Beyond Classical Search: Local Search

CMPSCI 383
September 23, 2011

Today's lecture

- Local Search
  - Hill-climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms
  - Genetic programming
  - Local search in continuous state spaces

Recall: Evaluating a search strategy

- **Completeness** — Does it always find a solution if one exists?
- **Optimality** — Does it find the best solution?
- **Time complexity**
- **Space complexity**

Example: Breadth-first search

- Complete?
  - Yes (if *b* finite)
- Optimal?
  - Yes, if cost = 1 per step
    Not optimal in general
- Time
  - $1+b+b^2+b^3+\ldots+b^d+b(b^d-1) = O(b^{d+1})$
- Space
  - $O(b^{d+1})$

Is $O(b^{d+1})$ a big deal?

Depth	Time	Memory
2	.11 sec	1 megabyte
4	11 sec	106 megabytes
6	19 min	10 gigabytes
8	31 hours	1 terabytes
10	129 days	101 terabytes
12	35 years	10 petabytes
14	3523 years	1 exabyte

How can we ease up on ***completeness*** and ***optimality*** in the interest of improving ***time and space complexity***?

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

What is unique about local search?

- Many search problems only require finding a goal state, not the path to that goal state
- Examples
  - Actual physical (vs. virtual) navigation
  - Configuration problems — e.g., n-Queens problem, determining good compiler parameter settings
  - Design problems — e.g., VLSI layout or oil pipeline network design
- State space is set of configurations.

Example: *n*-queens

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

Optimization Problems

Aim is to find the best state according to an objective function.

No goal test
No path cost

cf. Evolution

Why use local search?

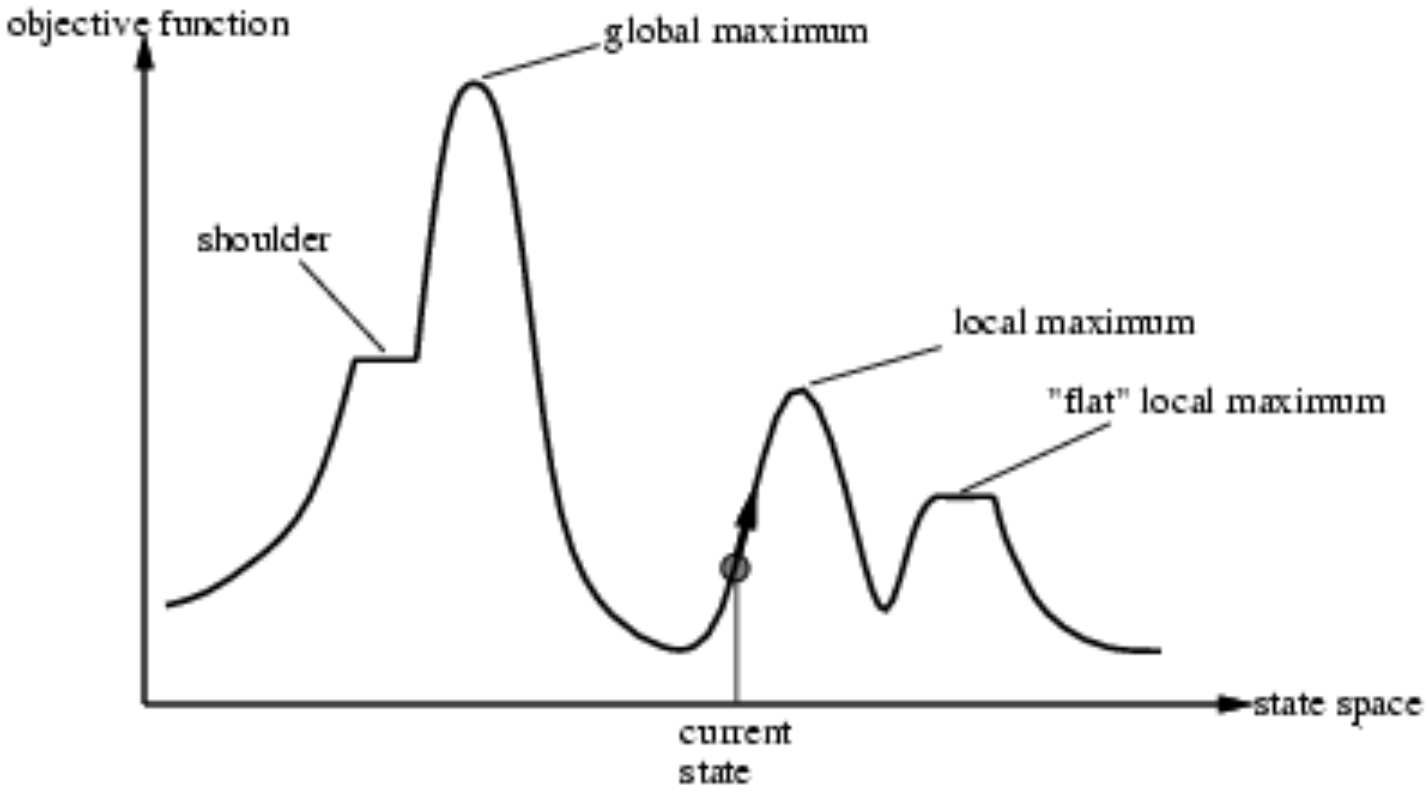
- Low memory requirements

  Usually constant

- Effective

  Can often find good solutions in extremely large state spaces

State-space landscape



Hill-climbing search ("Steepest Ascent" version)

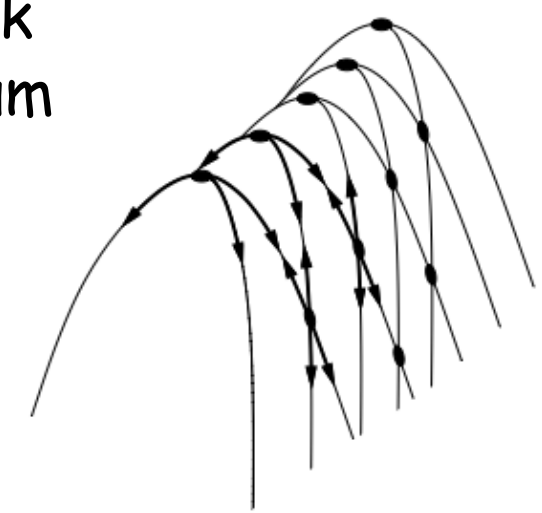
- Like "trying to find the top of Mount Everest in a thick fog while suffering from amnesia."

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Challenges for hill climbing

- Local maxima
  - Once a local maximum is reached, there is no way to backtrack or move out of that maximum
- Ridges
  - Ridges can produce a series of local maxima
- Plateaux
  - Hillclimbing can have difficult time finding its way off of a flat portion of the state space landscape



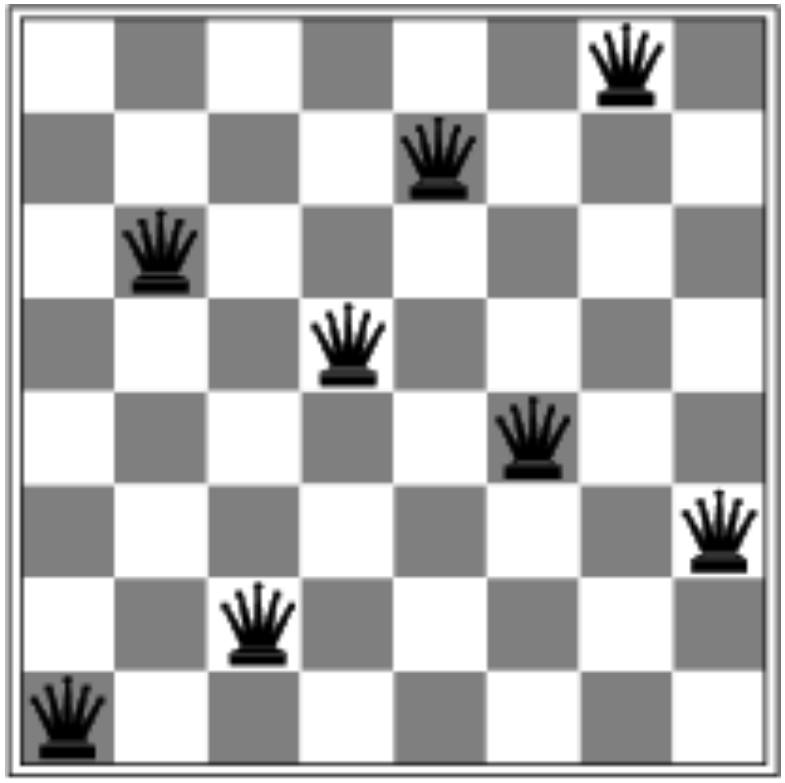
Hill-climbing search: 8-queens problem

Complete-state formulation

Actions: move a single queen to any other square in same column

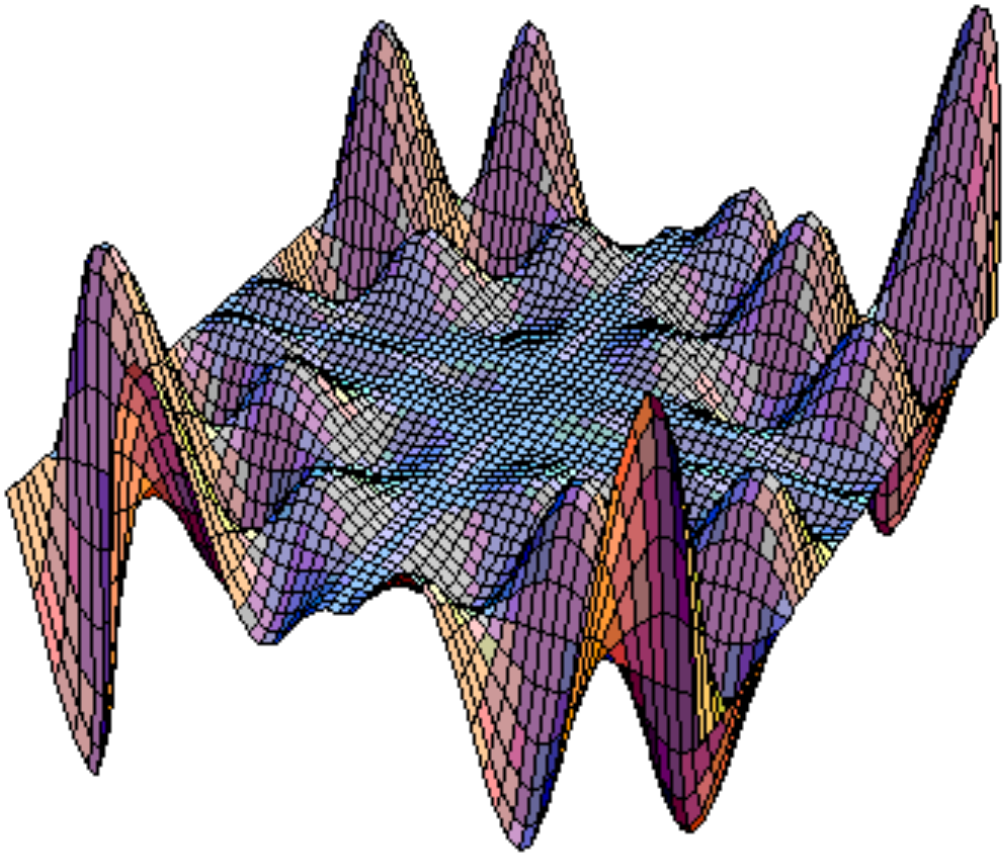
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-climbing search: 8-queens problem



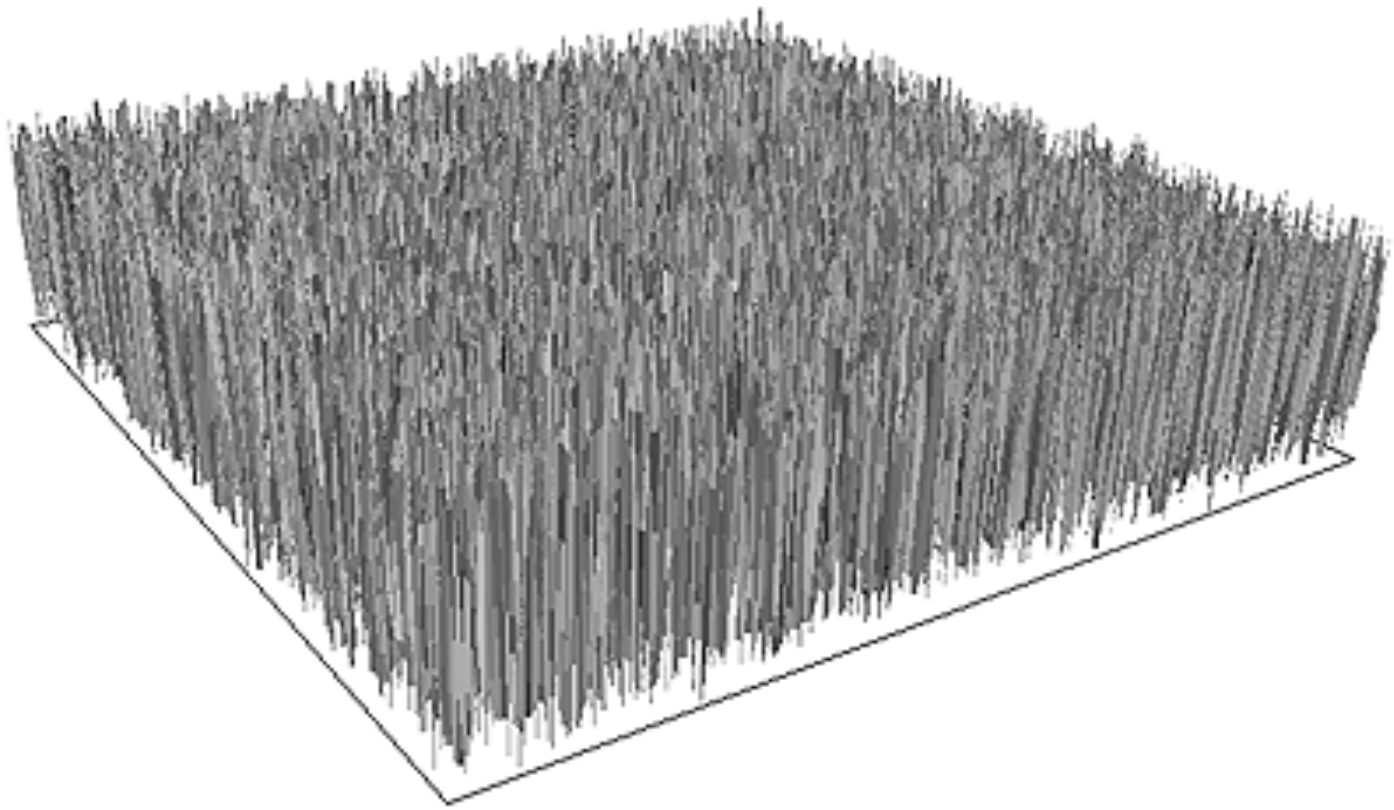
- *A local minimum with h = 1*

A state-space landscape



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

Another state-space landscape



From: Richard Wein (2002). Not a Free Lunch But a Box of Chocolates. http://www.talkorigins.org/design/faqs/nfl/

Variants of local hill climbing

- Stochastic hill climbing
  - Select randomly from all moves that improve the value of the evaluation function
- First-choice hill climbing
  - Generate successors randomly and select the first improvement
- Random-restart hill climbing
  - Conducts a series of hill-climbing searches, starting from random positions
  - Very frequently used general method in AI

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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)
```

Properties of simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc.

Local beam search

- Analogy
  - Hill-climbing — "…trying to find the top of Mt. Everest in a thick fog while suffering amnesia."
  - Local beam search — Doing this with several friends, each of whom has a short-range radio and an altimeter.

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat
- Stochastic beam search — Select successors at random weighted by value

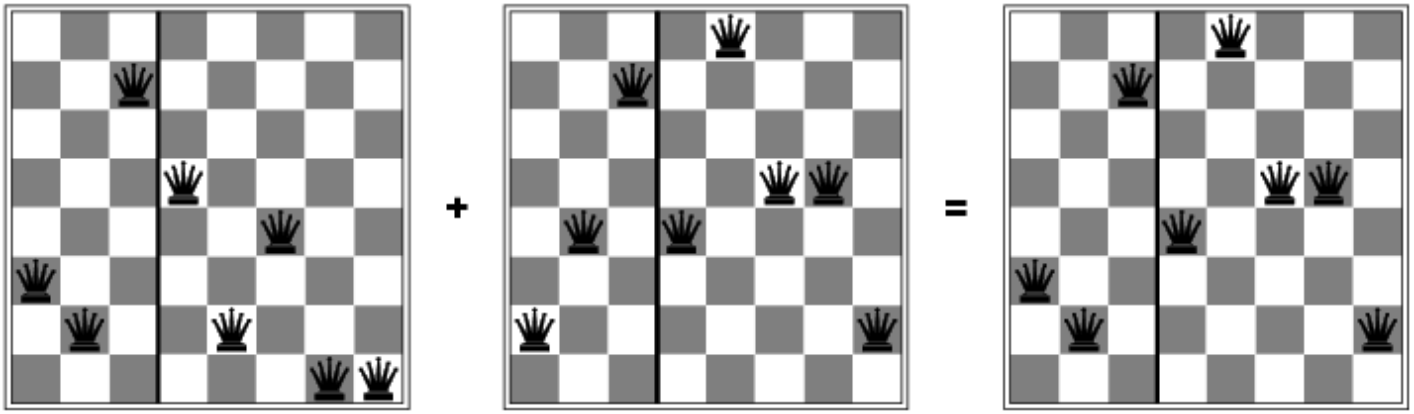
Genetic algorithms

- A variant of stochastic beam search where new states are generated by combining existing states
- Basic components
  - *Population* — A set of states, initially generated randomly
  - *Fitness function* — An evaluation function
  - *Reproduction* — A method for generating new states from pairs of old ones (e.g., crossover)
  - *Mutation* — A method for randomly modifying states to create variation in the population

Genetic algorithms

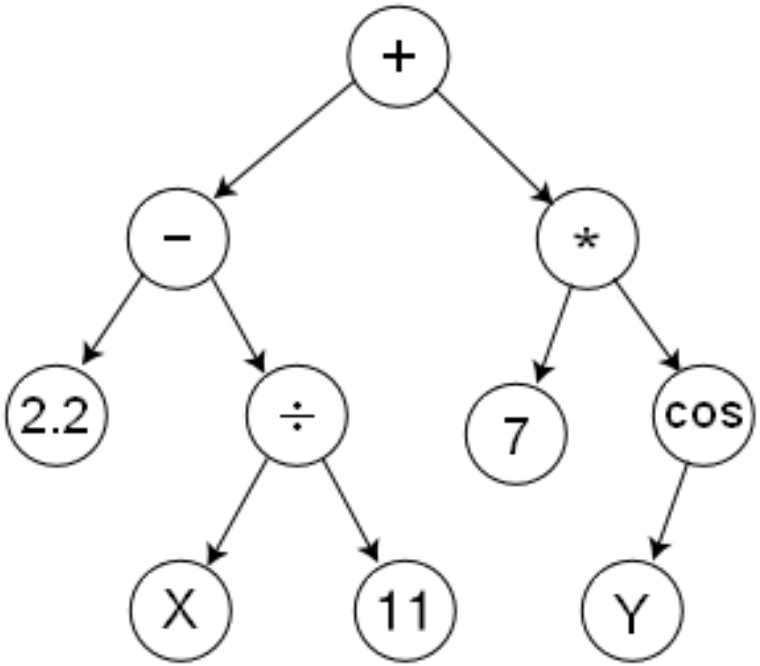
- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Genetic Programming

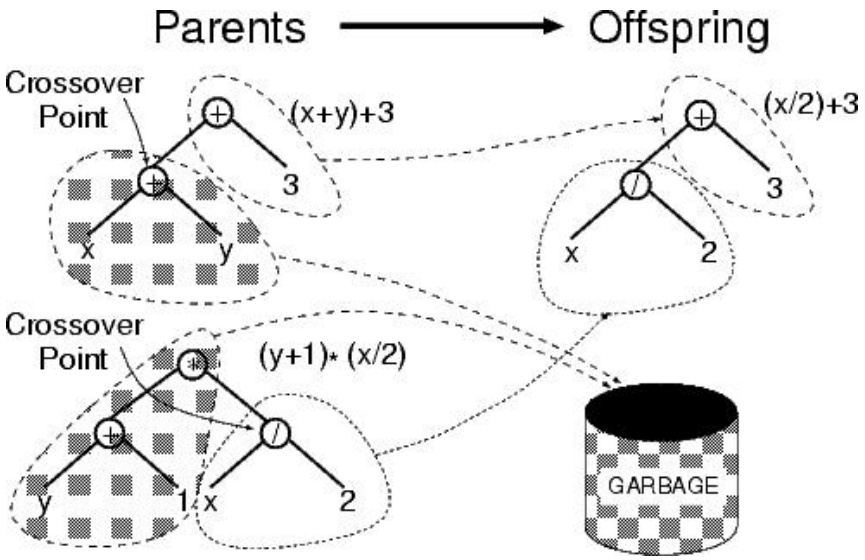
- A specialization of genetic algorithms where each individual is a program



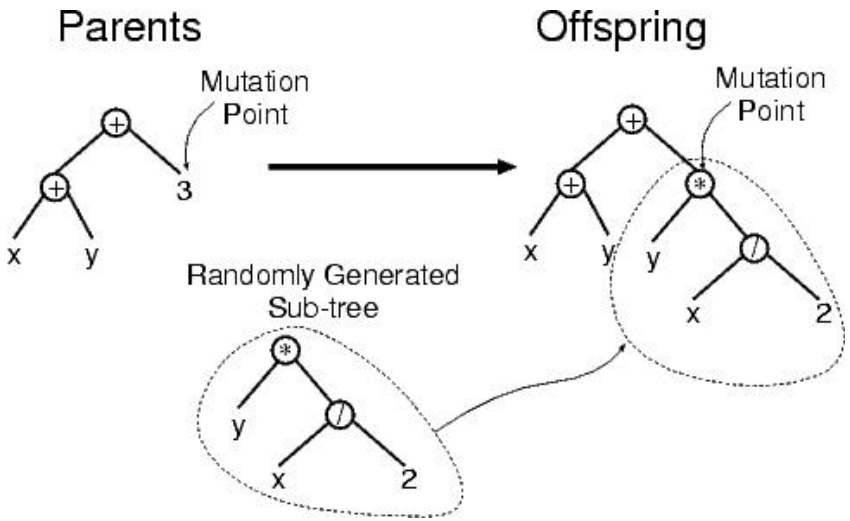
$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

GP Genetic Operators

- Subtree crossover



- Subtree mutation



Local Search in Continuous Spaces

- Spaces consisting of real-valued vectors
- Can discretize the space
- Gradient Ascent (descent)
- Line search
- Newton-Raphson
- Constrained optimization
  - Linear Programming
  - Convex optimization

Gradient Methods

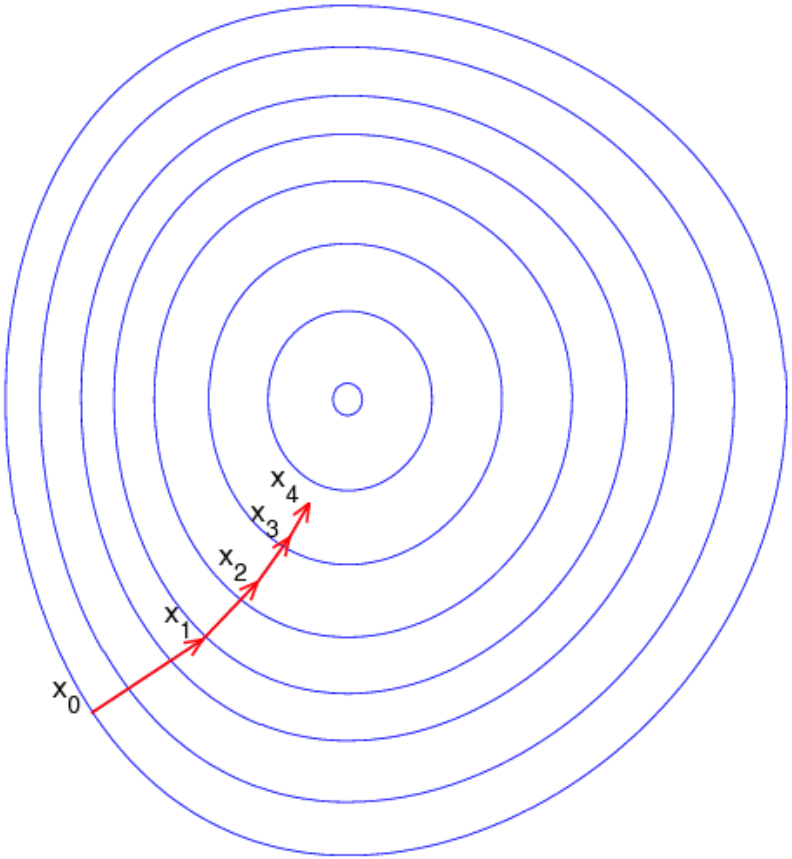
Gradient methods compute

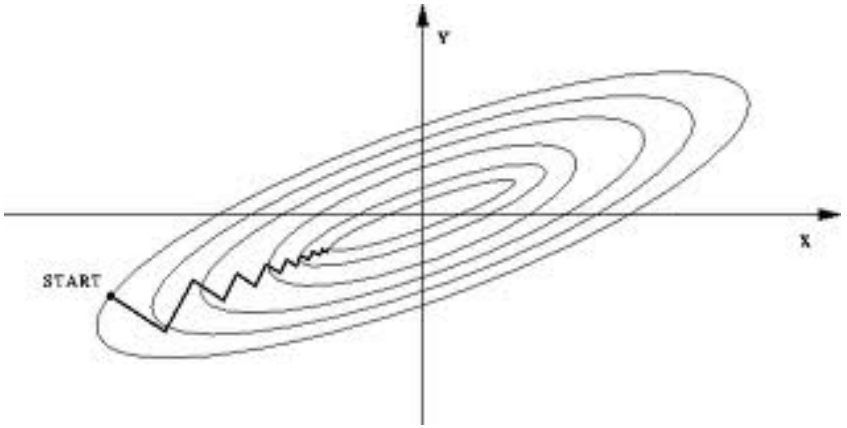
$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

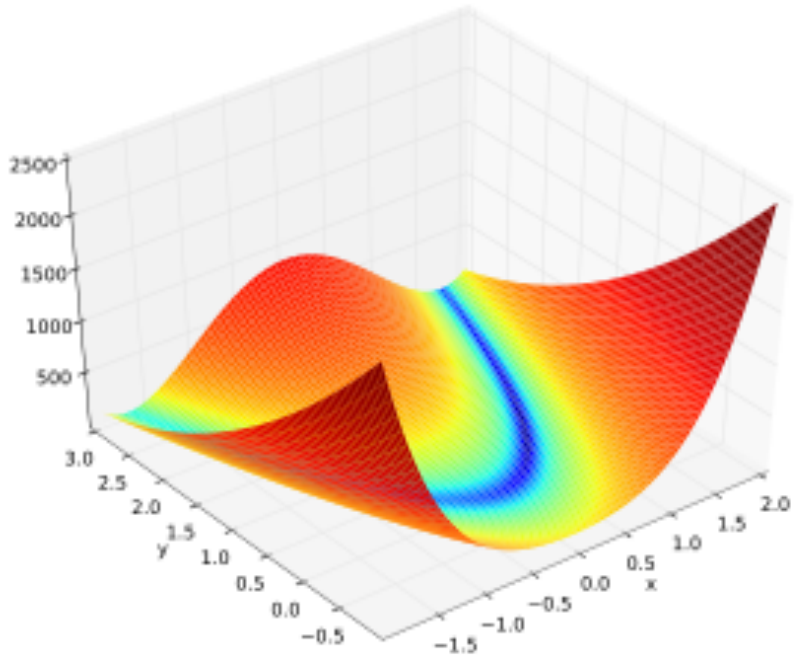
Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

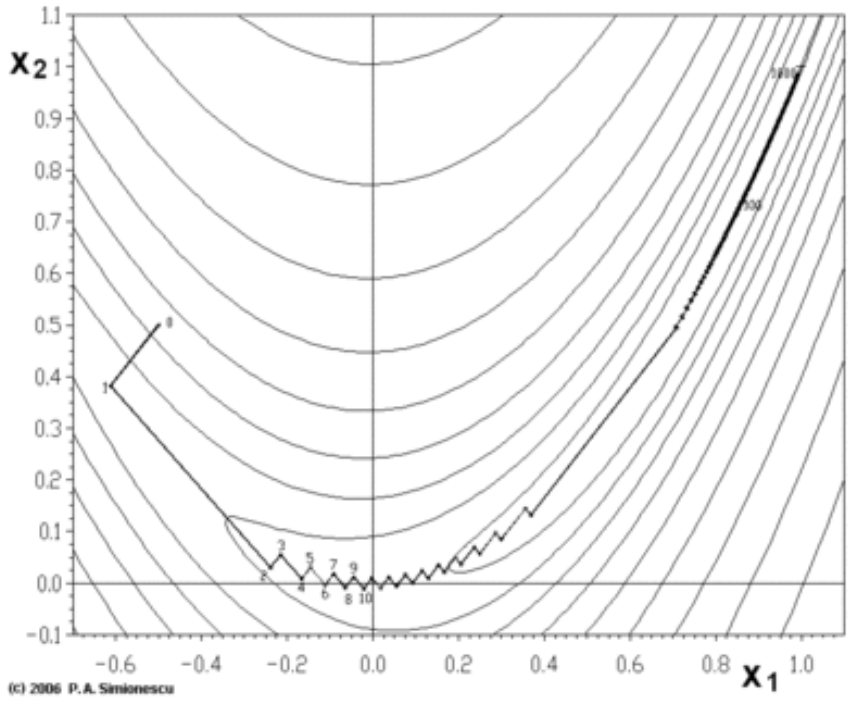
Gradient Descent



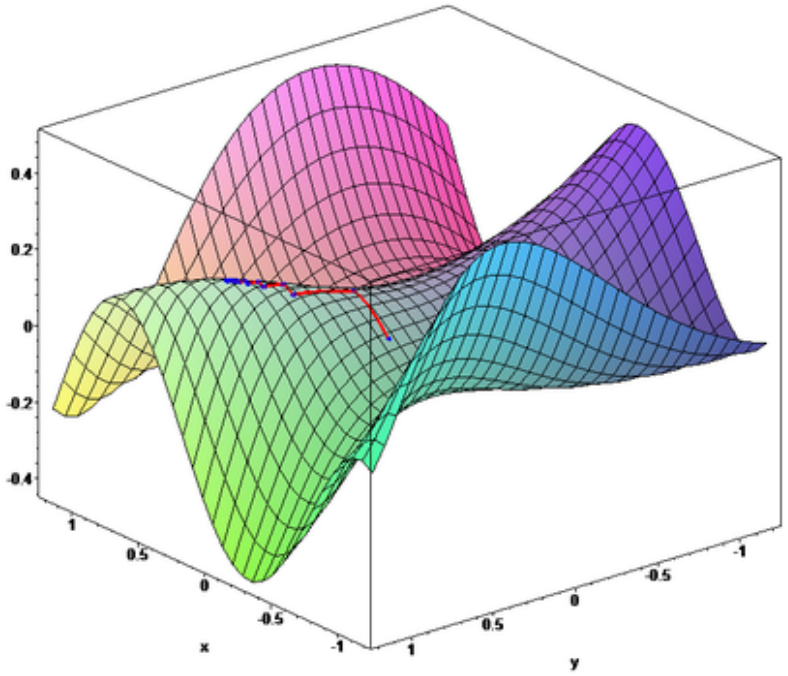


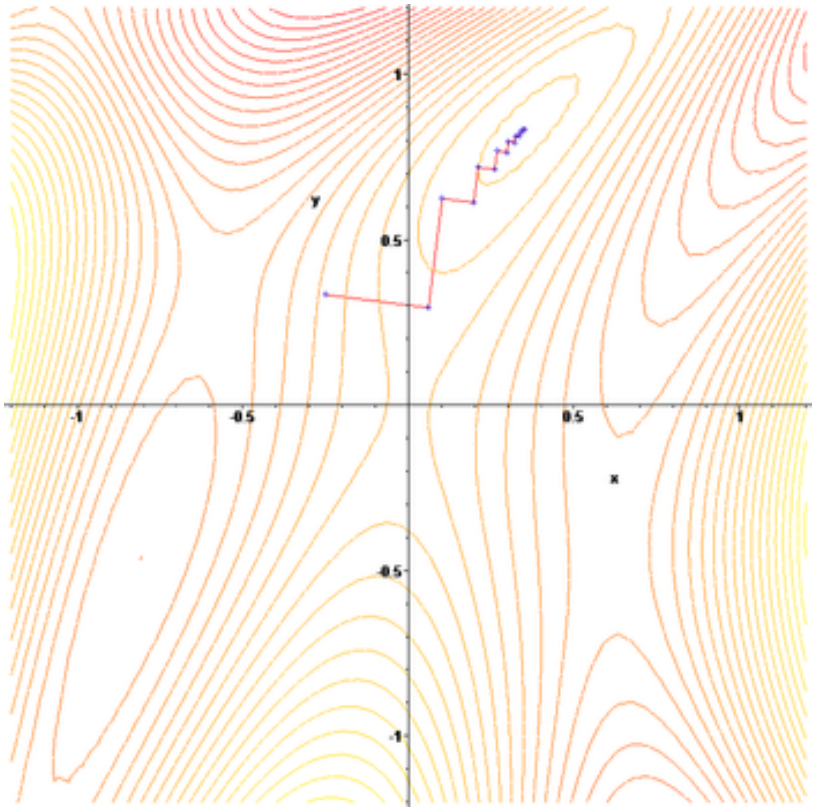
Gradient Descent: Rosenbrock Function



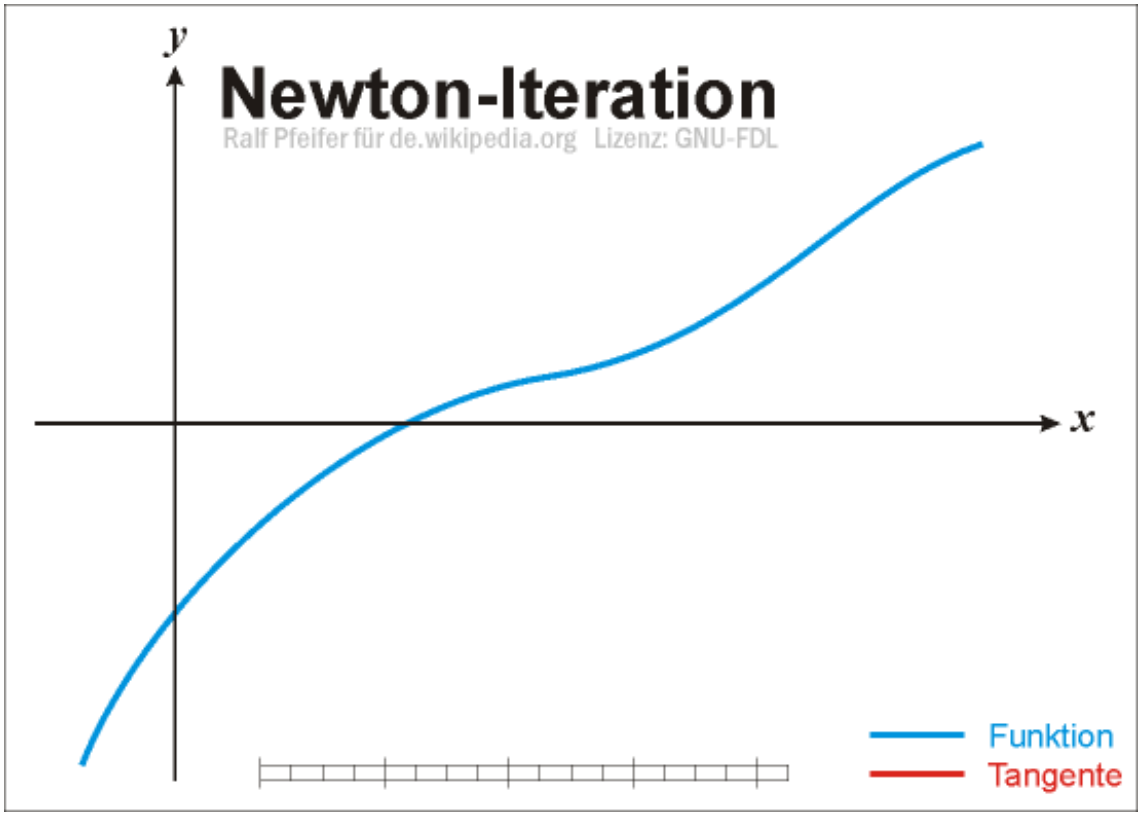


Gradient Descent

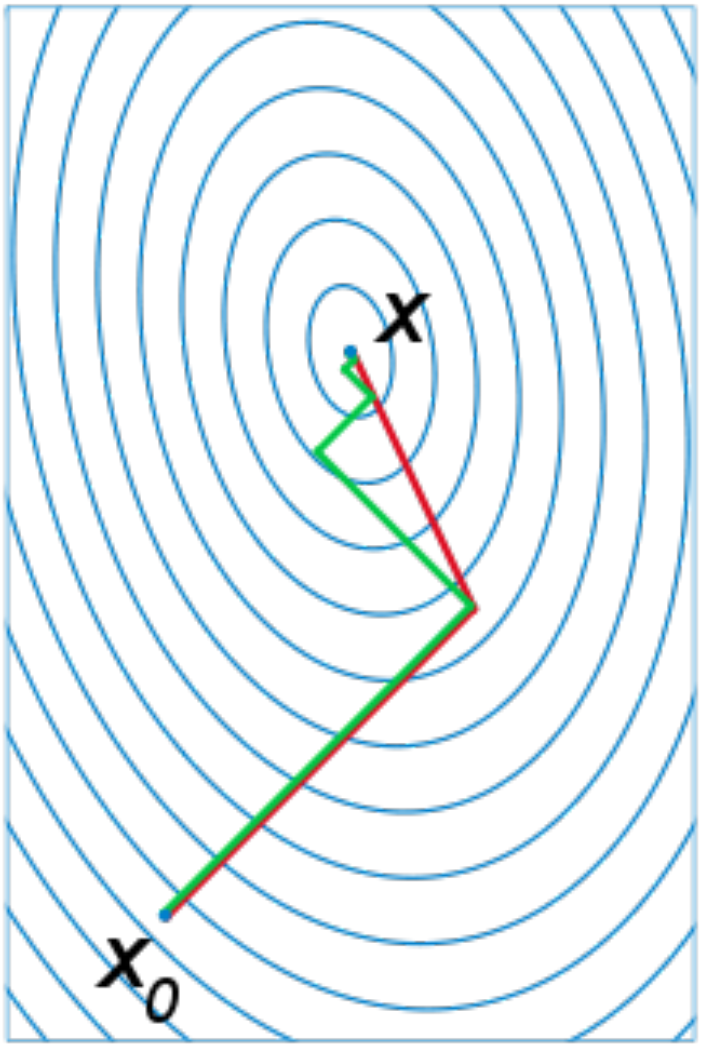




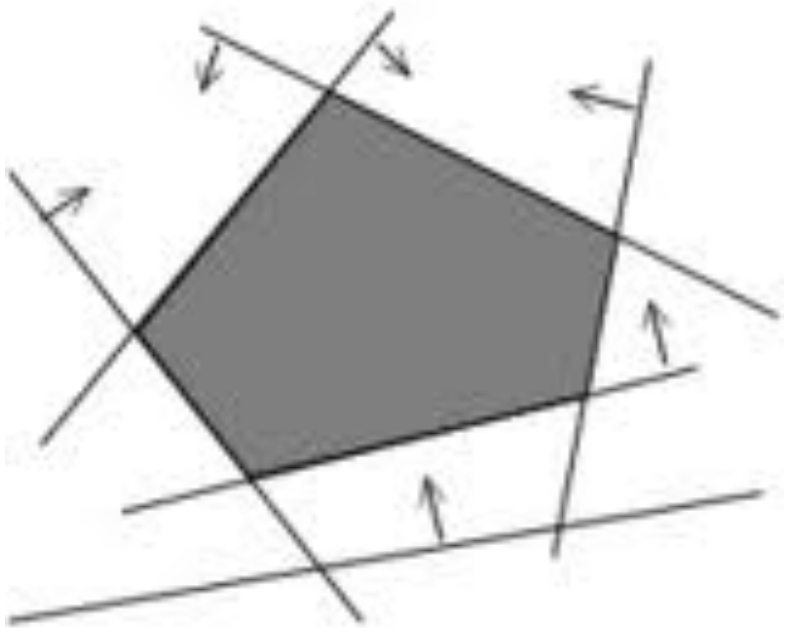
Newton Raphson



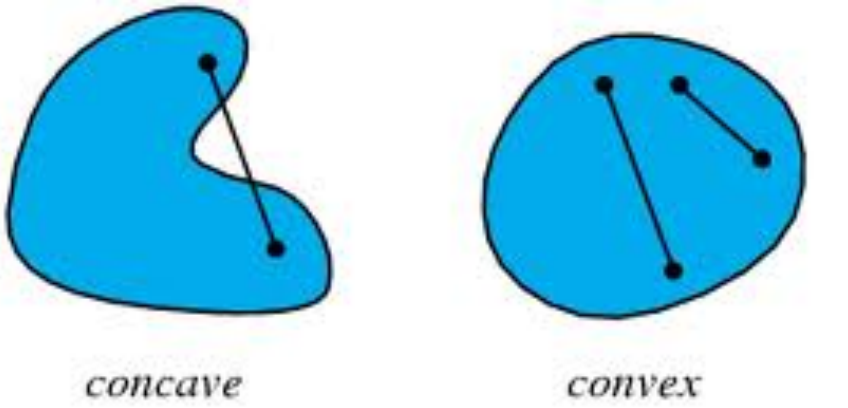
Conjugate Gradient Method

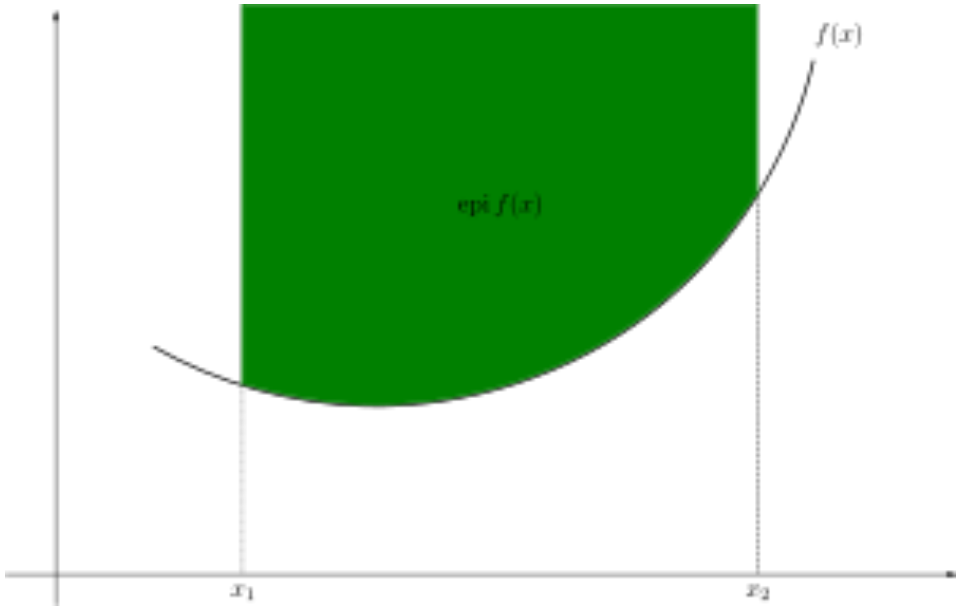


Linear Programming



Convex Optimization





Local Search

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Next Class

- Nondeterministic Actions and Partial Observations; Online Search
- Sec. 4.3, 4.4, 4.5