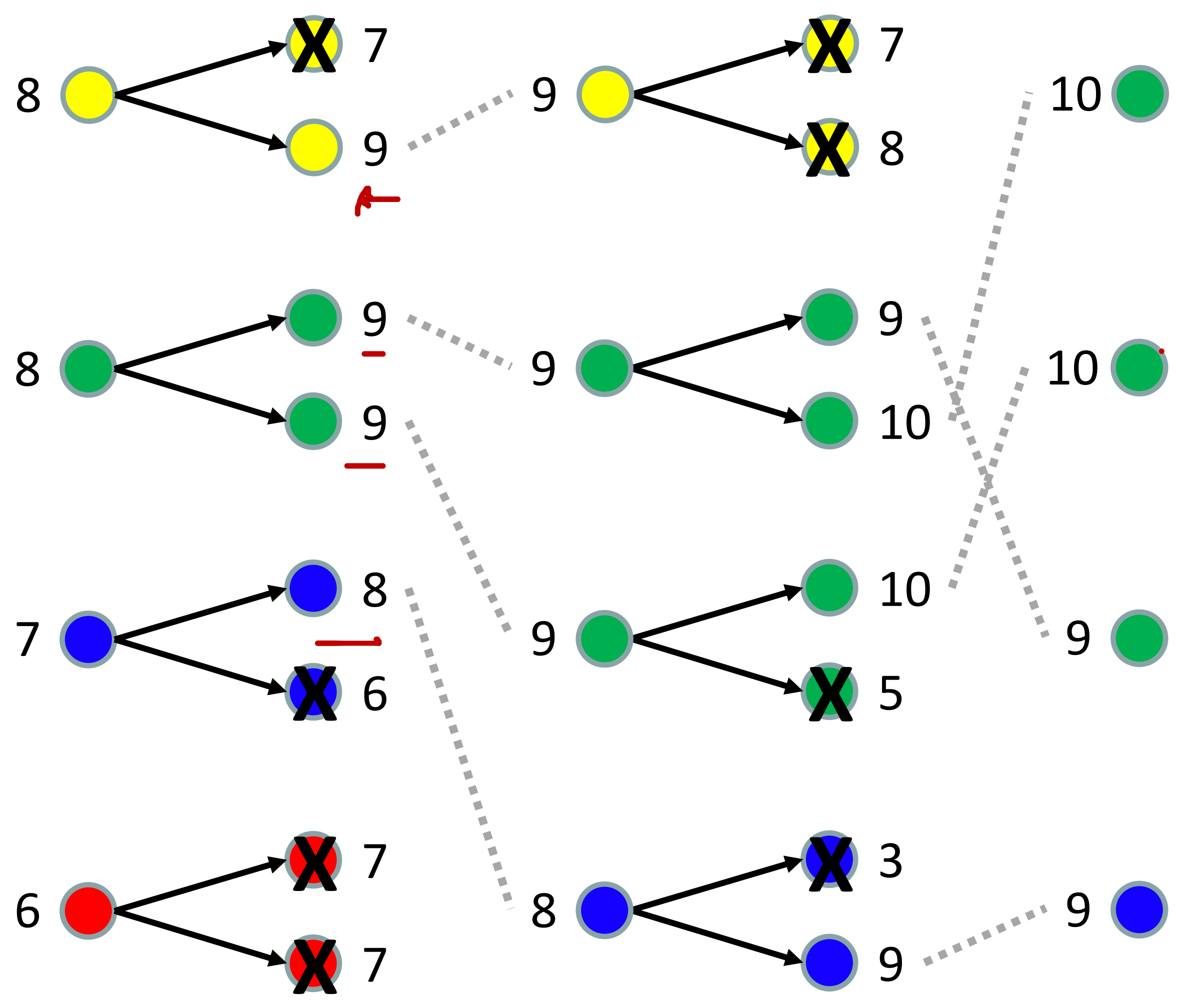
- Is this convergence an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - “Slowly enough” may mean exponentially slowly
  - Random restart hillclimbing also converges to optimal state…
- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems

Local beam search

- Basic idea:
  - *K* copies of a local search algorithm, initialized randomly
  - For each iteration
    - Generate ALL successors from *K* current states
    - Choose best *K* of these to be the new current states

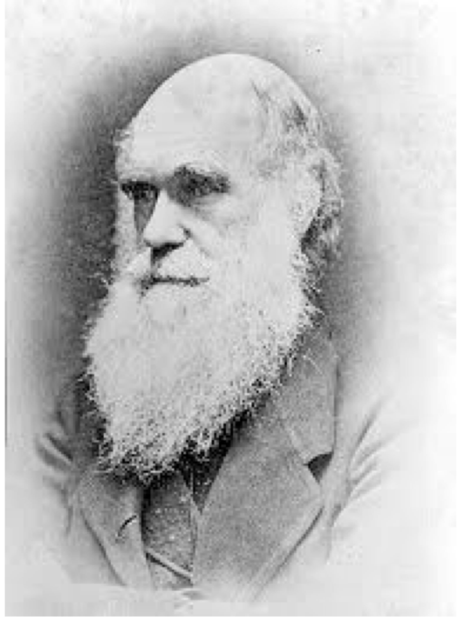
Or, K chosen randomly with a bias towards good ones

Beam search example (*K*=4)

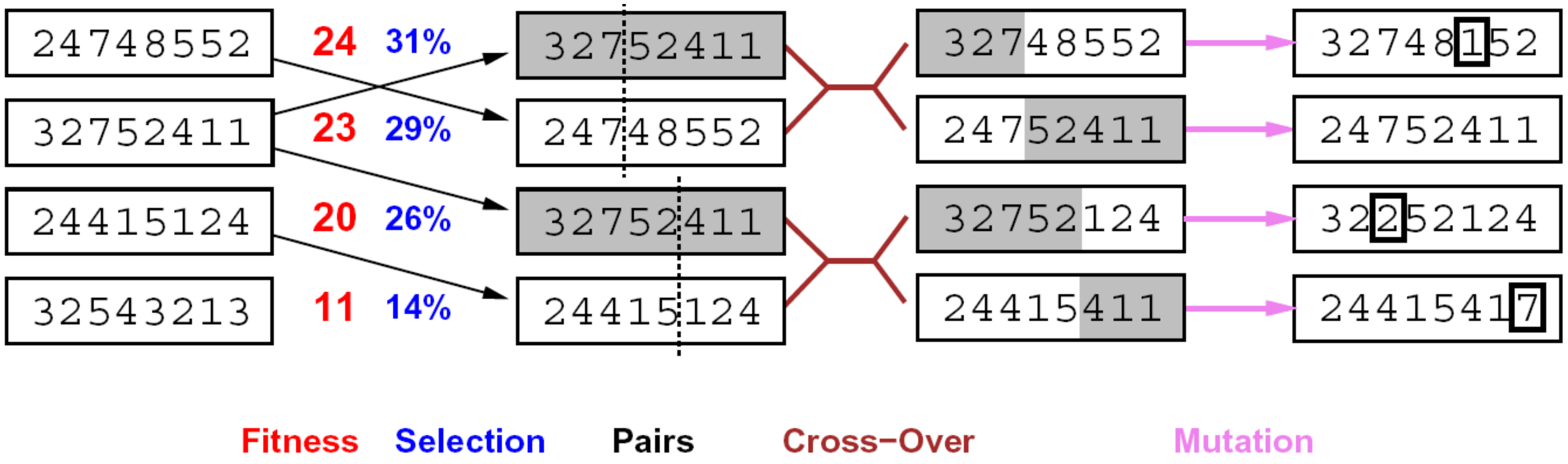


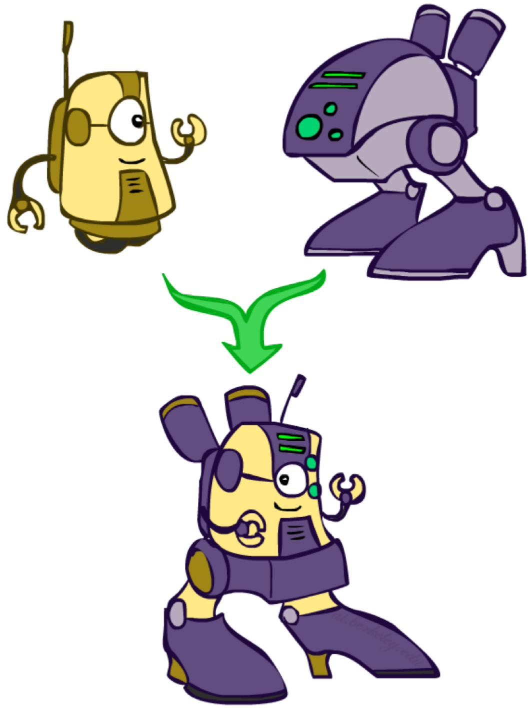
Local beam search

- Why is this different from *K* local searches in parallel?
  - The searches ***communicate***! "Come over here, the grass is greener!"
- What other well-known algorithm does this remind you of?
  - Evolution!



Genetic algorithms



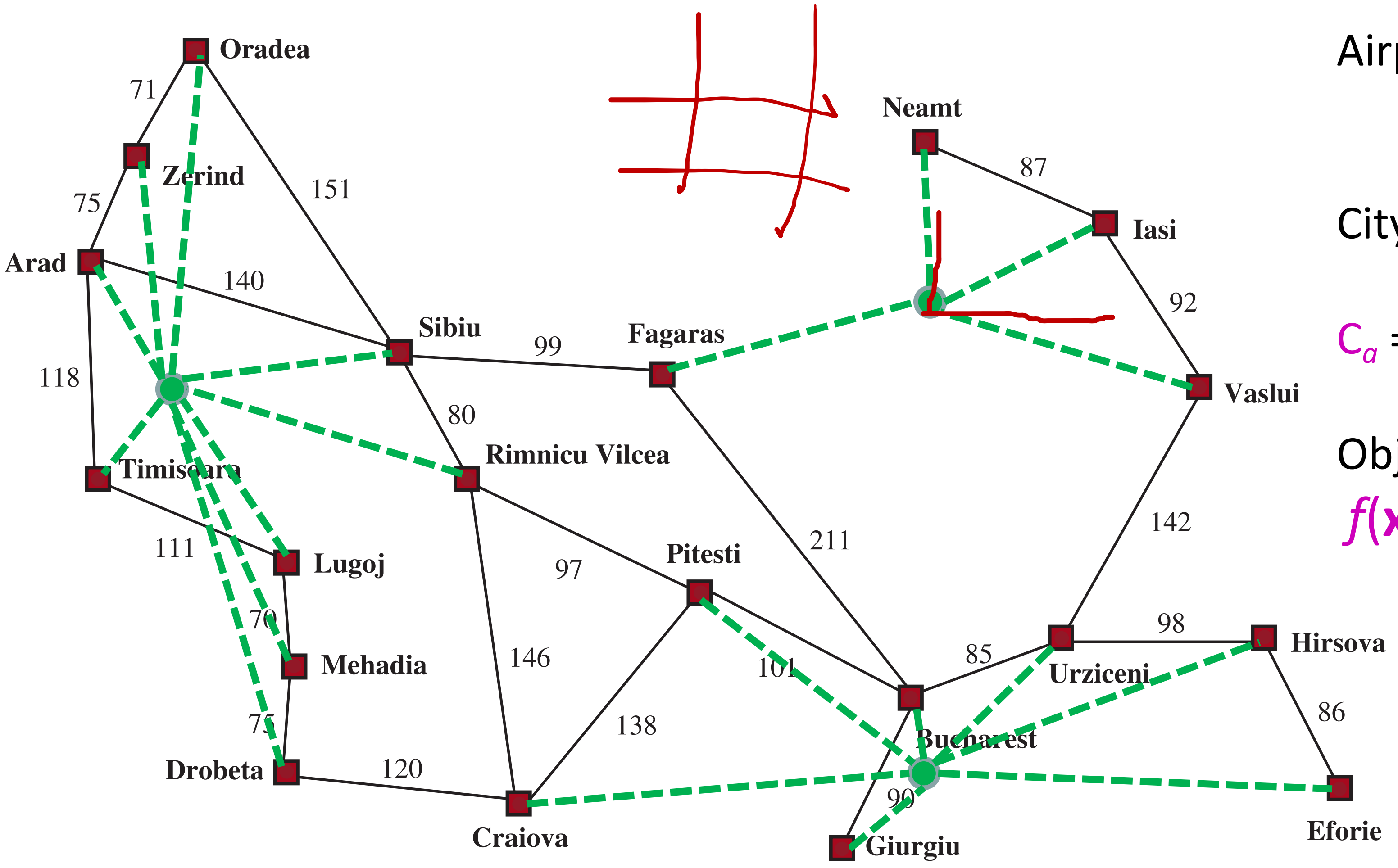


- Genetic algorithms use a natural selection metaphor
  - Resample *K* individuals at each step (selection) weighted by fitness function
  - Combine by pairwise crossover operators, plus mutation to give variety

Local search in continuous spaces

Example: Siting airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport



Airport locations

$\mathbf{x} = (x_1,y_1), (x_2,y_2), (x_3,y_3)$

City locations (x_c,y_c)

C_a = cities closest to airport a

Objective: minimize

$f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$

Handling a continuous state/action space

1. Discretize it!
   - Define a grid with increment δ, use any of the discrete algorithms

2. Choose random perturbations to the state
   a. First-choice hill-climbing: keep trying until something improves the state
   b. Simulated annealing (decreasing δ)

3. Compute gradient of $f(\mathbf{x})$ analytically

Finding extrema in continuous space

- Gradient vector $\nabla f(\mathbf{x}) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, \ldots)^T$
- For the airports, $f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\partial f/\partial x_1 = \Sigma_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(\mathbf{x}) = 0$
- Is this a local or global minimum of f?
- Gradient descent: $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$
  - Huge range of algorithms for finding extrema using gradients
- Constrained optimization
  - Most famous: **linear programming problems**

Summary

- Many configuration and optimization problems can be formulated as local search
- General families of algorithms:
  - Hill-climbing, continuous optimization
  - Simulated annealing (and other stochastic methods)
  - Local beam search: multiple interaction searches
  - Genetic algorithms: break and recombine states

Many machine learning algorithms are local searches