Continuous Optimization

2/3/17

Review: discrete local search

Requirements:

- Generate initial candidate
- Generate neighbor candidates
- Evaluate cost/value of a candidate

Algorithms:

- Hill climbing
- Simulated annealing
- Beam search
- Genetic algorithm

Building up the algorithms

Hill climbing: move up the value hill

- In the steepest possible direction (best neighbor)
- In a random uphill direction (random neighbor)

Question: what differences does this choice make?

Problem: getting stuck in local optima

Solutions:

- Random restarts (get onto a different mountain)
- Random moves (get over small hills)

Building up the algorithms

Simulated annealing: fewer random moves over time

- Like hill climbing with random neighbor AND random moves
- As temperature decays, the probability of a random move goes down.

Question: why take more random moves early?

- No random restart probability; instead run it several times.

Building up the algorithms

Beam search: focus on more important regions

- Track a bunch of states at once (like running several HC searches in parallel)
- Instead of picking one neighbor of each state, pick the best subset of all states' neighbors.

Problems:

- We still want random moves
- Focuses *too* fast

Solution (stochastic beam search):

- Don't pick strictly the best neighbors
- Pick randomly, give better states higher probability

Building up the algorithms

Genetic algorithm: combine states via crossover

- As long as we're tracking a population of states, maybe we can combine partial solutions from different ones.
- Instead of generating neighbors by small changes, generate offspring by crossover.
- Requires us to define crossover for our states.
- We still want random perturbations, so do mutations.

- There are many variants on GAs, some more and some less connected to the biological inspiration.

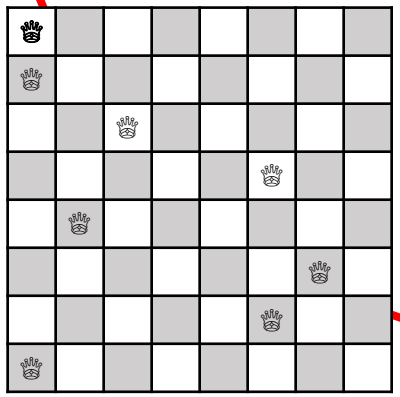
Satisfying vs. Optimizing

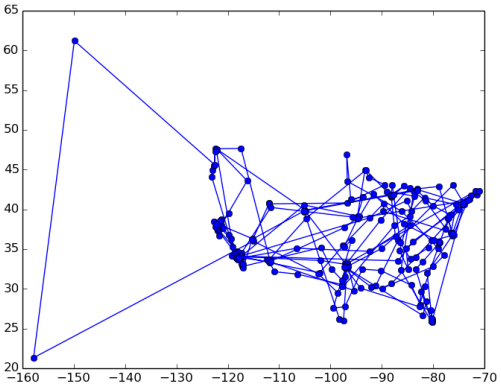
search for states that **satisfy constraints**

state = an assignment of values to variables

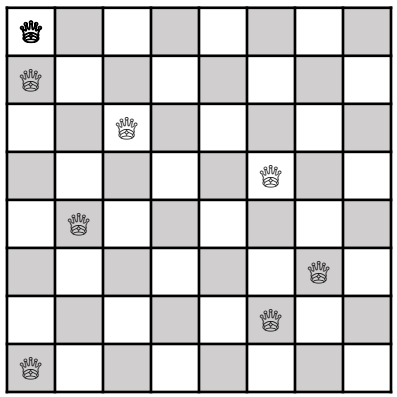
constraints specify allowed states

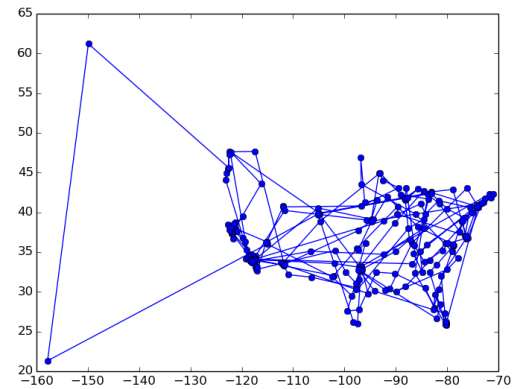
search for states that satisfy constraints **and optimize objective**





Variables, constraints, and objective





Variables: $Q_0 \ldots Q_{N-1}$

Constraints:

$Q_0 \neq Q_1$, $Q_0 \neq Q_2$, …

$Q_0 \neq Q_1$, $Q_0 \neq Q_2$, …

$Q_0 \neq Q_1 + 1$, $Q_0 \neq Q_2 + 2$, …

Variables: $C_0, \ldots C_{N-1}$

Constraints: $C_0 \neq C_1$, $C_0 \neq C_2$, …

Objective: minimize

$\text{cost}(C_0,C_1) + \text{cost}(C_1,C_2) + \ldots$

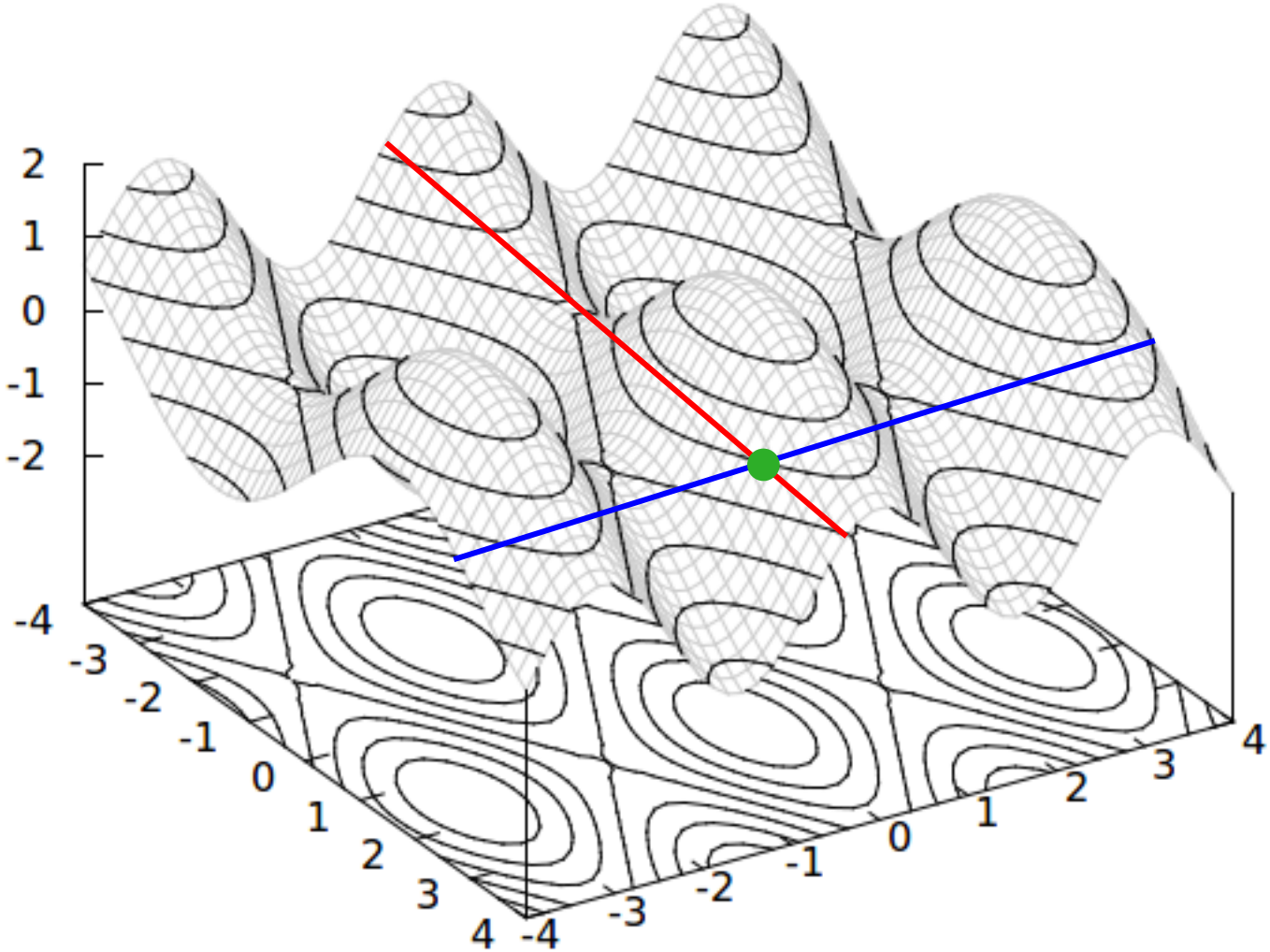
Discrete vs Continuous Optimization

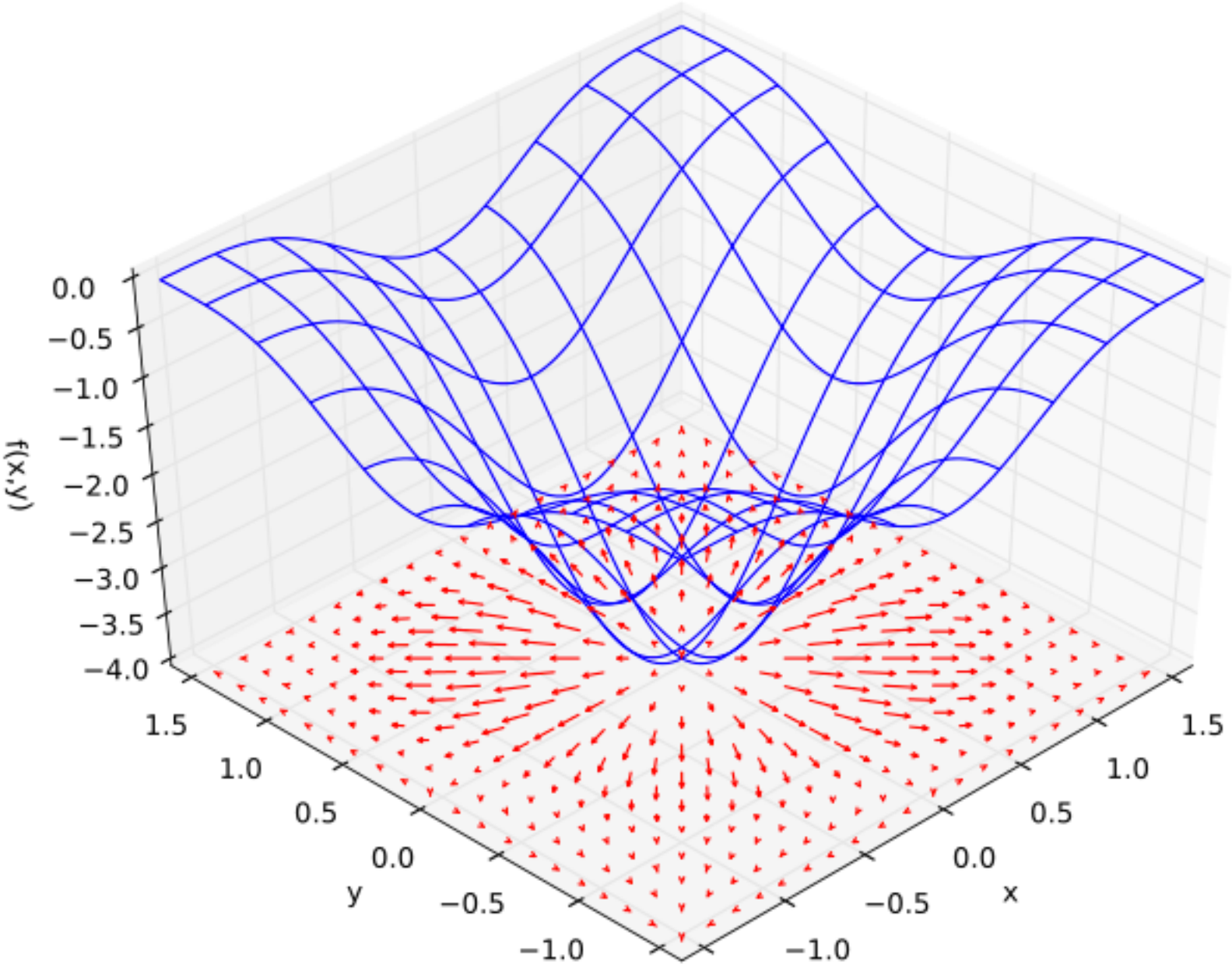
All our examples so far have had discrete variables:

- rows
- colors
- cities

Sometimes the variables can be continuous:

- Motion planning in continuous space.
- Inputs/outputs in a manufacturing supply chain.
- Security personnel positioning to defend against terrorist attacks.
- Electricity transmission on a power grid.

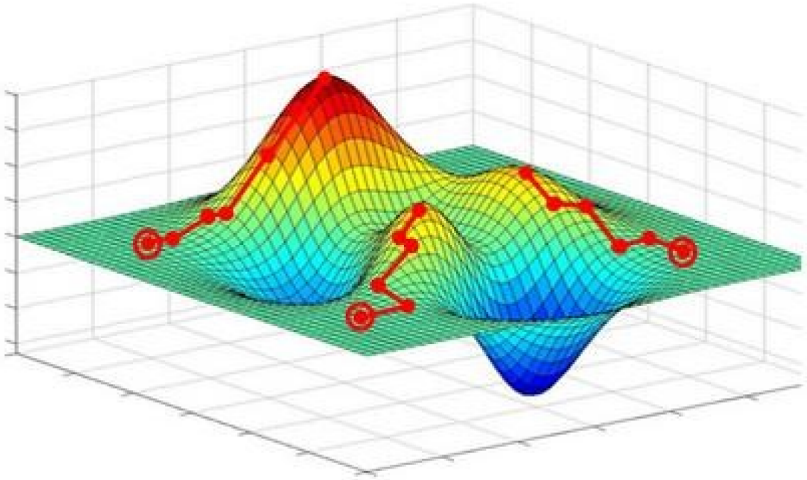

2
1
0
-1
-2
-4
-3
-2
-1
0
1
2
3
4
-4
-3
-2
-1
0
1
2
3
4


0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
f(x,y)
1.5
1.0
0.5
0.0
−0.5
−1.0
y
1.5
1.0
0.5
0.0
−0.5
−1.0
x

Gradient Ascent

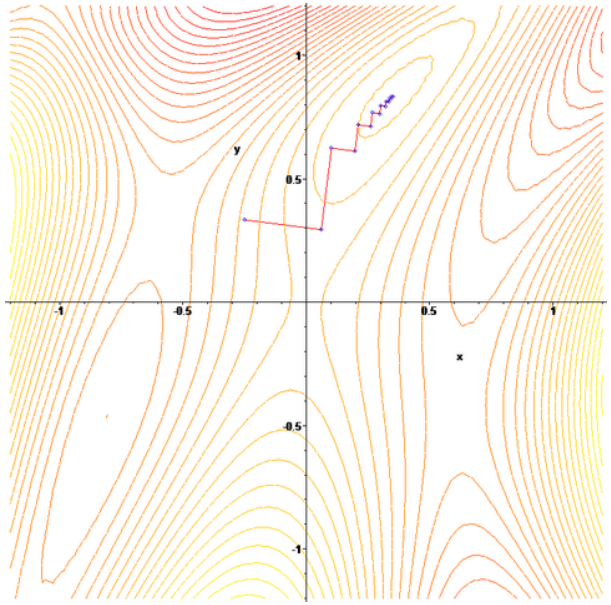
General idea:

- Start from a random point.



- Compute the gradient.

- Take a step in that direction.



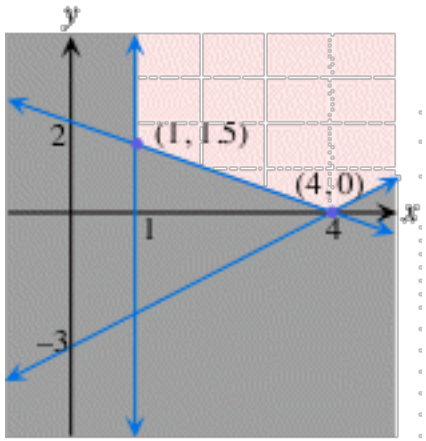
- Recompute gradient and repeat.

Linear Programming

If your objective and all of your constraints are linear functions:

$5x + 6y \leq 5$

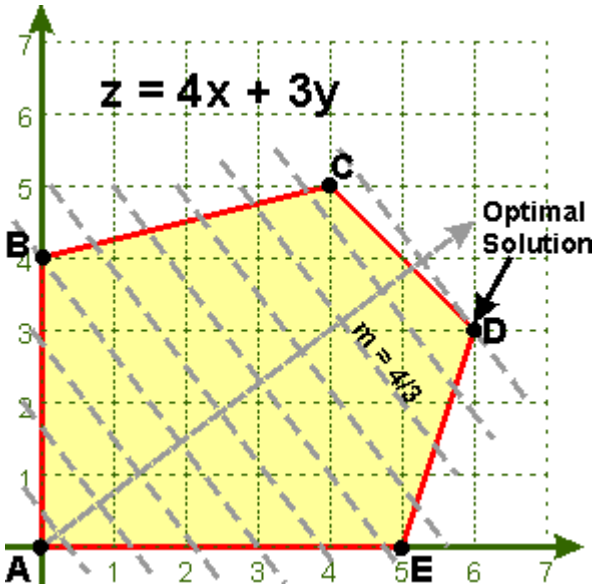
minimize: $2x - z$



Then continuous optimization is “easy”.

We have good linear program solvers.

We can also handle some non-linear functions if they’re convex.



Continuous optimization takeaways

- We'll see gradient ascent (descent) again.

- If you can formulate an optimization problem as a linear program, you should.
  - LP solvers are very powerful!
  - This is different way of thinking about "programming": describe the problem in terms of equations and then the LP solver will spit out an answer.

- You don't need to know algorithm details here, just the big ideas.