Informed Search

Chapter 4 (b)

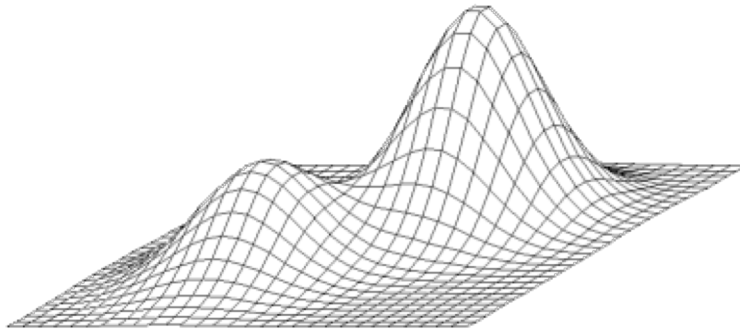
Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Today's class: local search

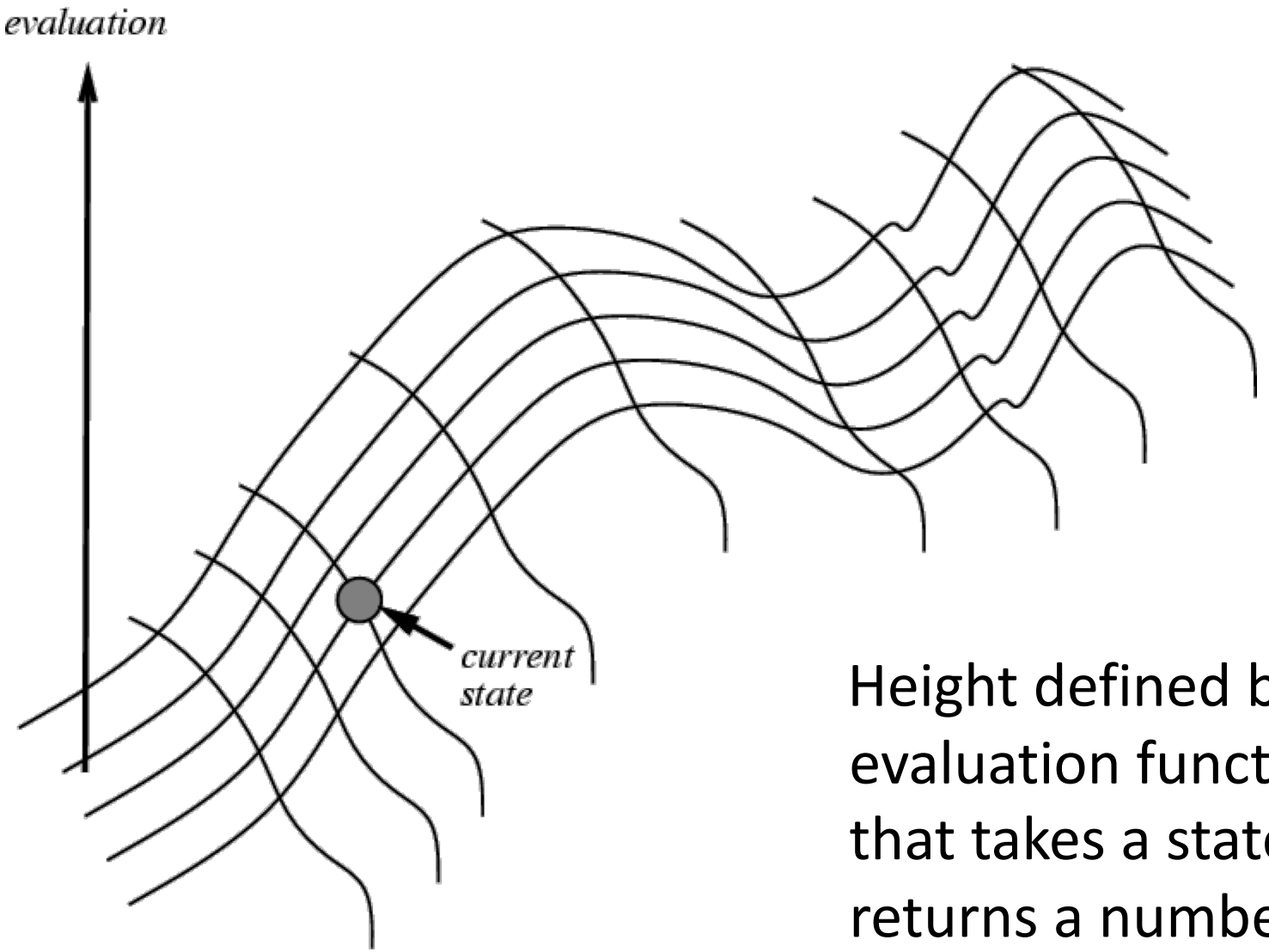
- Iterative improvement methods (aka local search) move from potential solution to potential solution until a goal is reached
- Examples
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms
- Online search

Hill Climbing

- Extended current path with successor that's closer to solution than end of current path
- If goal is to get to the top of a hill, then always take a step that leads you up
- Simple hill climbing: take any upward step
- Steepest ascent hill climbing: consider all possible steps, take one that goes up most
- No memory required



Hill climbing on a surface of states



Height defined by an evaluation function that takes a state & returns a number

Hill climbing for search

- For informed search and many other problems (e.g., neural network training) we want to find a global **minimum**
  - Search evaluation function: measure of how far the current state is from a goal
- This is an easy change to make in the algorithm, or we can just negate the evaluation function
- We still call it hill climbing though

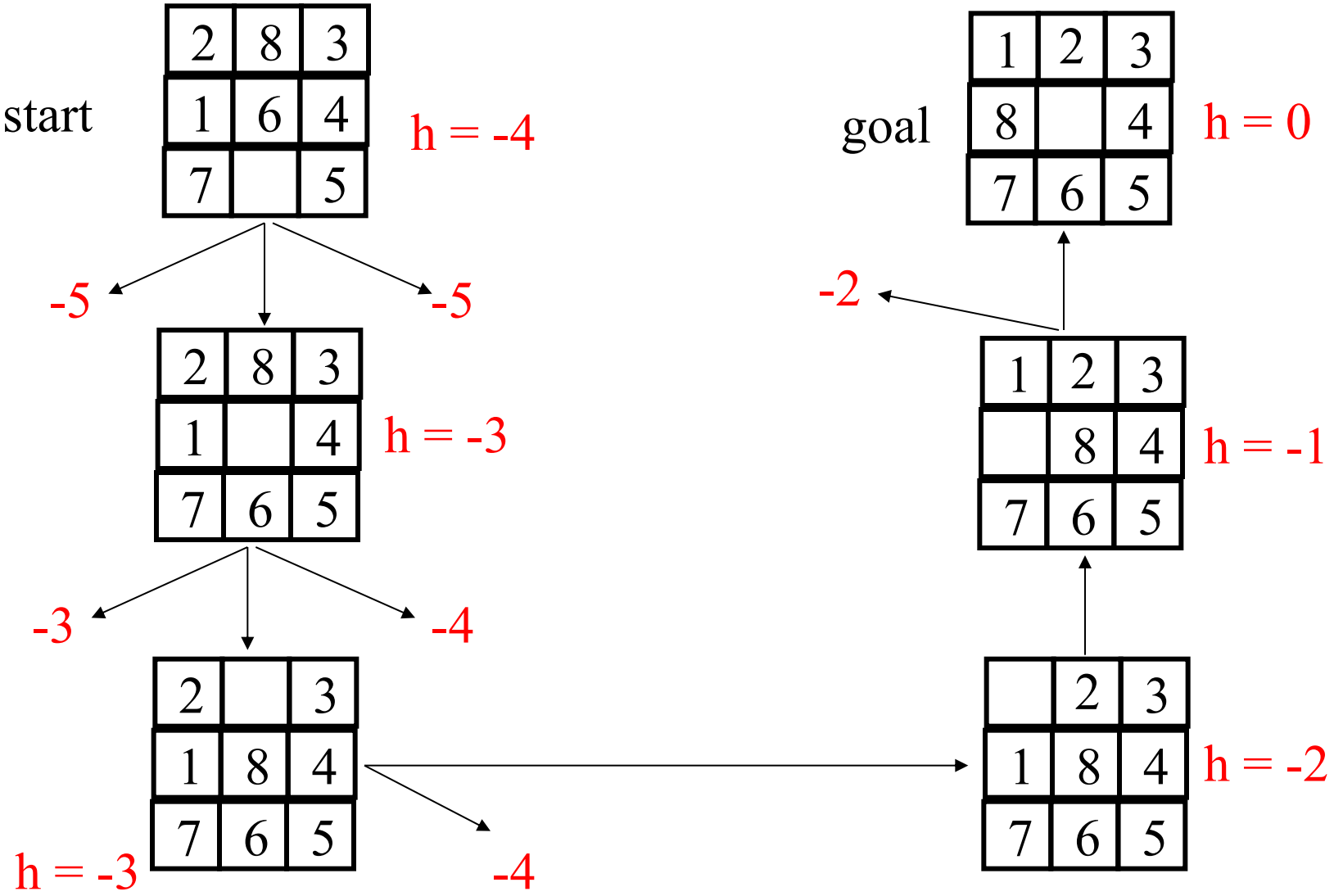
Hill-climbing search

- If there's successor **s** for current state **n** such that
  - h(s) < h(n) and h(s) <= h(t) for all successors t

  then move from **n** to **s**; otherwise, halt at **n**

  i.e.: Look one step ahead to decide if a successor is better than current state; if so, move to best successor
- Like *greedy search*, but doesn't allow backtracking or jumping to alternative path since it has no memory
- Like beam search with a beam width of 1 (i.e., maximum size of the nodes list is 1)
- Not complete since search may terminate at a local minima, plateau or ridge

Hill climbing example



f(n) = -(number of tiles out of place)

Exploring the Landscape

- **Local Maxima**: peaks not highest point in space
- **Plateaus:** broad flat region that gives search algorithm no guidance (use random walk)
- **Ridges:** flat like plateau, but with drop-offs to sides; steps to North, East, South and West may go down, but step to NW may go up

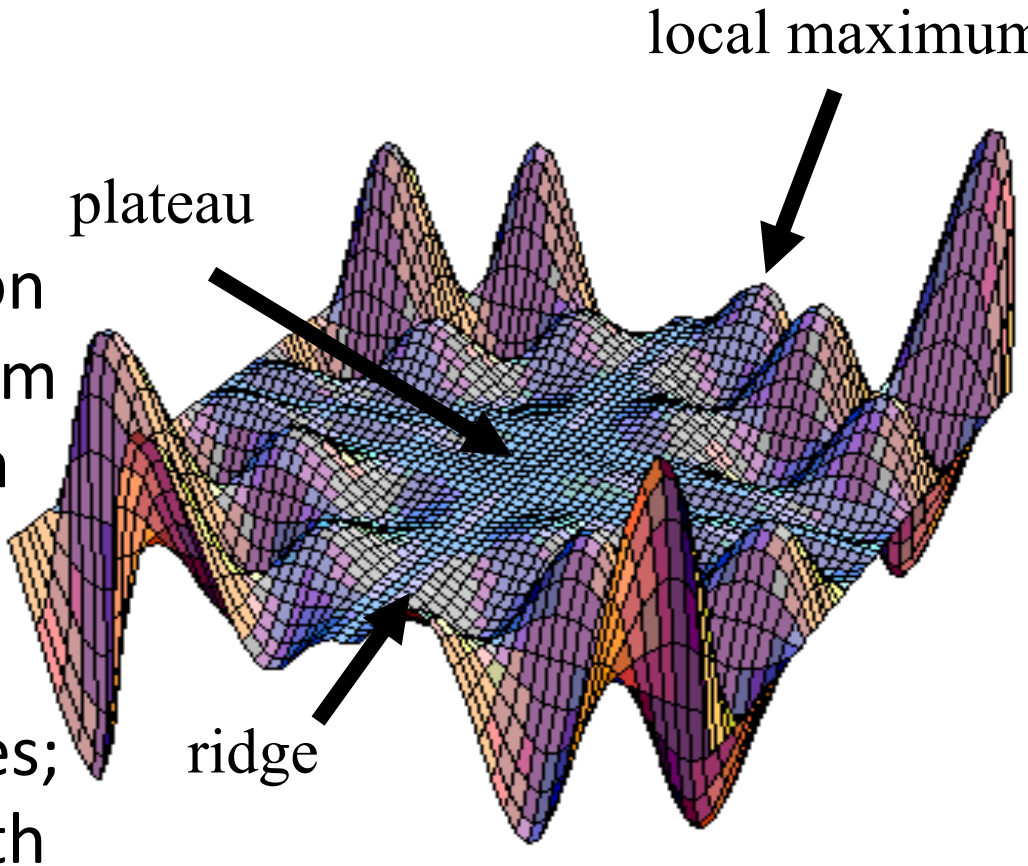


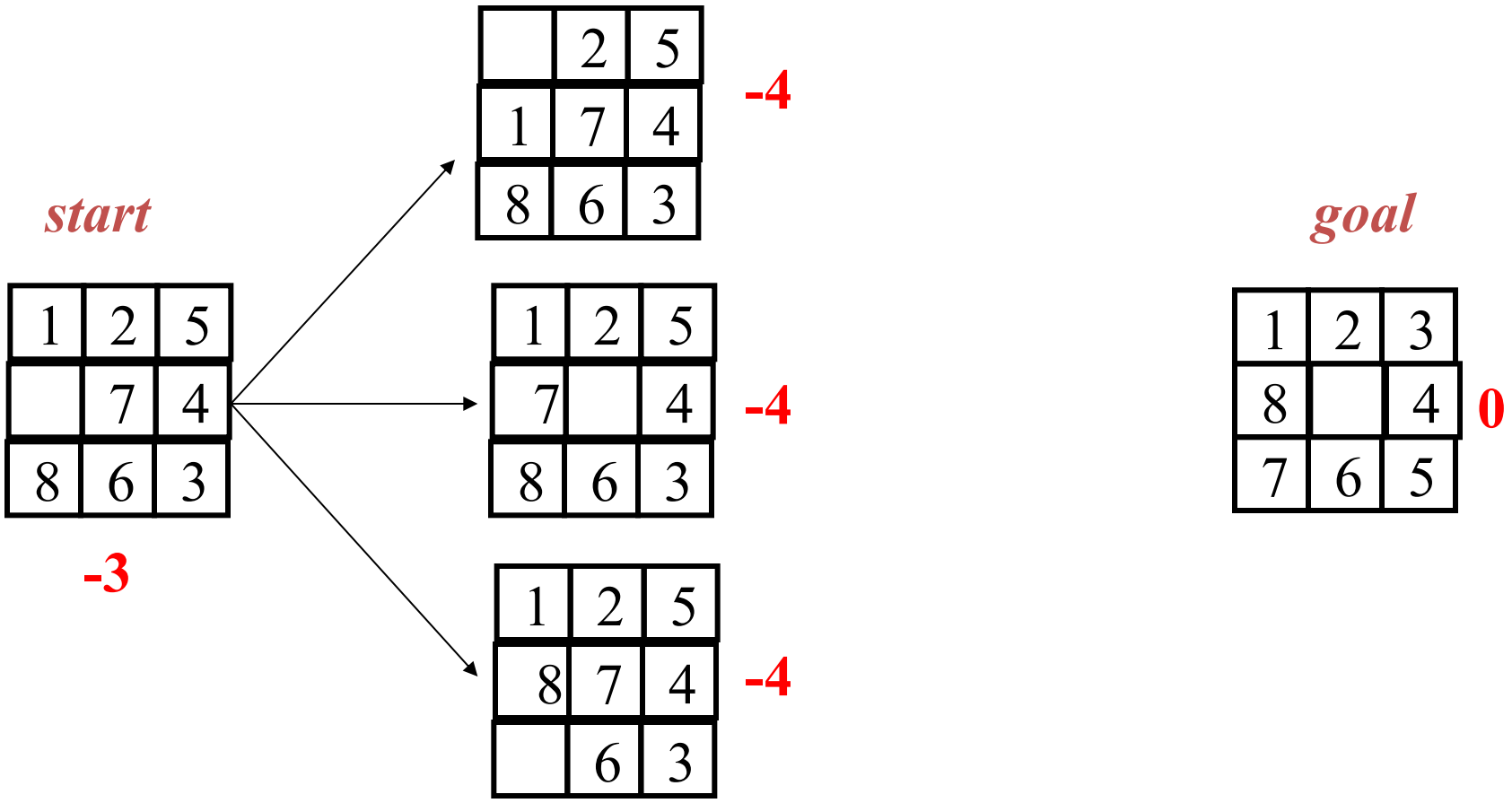
Image from: http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html

Drawbacks of hill climbing

- Problems: local maxima, plateaus, ridges
- Possible remedies:
  - **Random restart:** keep restarting search from random locations until a goal is found

    may require an estimate – *how low can we go*
  - **Problem reformulation:** reformulate search space to eliminate these problematic features
- Some problem spaces are great for hill climbing and others are terrible

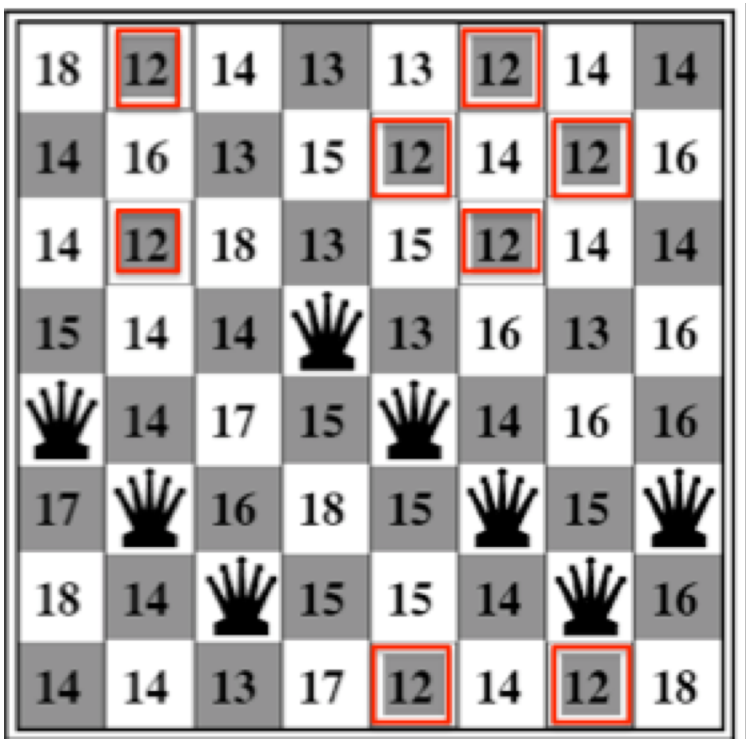
Example of a local optimum



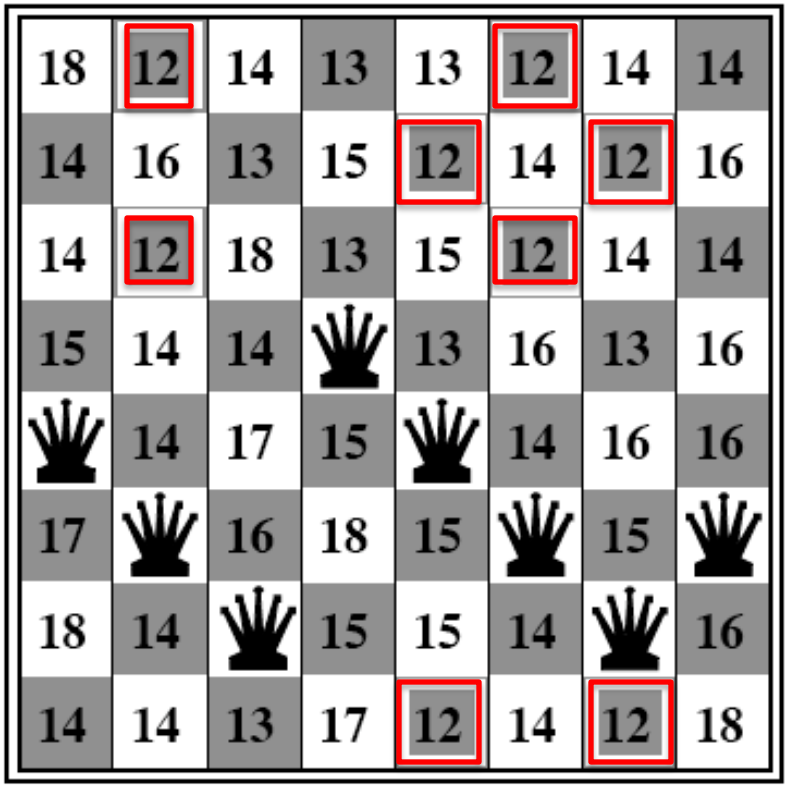
Hill Climbing and 8 Queens

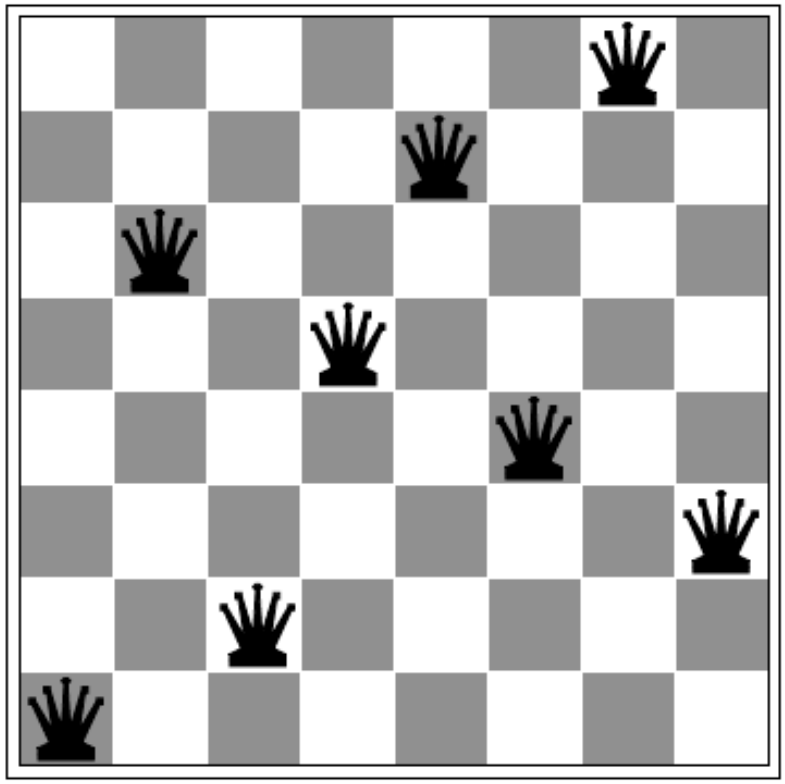
The 8 Queens problem often setup as follows:

- Randomly put one queen in each column
- Goal state: no two queens attack one another
- An action is moving any queen to a different row
- Each state thus has 65 successors
- Heuristic h: # of pairs attacking one another
- Current state: h= 17
- h=0 => solution



Hill Climbing and 8 Queens





(a) (b)

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

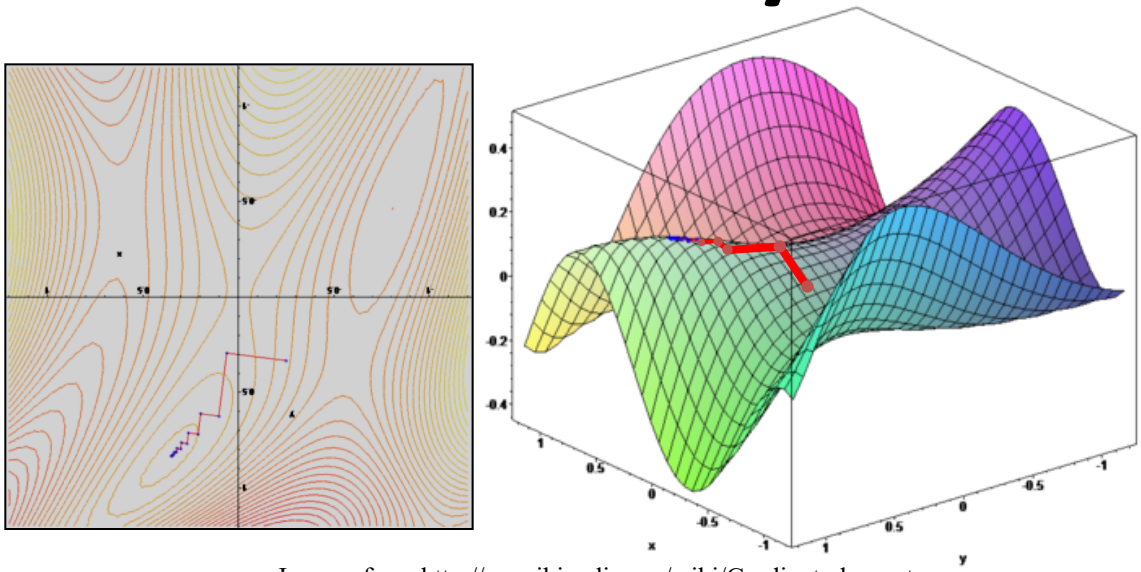
argmax and argmin

- The argmax and argmin concept is common in many AI and machine learning algorithms
- See argmax on Wikipedia (argmin is similar)

$$\arg\max_x f(x) := \{x \mid \forall y : f(y) \leq f(x)\}.$$

- Argmax_x f(x) finds the value of x for which f(x) is largest

Gradient ascent / descent



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

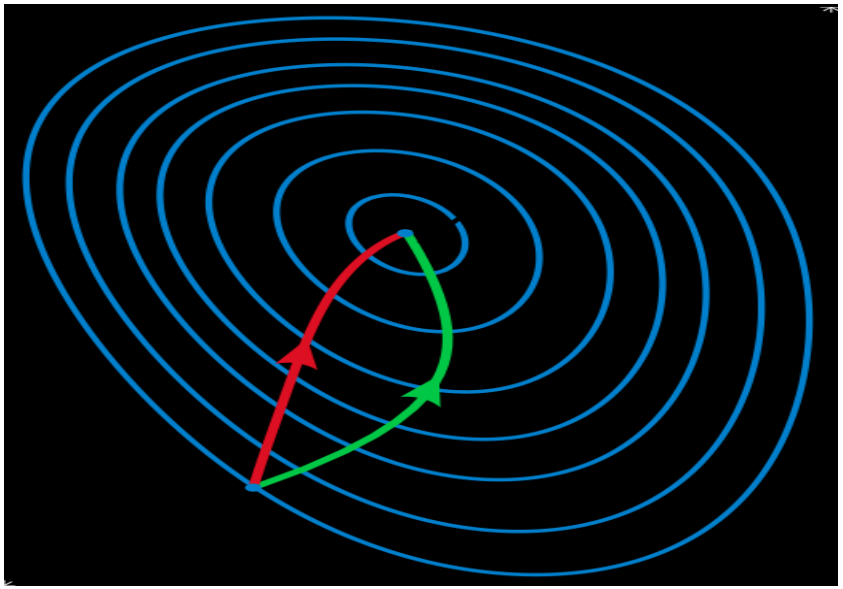
- Gradient descent procedure for finding the $arg_x\ min\ f(x)$
  - choose initial x_0 randomly
  - repeat
    - $x_{i+1} \leftarrow x_i - \eta\, f'(x_i)$
  - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size η (eta) is small (perhaps 0.1 or 0.05)
- Often used in machine learning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

Gradient methods vs. Newton's method

- 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 (e.g., 2nd derivative) to take a faster route to a minimum
- Second-order info. is more expensive to compute, but converges quicker
- See gradient descent



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

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

Annealing

- In metallurgy, annealing is a technique involving heating & controlled cooling of a material to increase size of its crystals & reduce defects
- Heat causes atoms to become unstuck from initial positions (local minima of internal energy) and wander randomly through states of higher energy
- Slow cooling gives them more chances of finding configurations with lower internal energy than initial one

Simulated annealing (SA)

- SA exploits analogy between how metal cools and freezes into a minimum-energy crystalline structure & search for a minimum/maximum in a general system
- SA can avoid becoming trapped at local minima
- SA uses random search accepting changes decreasing objective function f & some that **increase** it
- SA uses a control parameter T, which by analogy with the original application, is known as the system ***temperature***
- T starts out high and gradually decreases toward 0

SA intuitions

- Combines **hill climbing** (for efficiency) with **random walk** (for completeness)
- Analogy: getting a ping-pong ball into the deepest depression in a bumpy surface
  - Shake the surface to get the ball out of local minima
  - Don't shake too hard to dislodge it from global minimum
- Simulated annealing:
  - Start shaking hard (high temperature) and gradually reduce shaking intensity (lower temperature)
  - Escape local minima by allowing some "bad" moves
  - But gradually reduce their size and frequency

Simulated annealing

- A "bad" move from A to B is accepted with a probability

$$e^{-(f(B)-f(A)/T)}$$

- The higher the temperature, the more likely it is that a bad move can be made
- As T tends to zero, probability tends to zero, and SA becomes more like hill climbing
- If T lowered slowly enough, SA is complete and admissible

Simulated annealing algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: T, a "temperature" controlling the probability of downward steps

  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 $e^{\Delta E/T}$

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of T as a function of time.

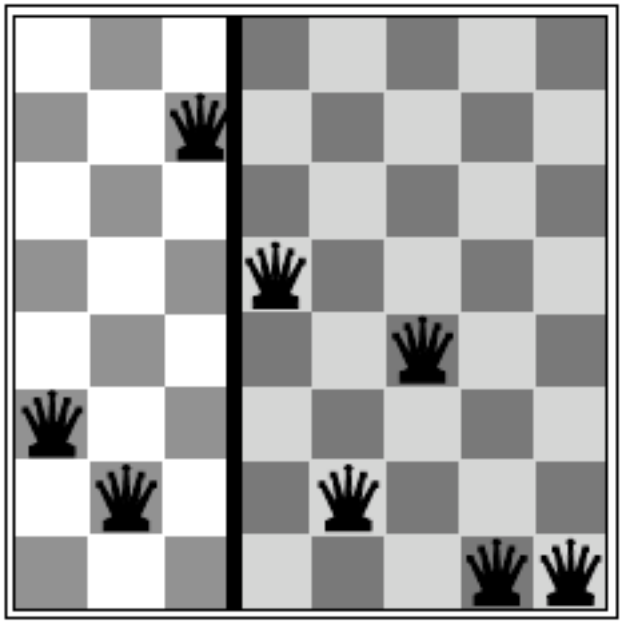
Local beam search

- Basic idea
  - Begin with k random states
  - Generate all successors of these states
  - Keep the k best states generated by them
- Provides a simple, efficient way to share some knowledge across a set of searches
- *Stochastic beam search* is a variation:
  - Probability of keeping a state is *a function* of its heuristic value

Genetic algorithms (GA)

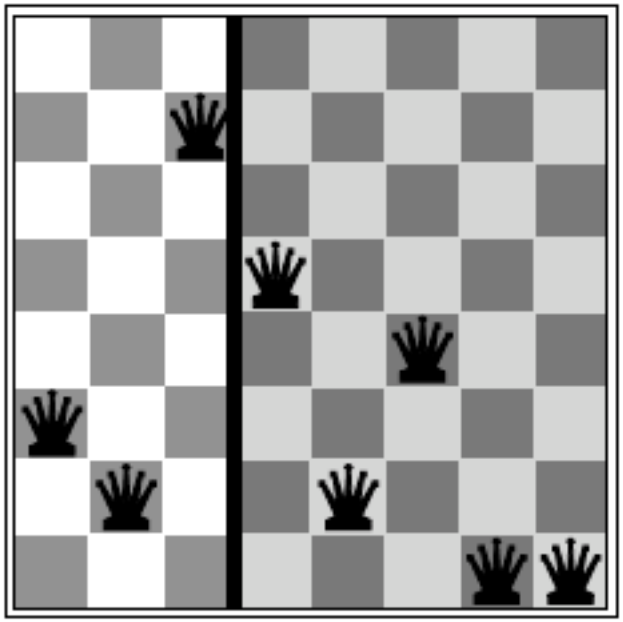
- Search technique inspired by *evolution*
- Similar to stochastic beam search
- Start with *initial population* of k random states
- New states generated by *mutating* a single state or *reproducing* (combining) two parent states, selected according to their *fitness*
- Encoding used for *genome* of an individual strongly affects the behavior of search

Ma and Pa solutions



8 Queens problem

- Represent state by a string of 8 digits in {1..8}
- S = ‘32752411’
- Fitness function = # of non-attacking pairs
- $F(S_{solution}) = 8*7/2 = 28$
- $F(S_1) = 24$



Genetic algorithms

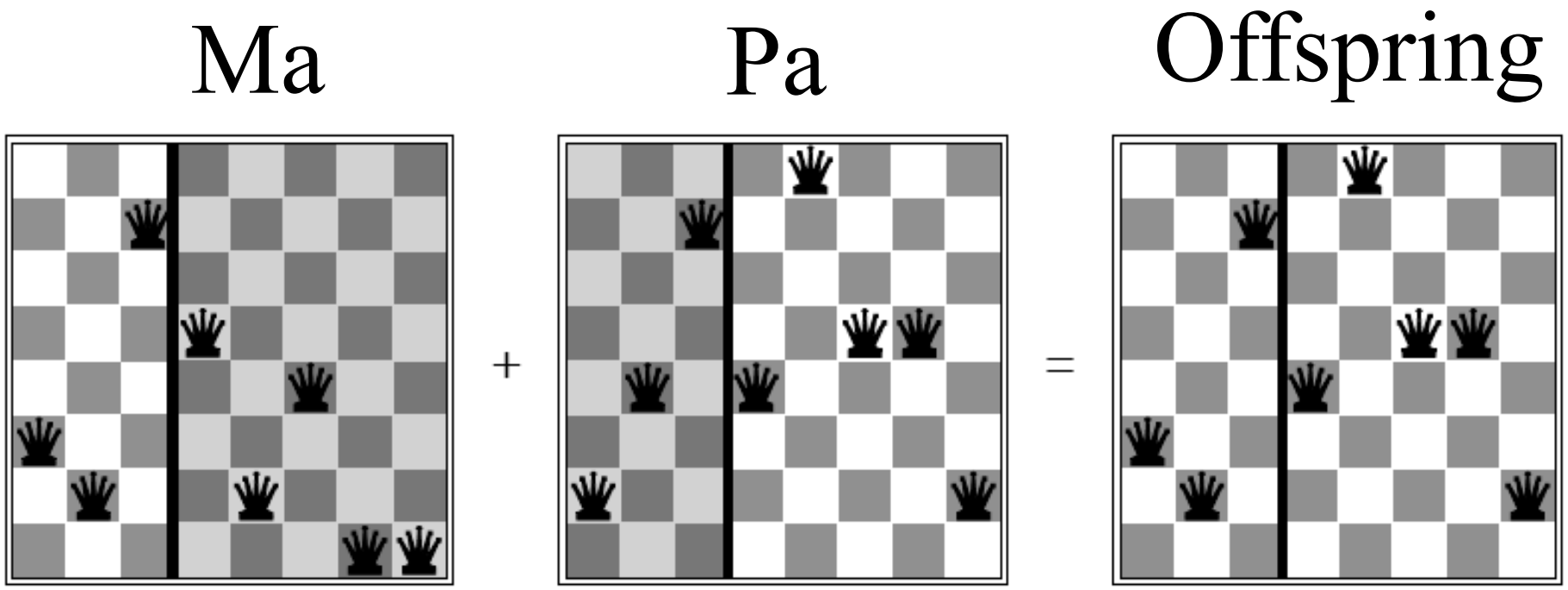


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The shaded columns are lost in the crossover step and the unshaded columns are retained.

Genetic algorithms

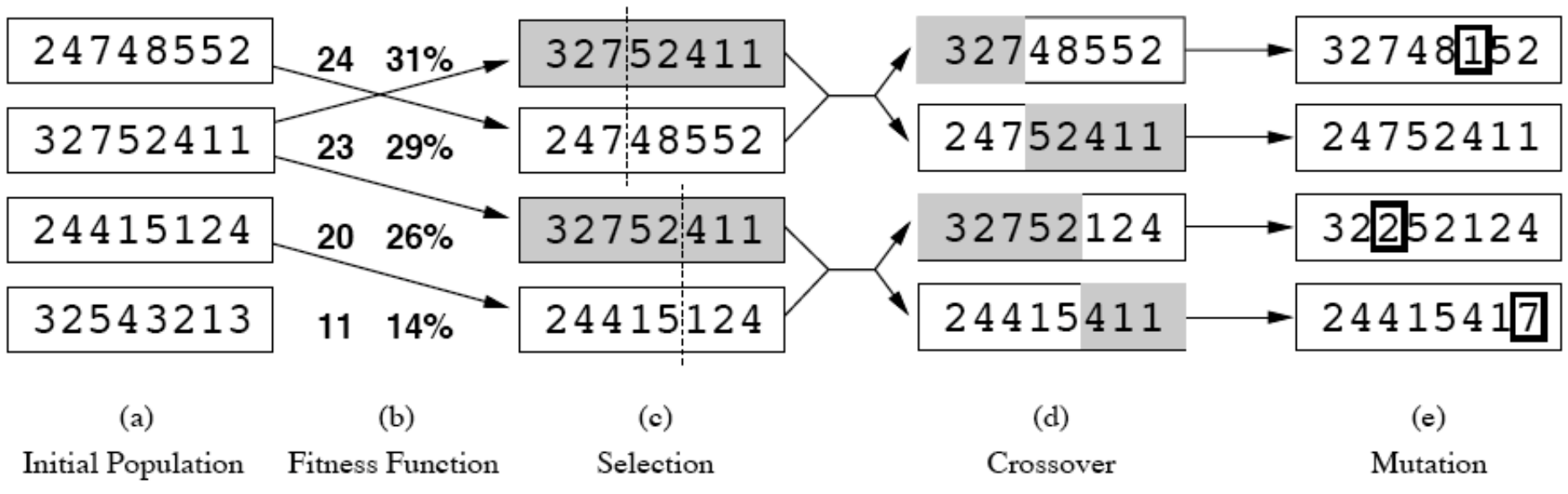


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

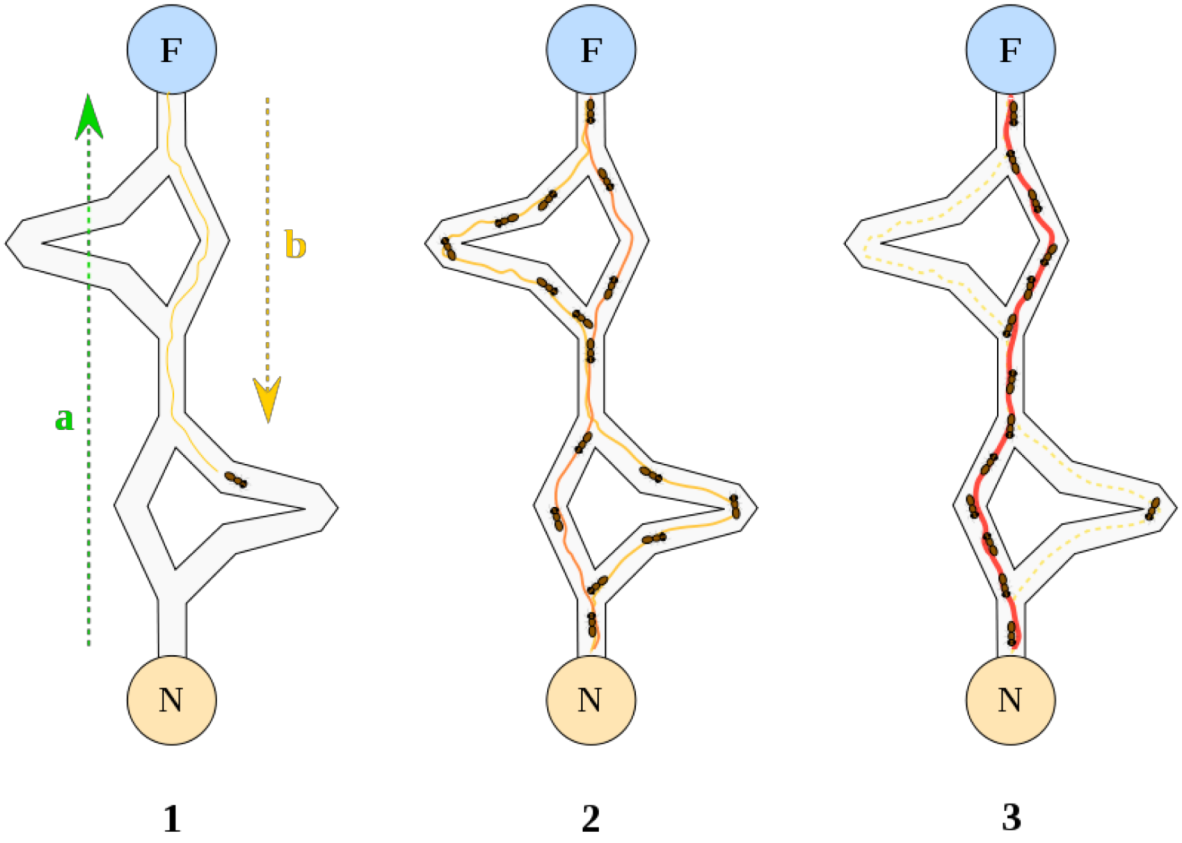
- Fitness function: number of non-attacking pairs of queens (min=0, max=$(8 \times 7)/2 = 28$)
- Probability of mating is a function of fitness score
- Cross-over point for a mating pair chosen randomly
- Resulting offspring subject to a random mutation with probability

Ant Colony Optimization

A probabilistic search technique for problems reducible to finding good paths through graphs

Inspiration

- Ants leave nest
- Discover food
- Return to nest, preferring shorter paths
- Leave pheromone trail
- Shortest path is reinforced



An example of agents communicating through their environment

Taboo search

- Problem: Hill climbing can get stuck on local maxima
- Solution: Maintain a list of *k* previously visited states, and prevent the search from revisiting them
- An example of a **metaheuristic**

Online search

- Nothing to do with Web search!
- Involves interleave computation AND action
  - search some, act some, repeat
- Exploration: Can't be sure of action outcomes; must perform them to know result
- Relatively easy if actions are reversible (ONLINE-DFS-AGENT)
- LRTA* (Learning Real-Time A*): Update h(s) (in state table) based on experience

Summary: Informed search

- **Hill-climbing algorithms** keep only a single state in memory, but can get stuck on local optima
- **Simulated annealing** escapes local optima, and is complete and optimal given a "long enough" cooling schedule
- **Genetic algorithms** can search a large space by modeling biological evolution
- **Online search** algorithms are useful in state spaces with partial/no information