Continuous Methods in Computer Science: from Random Graph Coloring to Linear Programming

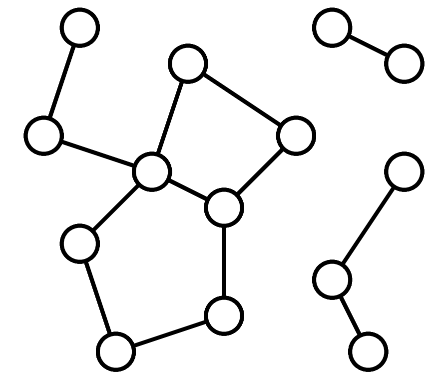
Cristopher Moore
University of New Mexico
and the Santa Fe Institute

From discrete to continuous

- Computer science students get very little training in continuous mathematics
- As a result, they are intimidated by continuous methods, just as students from other fields are intimidated by discrete math and theoretical computer science
- This is unfortunate—no one should be intimidated by anything
- Moreover, continuous methods (e.g. differential equations) arise in several ways in computer science:
- As limits of discrete processes, especially in random structures
- As a compelling picture of some of our favorite optimization algorithms

Random graphs

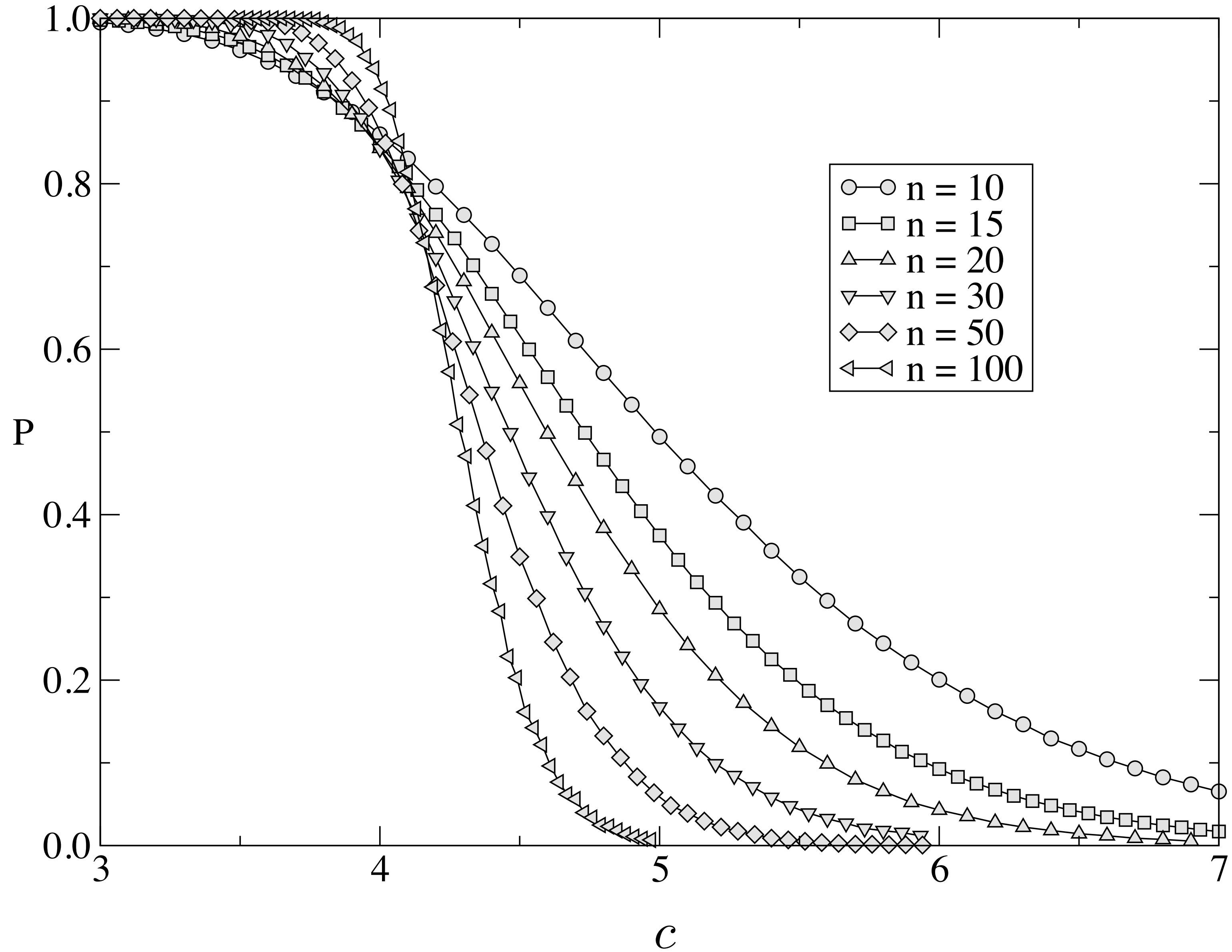
- Erdős-Rényi random graph $G(n,p)$: each pair of vertices is independently connected with probability p



- Sparse case: $p=c/n$, so the average degree is c
- Emergence of the giant component—a phase transition:
  - when $c<1$, with high probability the largest component has size $O(\log n)$, and all but $O(\log n)$ of these components are trees
  - when $c=1$, the largest component has size $O(n^{2/3})$; power-law distribution of component sizes
  - when $c>1$, with high probability there is a unique component of size an, where $a=a(c)$. Other components are mostly trees, like the $c<1$ case.

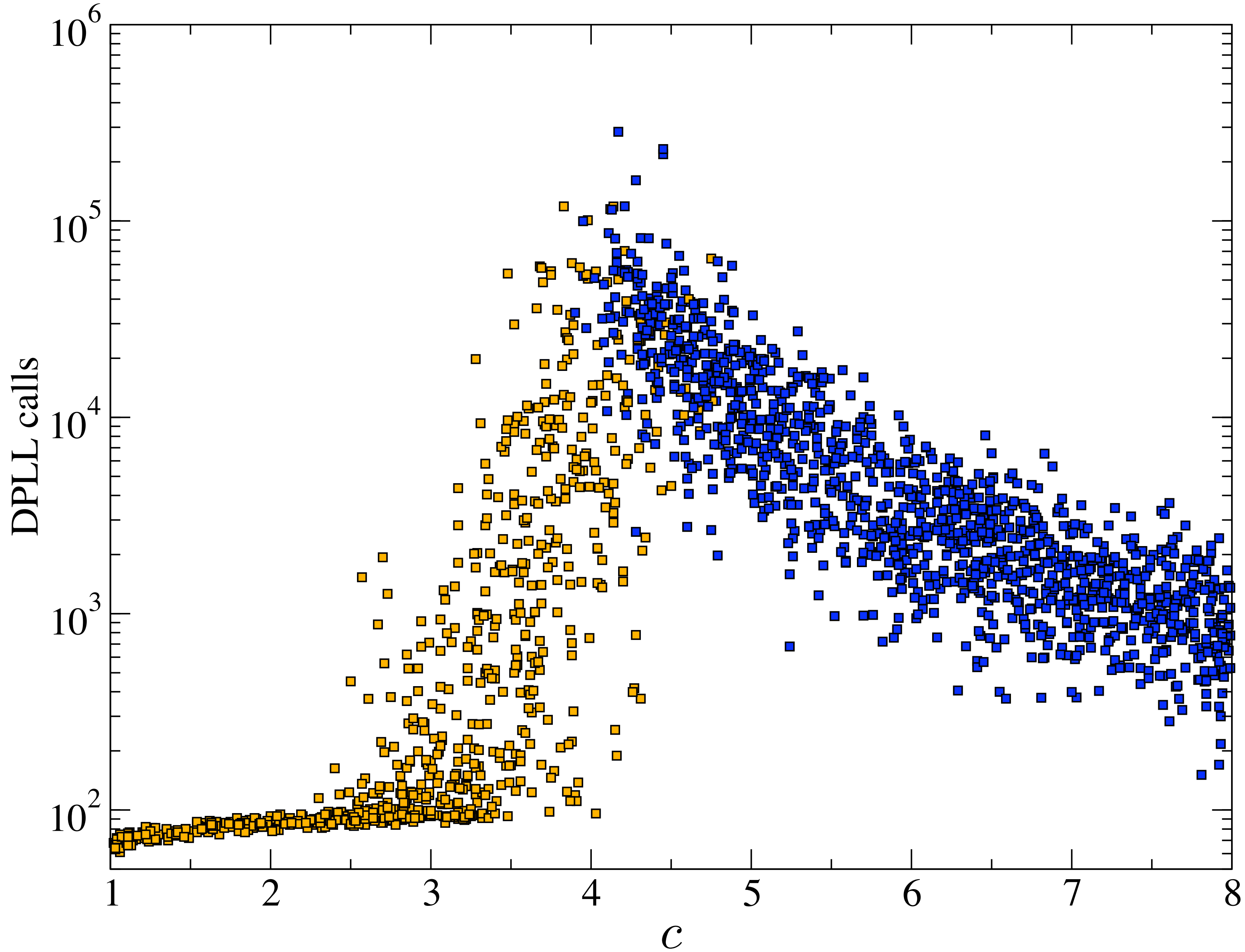
Coloring random graphs

- How often is a random graph 3-colorable? Experimental results:



Search times

- The hardest instances (among the random ones) are at the transition:



The colorability conjecture

- There is a constant c^* such that

$$\lim_{n\to\infty} \Pr[G(n, p = c/n) \text{ is 3-colorable}] = \begin{cases} 1 & \text{if } c < c^\star \\ 0 & \text{if } c > c^\star \end{cases}$$

- Known in a "non-uniform" sense [Friedgut]: namely, there is a function $c(n)$ such that, for all ϵ>0,

$$\lim_{n\to\infty} \Pr[G(n, p = c/n) \text{ is 3-colorable}] = \begin{cases} 1 & \text{if } c < (1-\epsilon)c(n) \\ 0 & \text{if } c > (1+\epsilon)c(n) \end{cases}$$

- But we don't know that $c(n)$ converges to a constant.

An easy upper bound

- Compute the expected number of colorings, E[X].
- If there are *m=cn/*2 edges, then

$$\mathbb{E}[X] \leq 3^n (2/3)^m = \left(3(2/3)^{c/2}\right)^n$$

- This is exponentially small when

$$c > 2 \log_{3/2} 3 \approx 5.419$$

- Markov's inequality: Pr[3-colorable] is exponentially small too.
- So $c^\star \leq 5.419$. Arguments from physics give $c^\star \approx 4.69$.

How to get a lower bound?

- The second moment method: for any random variable $X \in \{0,1,2,...\}$,

$$\Pr[X > 0] \geq \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}$$

- Tricky calculations: we need to compute second moment, i.e., correlations
- Works very well for *k*-SAT for large *k*, and *q*-coloring for large *q* [Frieze and Wormald, Achlioptas and {Moore, Peres, Naor}]
- For 3-coloring, best lower bounds are algorithmic
- Algorithm lower bounds are also *constructive*: a proof that a polynomial-time algorithm usually works.

A greedy algorithm

- At each point, each vertex has 3, 2, or 1 available colors
- If any vertex has only 0 available colors, give up (no backtracking)

```
while there are uncolored vertices {
  if there are any 1-color vertices
    choose one and color it
  else if there are any 2-color vertices
    choose one, flip a coin, and color it
  else choose a random 3-color vertex,
    choose a random color, and color it
}
```

- For what values of *c* does this algorithm work with probability $\Omega(1)$?

The expected effect of each step

- Principle of deferred decisions: we don't choose the neighbors of each vertex until we color it ⇒ uncolored part of the graph is uniformly random in $G(n,p)$

- Let S_3 and S_2 denote the number of 3-color and 1-or-2-color vertices (we'll deal specifically with the 1-color vertices later) and let T denote the number of steps taken so far

- Each step: from the point of view of other vertices, a random vertex has been given a random color (symmetry between the colors needs to be proved)

- During the phase where the algorithm colors the giant component, each step colors a 1- or 2-color vertex

$$\mathbb{E}[\Delta S_3] = -pS_3$$
$$\mathbb{E}[\Delta S_2] = pS_3 - 1$$

A differential equation

- Write $S_3 = s_3\, n$, $S_2 = s_2\, n$, $T = tn$. Then with $p = c/n$, rescaling turns

$$\mathbb{E}[\Delta S_3] = -pS_3 \ , \quad \mathbb{E}[\Delta S_2] = pS_3 - 1$$

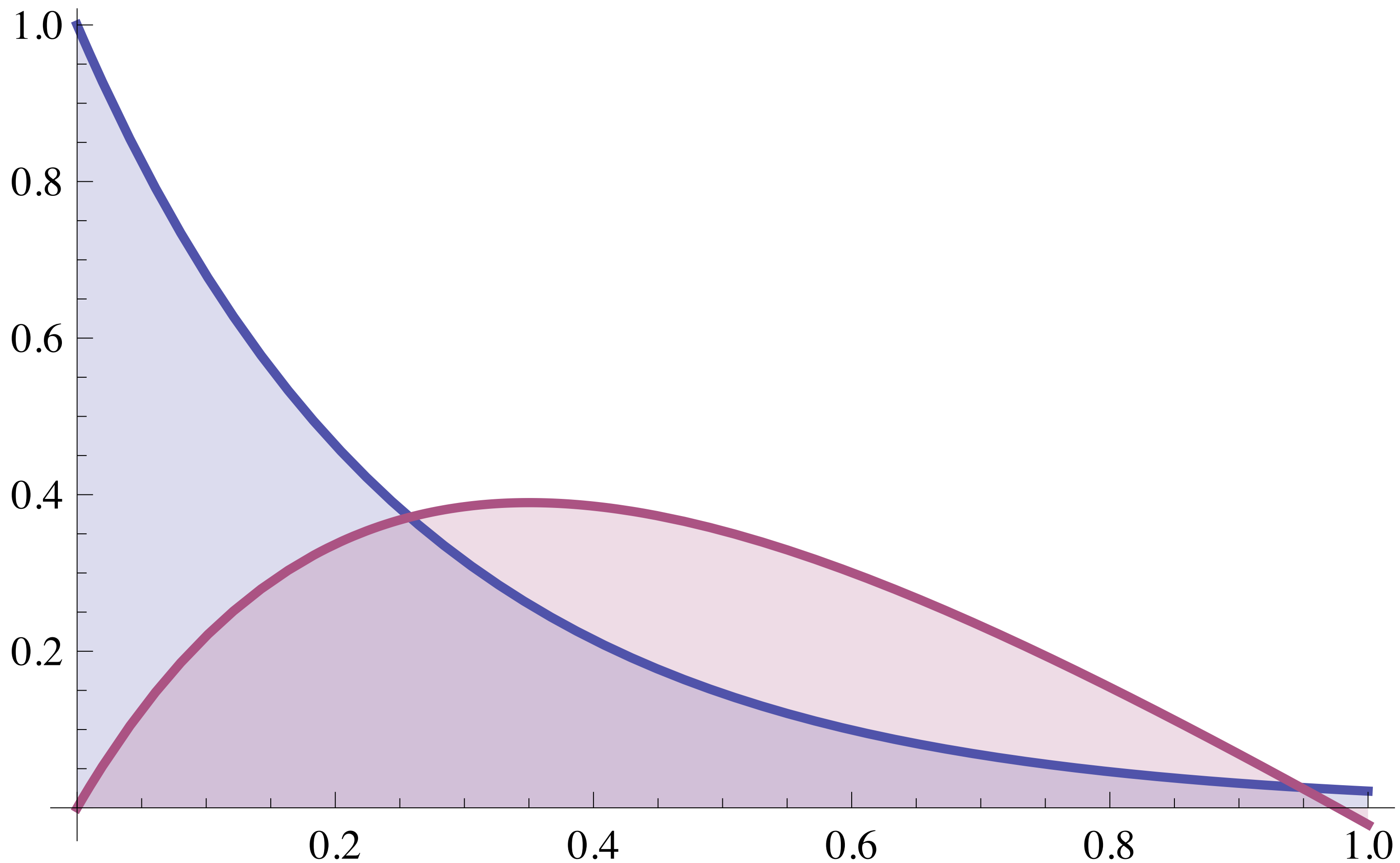
into

$$\frac{\mathrm{d}s_3}{\mathrm{d}t} = -cs_3 \ , \quad \frac{\mathrm{d}s_2}{\mathrm{d}t} = cs_3 - 1$$

- Easy to solve: with initial conditions $s_3(0) = 1, s_2(0) = 0$,

$$s_3(t) = \mathrm{e}^{-ct} \ , \quad s_2(t) = 1 - t - c\mathrm{e}^{-ct}$$

Progress over time



- the danger of conflict is greatest when $s_2(t)$ is maximized
- $s_2(t)=0$ when we have colored the giant component

When does the algorithm fail?

- The 1-color vertices obey a branching process. Coloring each one generates λ new ones on average, where

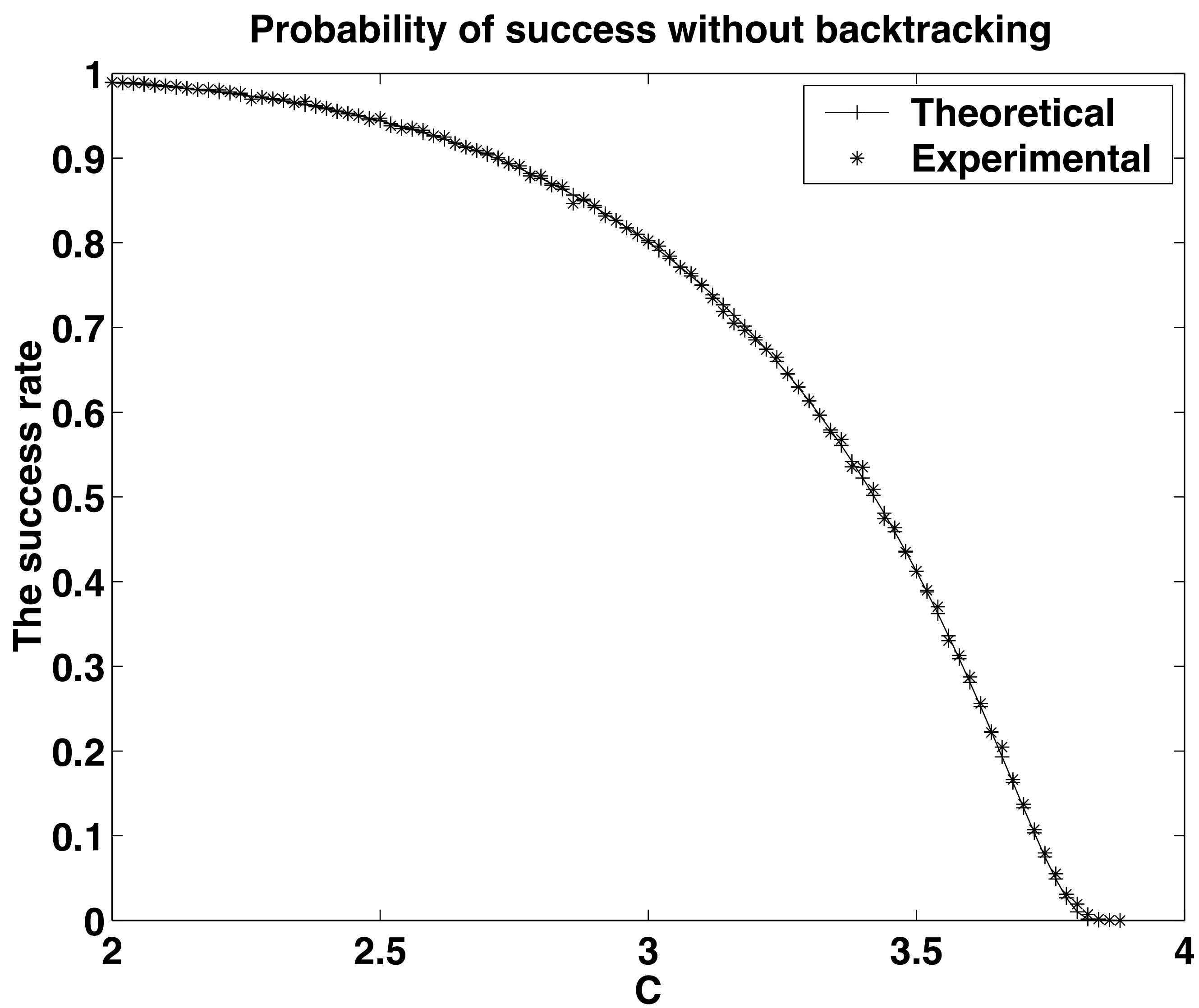
$$\lambda = (2/3)pS_2 = (2/3)cs_2$$

- As long as $\lambda(t) < 1$, this branching process is subcritical; integrating over all steps, we succeed with probability $\Omega(1)$
- If $\lambda > 1$, the process explodes, and two 1-color vertices conflict
- Maximizing $\lambda(t)$ over all t shows that $\lambda < 1$ as long as c is less than the root of

$$c - \ln c = 5/2$$

- This shows that $c^\star > 3.847$ [Achlioptas and Molloy]

Probability of success



$$P = \exp\left(-\int_0^{t_0} \mathrm{d}t\, \frac{c\lambda^2}{2(1-\lambda)(2+\lambda)}\right)$$

[Jia and Moore]

But does all this make sense?

- Theorem [Wormald]: if we have a Markov process *Y*(*t*) such that

$$\mathbb{E}[\Delta Y] = f(Y(T)/n) + o(1)$$

for a Lipschitz function *f*, and if *Y*(*t*)=Ω(*n*) for all *t*, and there are tail bounds on ΔY, then with high probability for all *T*,

$$Y(T) = ny(T/n) + o(n)$$

where *y* is the solution to the system of differential equations

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y)$$

- Idea: divide up the *n* steps into, say, $n^{1/2}$ blocks of $n^{1/2}$ steps each. In each block, total change in Y_i is within $n^{1/3}$ of its expectation (Azuma) with high enough probability that this is true of all blocks (union bound). Then the total change over Ω(*n*) steps is within o(*n*) of what the differential equation predicts.

Fancier algorithms, systems of equations

- A greedier algorithm: color high-degree vertices first
- Softer greed: choose vertices of degree *i* with probability proportional to *f*(*i*) for some smooth function *f*
- Infinite system of differential equations, keeping track of the degree distribution (random in the configuration model)
- This gives a lower bound of $c^\star > 4.03$ [Achlioptas and Moore]
- Still the best known lower bound
- We can only handle backtracking-free algorithms, where the rest of the problem is uniformly random (conditional on degree distributions etc.)

Linear programming

MAXIMIZATION OF A LINEAR FUNCTION OF VARIABLES SUBJECT TO LINEAR INEQUALITIES[1]

BY GEORGE B. DANTZIG

The general problem indicated in the title is easily transformed, by any one of several methods, to one which maximizes a linear form of non-negative variables subject to a system of linear equalities. For example, consider the linear inequality $ax + by + c > 0$. The linear inequality can be replaced by a linear equality in nonnegative variables

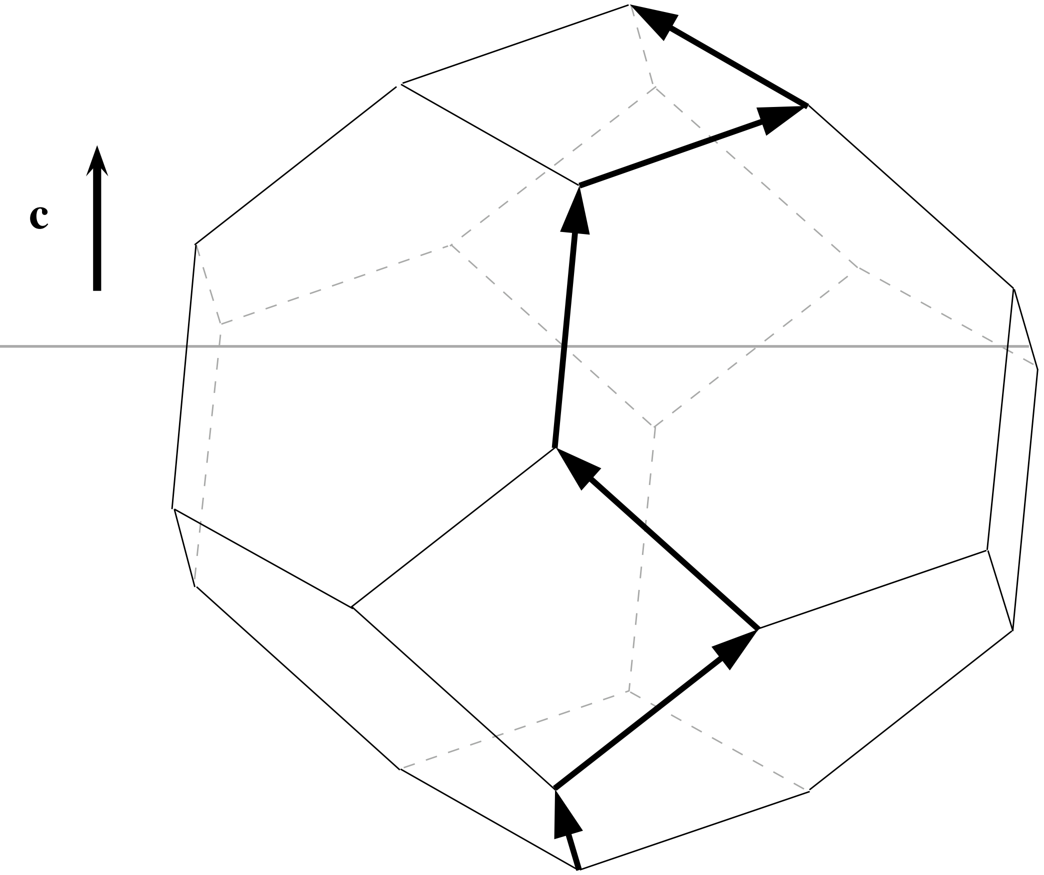
Linear programming

- Optimize a linear function, subject to linear inequalities:

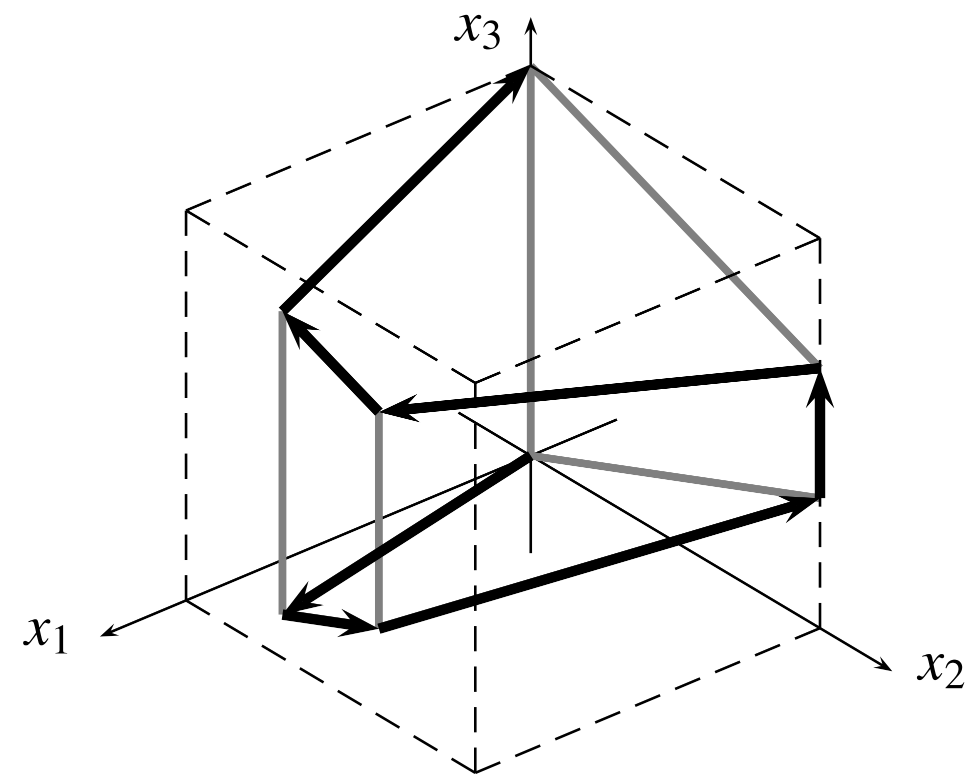
$$\max_{\mathbf{x}} \mathbf{c}^T \mathbf{x} \quad \text{subject to} \quad A\mathbf{x} \leq \mathbf{b}$$

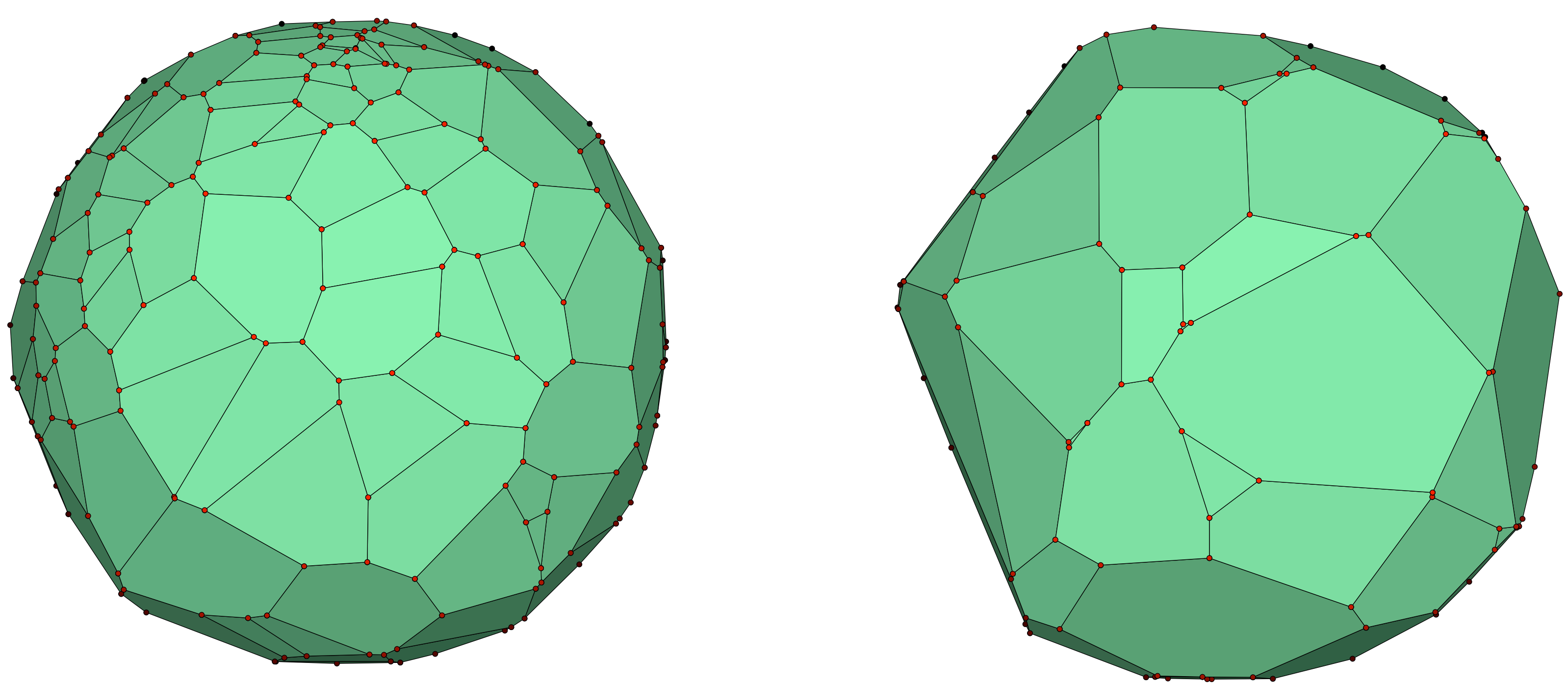
- Max Flow
- Shortest Path
- Min-(or Max)-Weight Perfect Matching
- “relax and round” approximation algorithms, e.g. Vertex Cover
- branch and bound / branch and cut optimization algorithms, including large instances of Traveling Salesman

Crawling on the surface



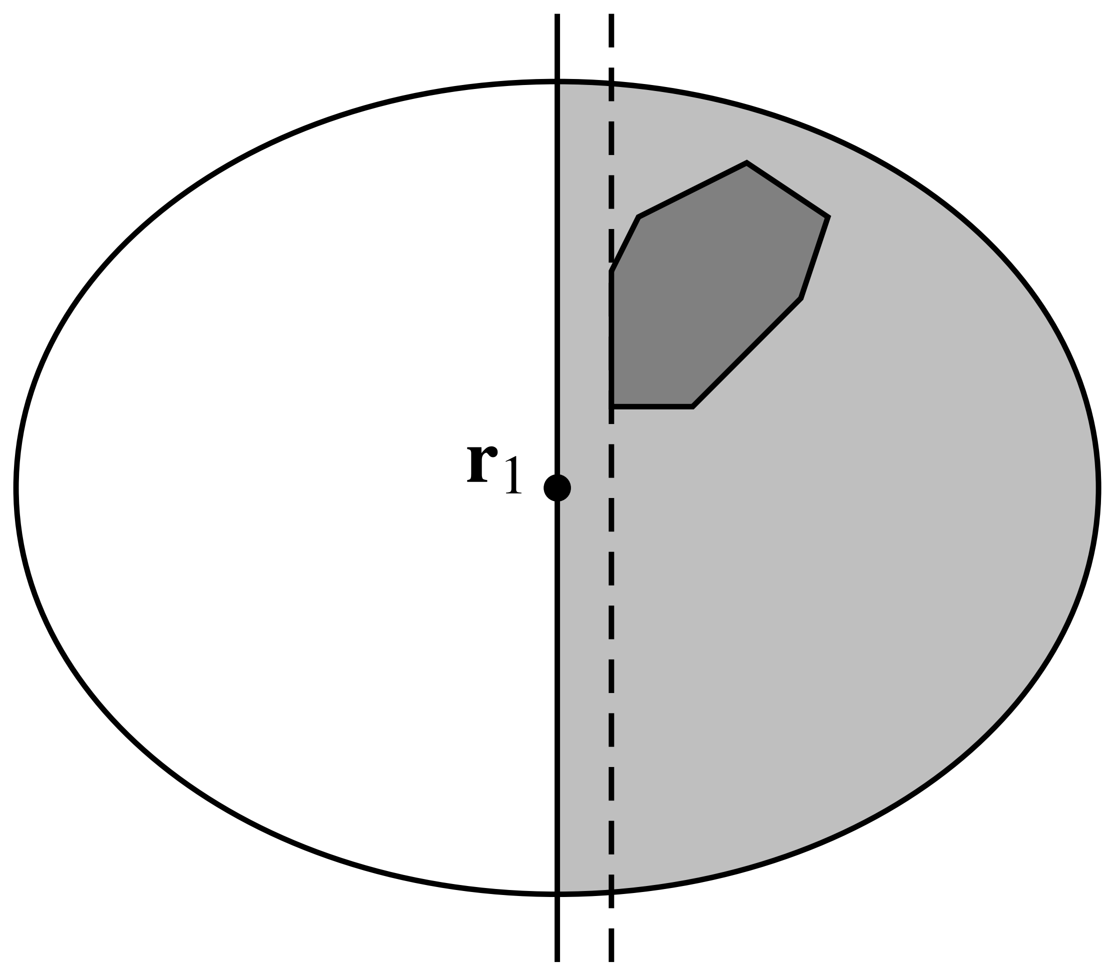
- Dantzig, 1947: the simplex algorithm
- Klee-Minty, Jeroslow, 1972: simplex takes exponential time in the worst case
- Spielman and Teng, 2004: the simplex algorithm works in polynomial time with high probability in the *smoothed analysis* setting—random perturbations of a worst-case instance





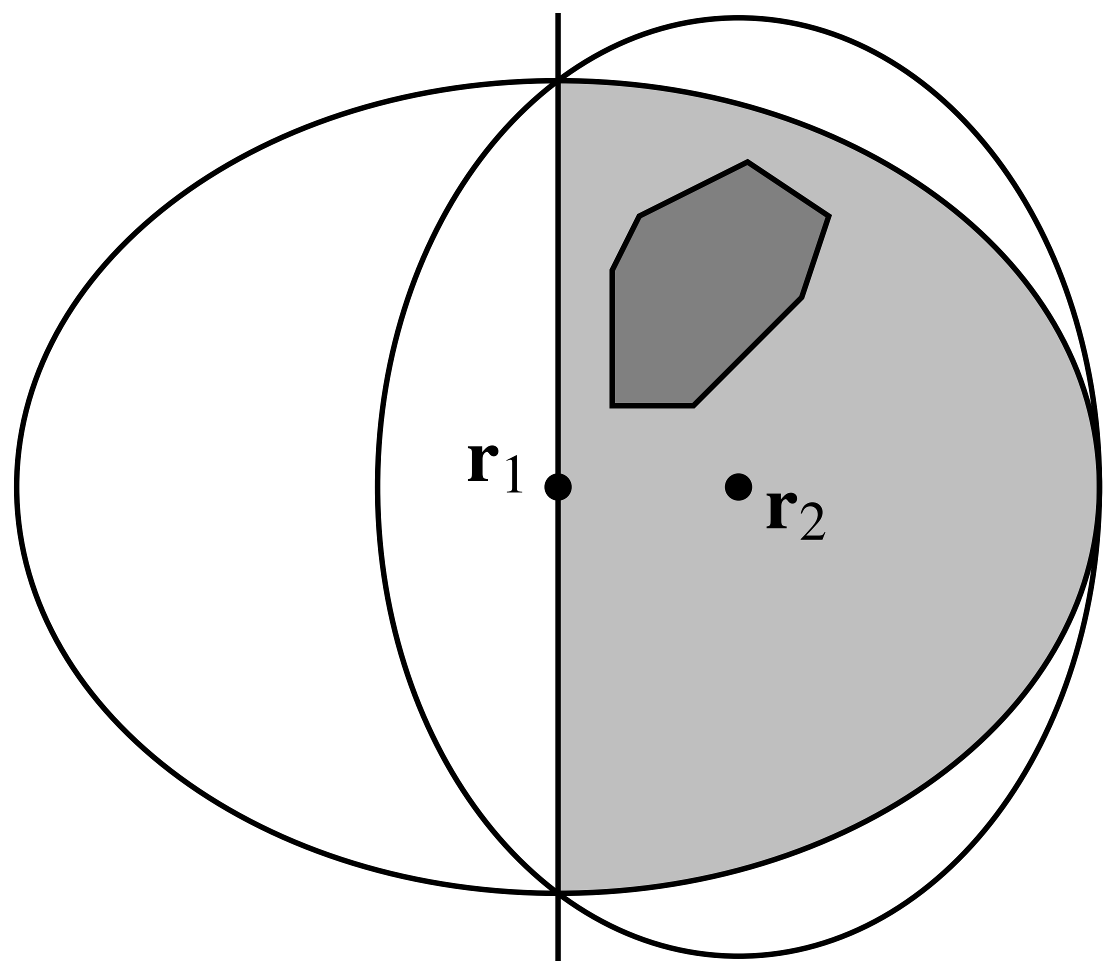
Ellipsoid algorithm

- Khachiyan, 1979: first proof that Linear Programming is in P



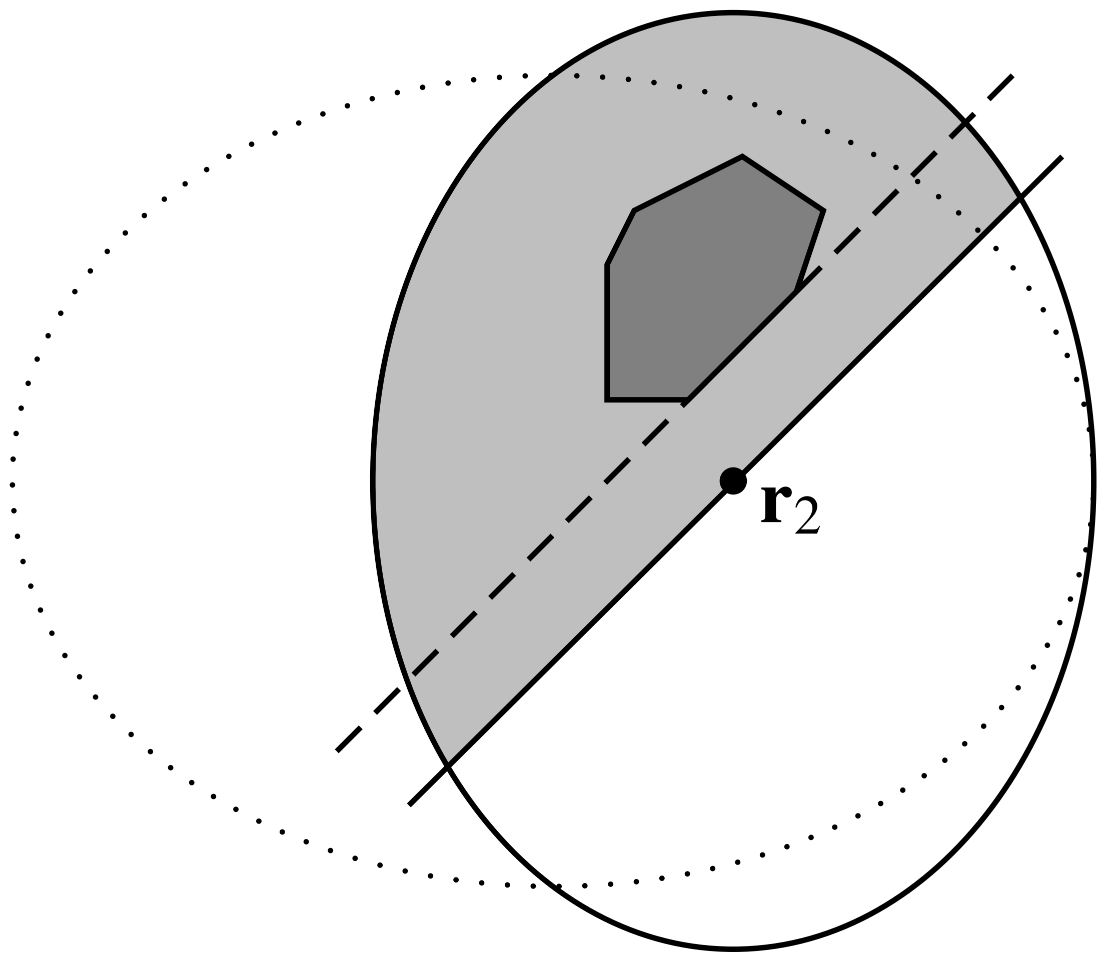
Ellipsoid algorithm

- Khachiyan, 1979: first proof that Linear Programming is in P



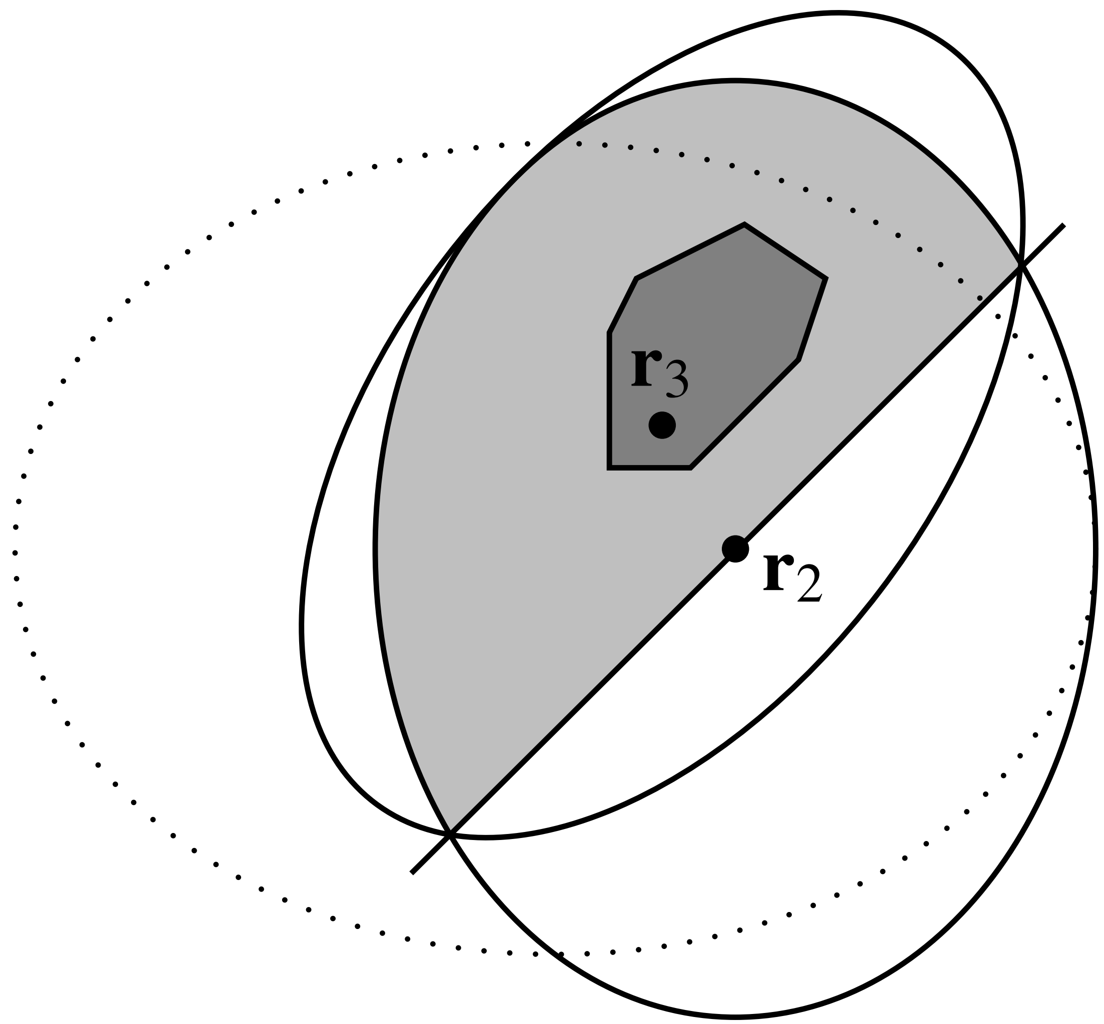
Ellipsoid algorithm

- Khachiyan, 1979: first proof that Linear Programming is in P



Ellipsoid algorithm

- Khachiyan, 1979: first proof that Linear Programming is in P



- any convex optimization problem with a polynomial-time separation oracle: either returns “r is feasible” or a separating hyperplane

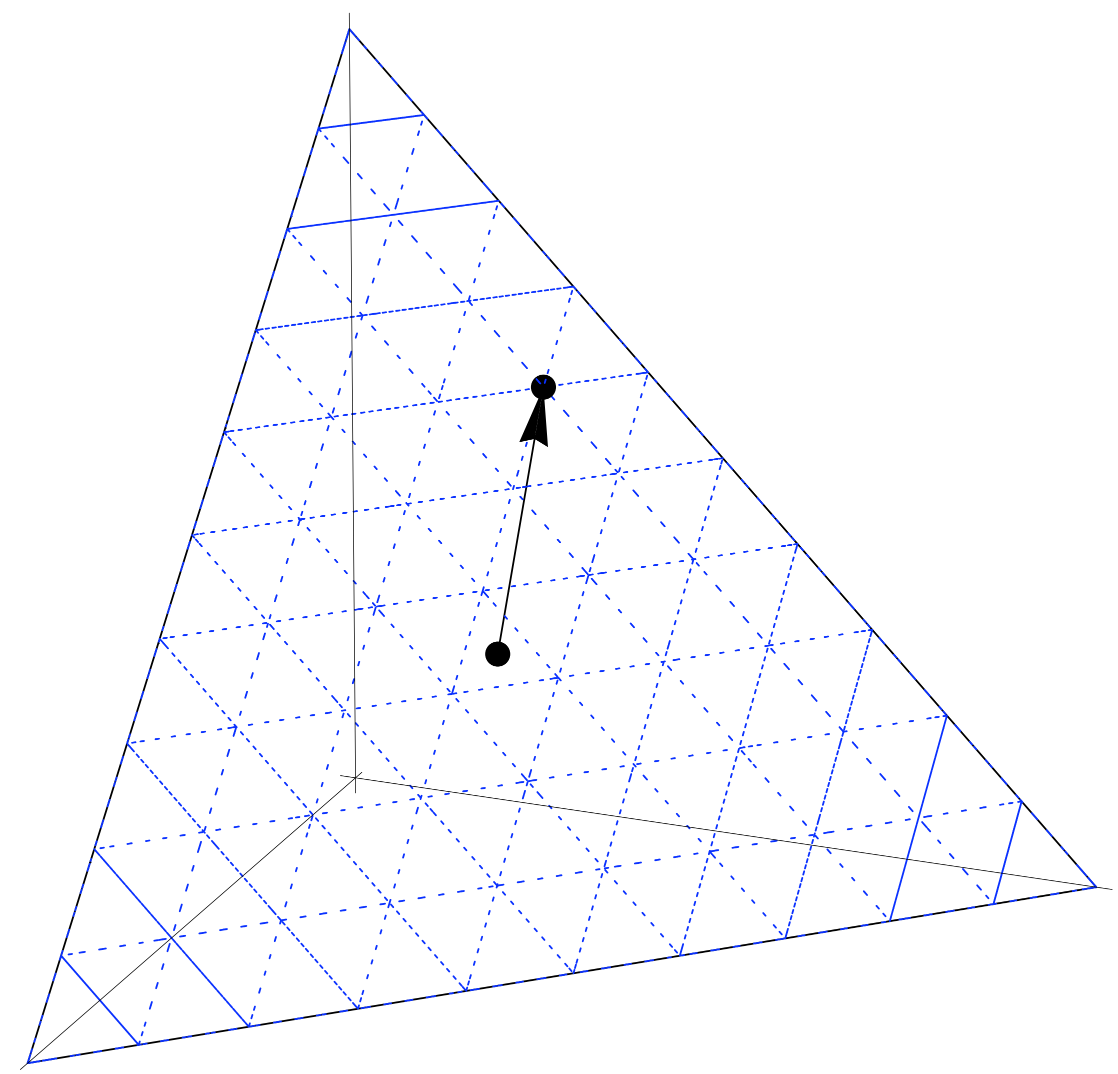
Karmarkar’s algorithm

- 1984: first algorithm for Linear Programming which is both theoretically and practically efficient

- First rigorously analyzed interior point method

Karmarkar's algorithm

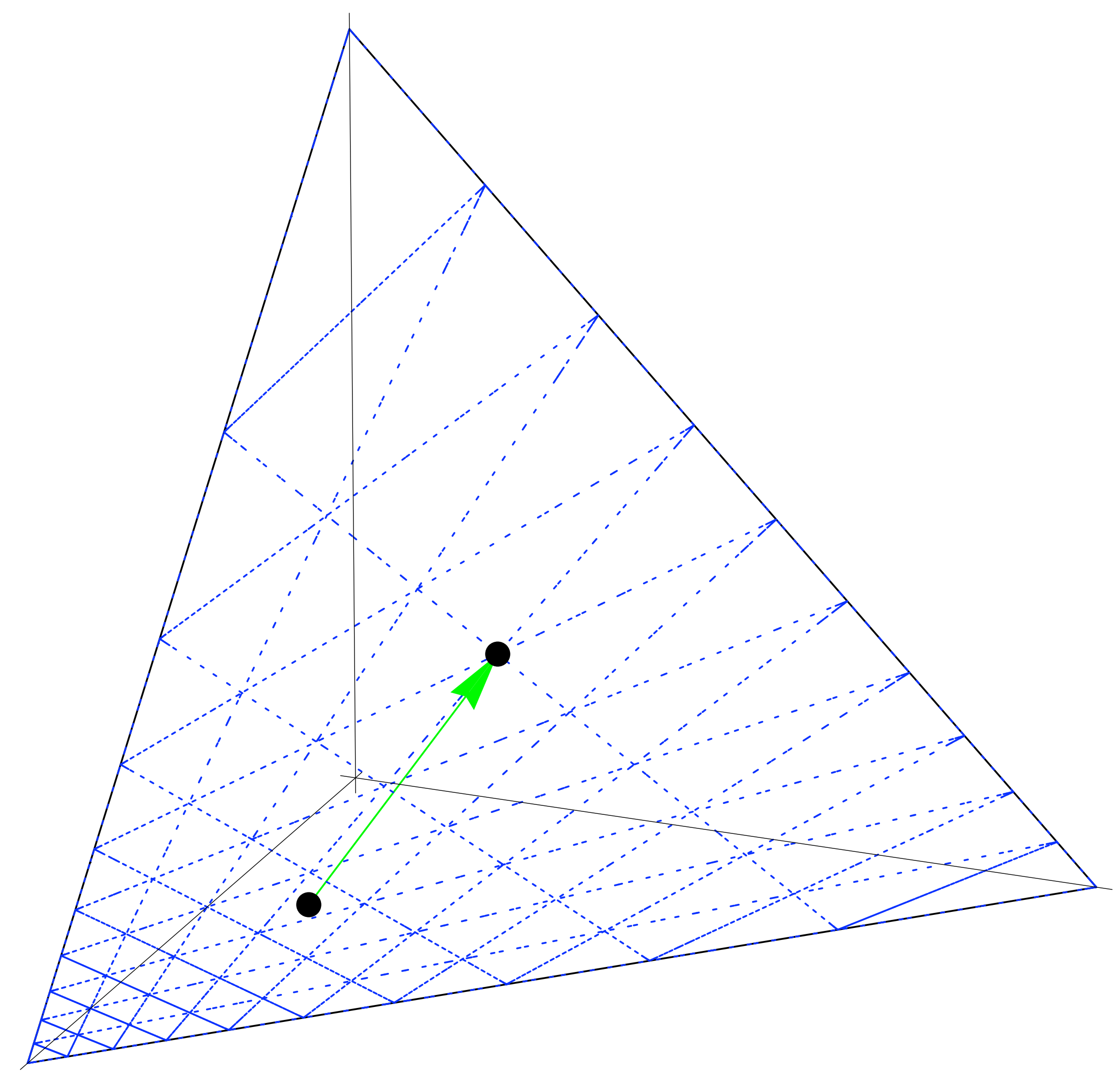
- Take a step in the direction of c



Karmarkar's algorithm

- Take a step in the direction of *c*

- Project back to the center

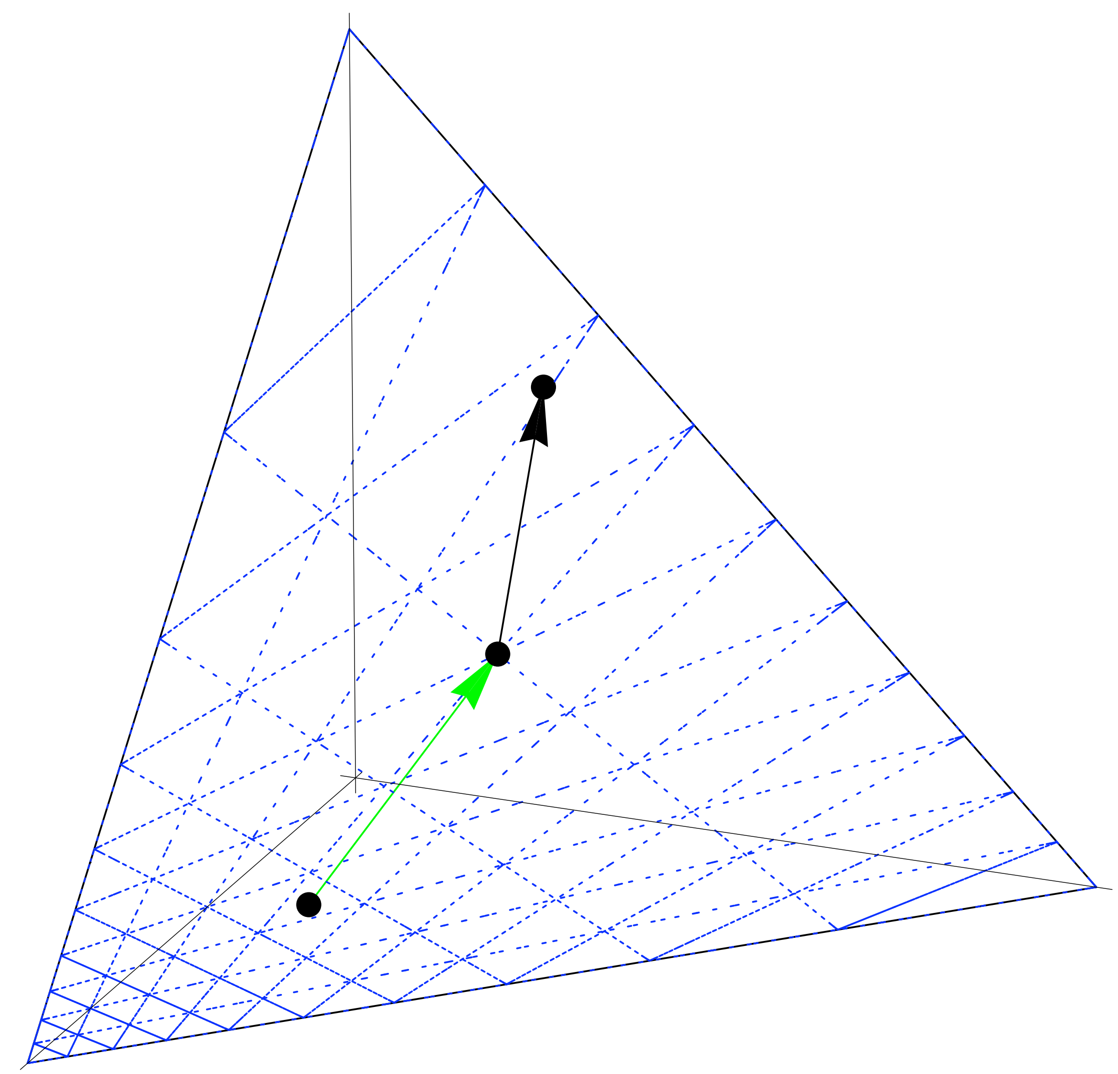


Karmarkar's algorithm

- Take a step in the direction of *c*

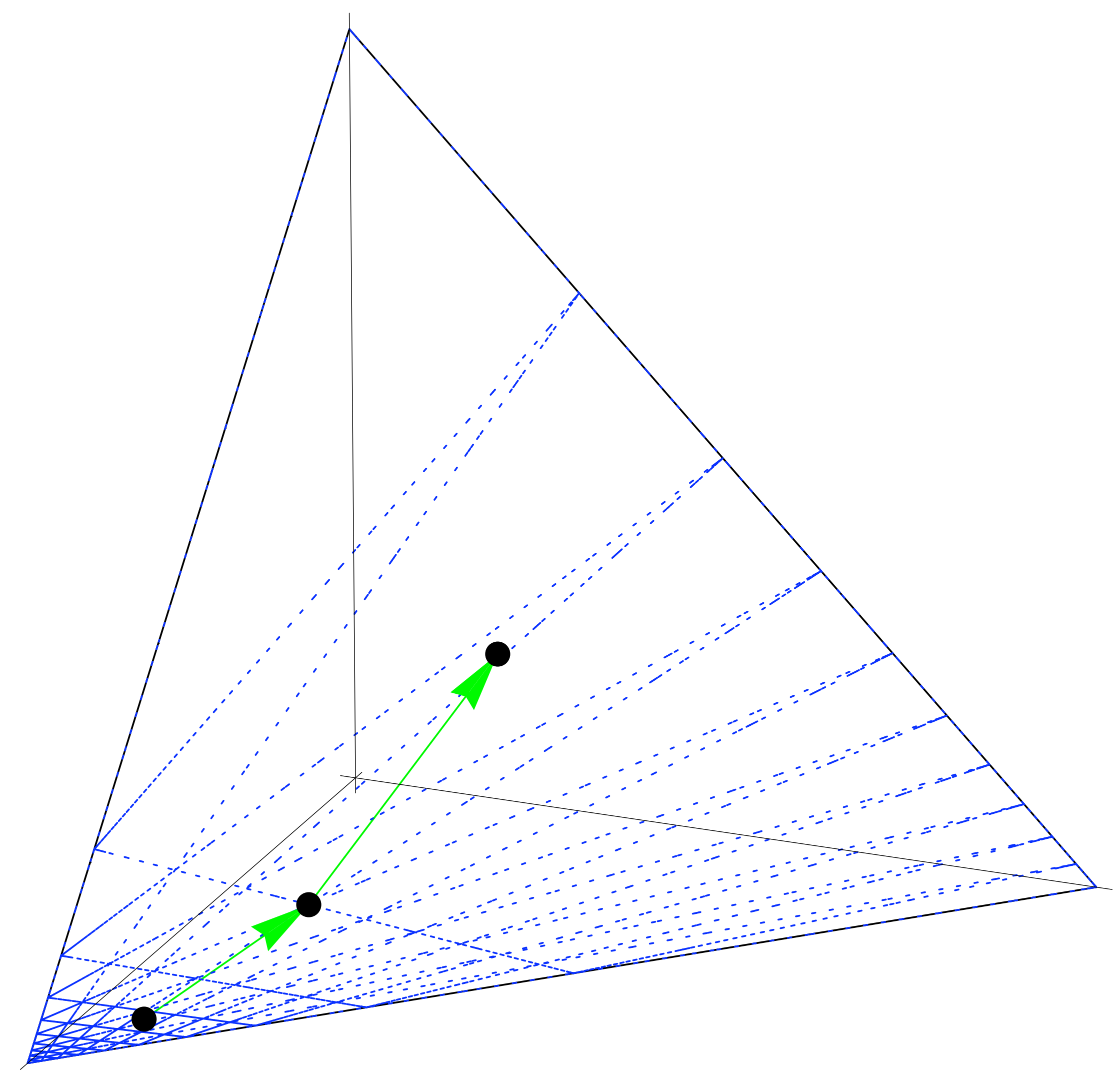
- Project back to the center

- Take another step



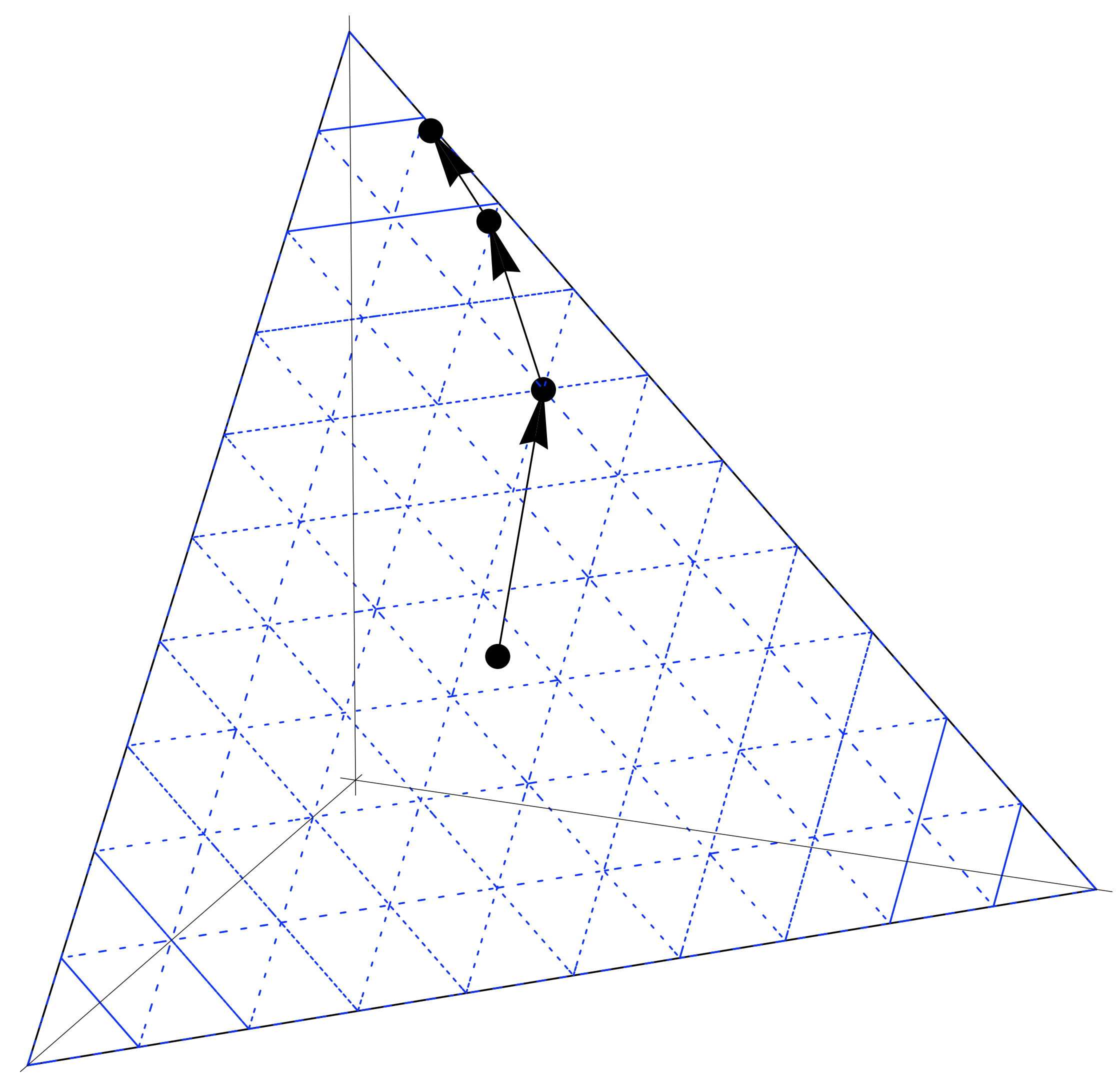
Karmarkar's algorithm

- Take a step in the direction of *c*
- Project back to the center
- Take another step
- Project again, and so on



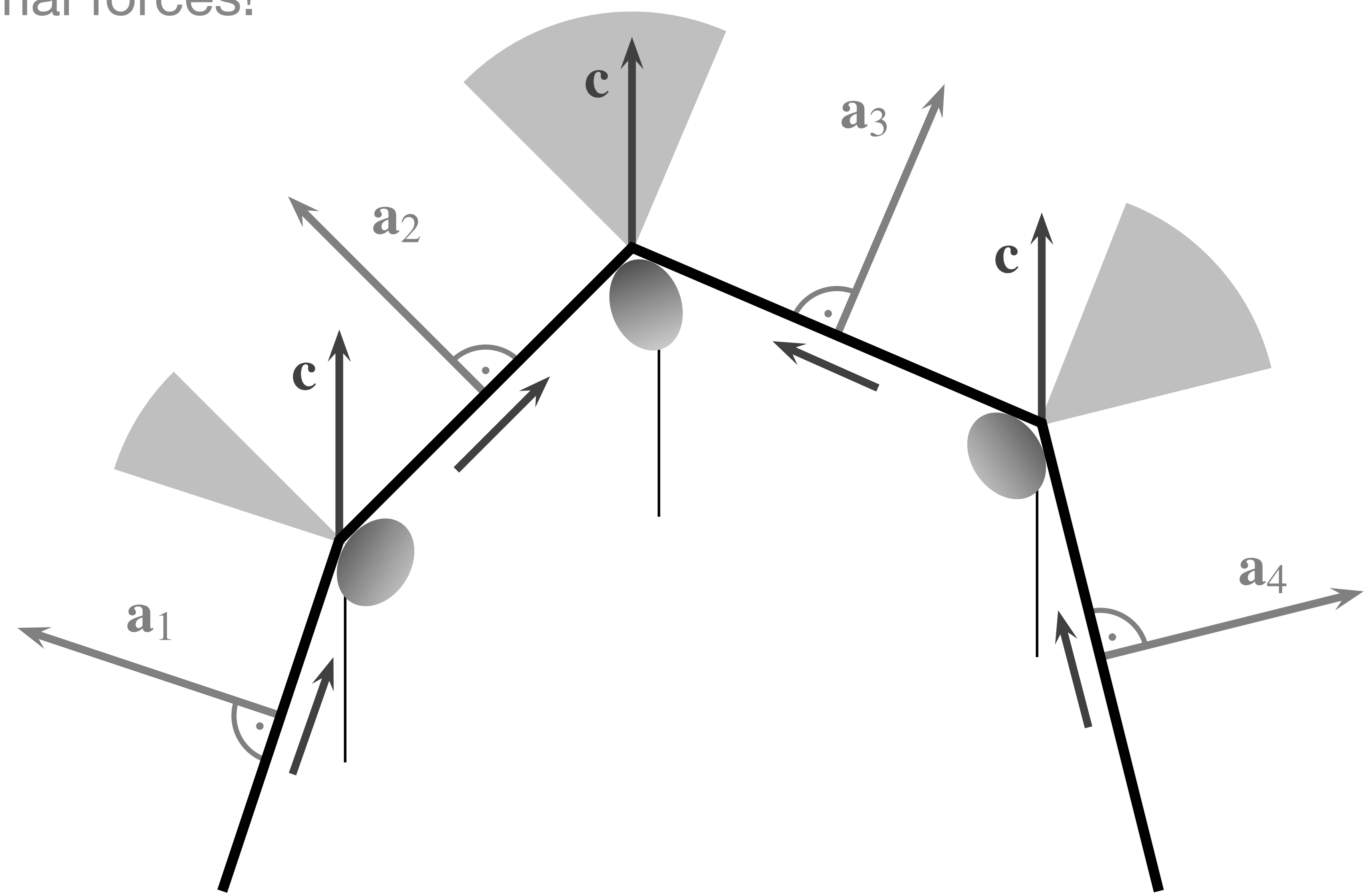
Karmarkar's algorithm

- Take a step in the direction of *c*
- Project back to the center
- Take another step
- Project again, and so on
- In the original coordinates,
  we converge to the optimum



A balloon rising in a cathedral

- The balloon's buoyancy causes it to rise until it meets the facets
- Dual variables = normal forces!



Potential energy

- A physical force is the gradient of the potential energy:

$$\mathbf{F} = -\nabla V(\mathbf{x}) \quad \text{or} \quad F_i = -\frac{\partial V}{\partial x_i}$$

- The balloon’s buoyancy is driven by

$$V(x) = -\mathbf{c}^T\mathbf{x}$$

- How do we keep it from bumping it into the facets, and getting stuck in a bad instance of the simplex algorithm?

The logarithmic barrier potential

- We want to repel the balloon from each facet,

$$\{\mathbf{x} : \mathbf{a}_j^T \mathbf{x} = b_j\}$$

- Add a potential energy that tends to +∞ as constraints become tight:

$$V_{\text{facets}}(\mathbf{x}) = -\sum_{j=1}^{m} \ln\left(b_j - \mathbf{a}_j^T \mathbf{x}\right)$$

- Force diverges as 1/r as we get close to a facet

- To get close to the optimum, make the balloon more buoyant:

$$V(\mathbf{x}) = V_{\text{facets}}(\mathbf{x}) - \lambda \mathbf{c}^T \mathbf{x}$$

The balloon at equilibrium

- The potential energy has a unique minimum $\mathbf{x}_\lambda$ in the interior, where

$$-\nabla V(\mathbf{x}_\lambda) = -\nabla V_{\text{facets}}(\mathbf{x}_\lambda) + \lambda \mathbf{c} = 0$$

- How does this equilibrium change as λ increases?

$$\mathbf{x}_{\lambda + \mathrm{d}\lambda} = \mathbf{x}_\lambda + \frac{\mathrm{d}\mathbf{x}_\lambda}{\mathrm{d}\lambda}\,\mathrm{d}\lambda$$

- Differentiating with respect to λ gives

$$\mathbf{H}\,\frac{\mathrm{d}\mathbf{x}_\lambda}{\mathrm{d}\lambda} = \mathbf{c} \quad \text{where} \quad H_{ij} = \frac{\partial^2 V_{\text{facets}}}{\partial x_i\,\partial x_j}$$

A differential equation

- The equilibrium $\mathbf{x}_\lambda$ obeys

$$\frac{\mathrm{d}\mathbf{x}_\lambda}{\mathrm{d}\lambda} = \mathbf{H}^{-1}\mathbf{c}$$

- Explicitly,

$$\nabla V_{\mathrm{facets}}(\mathbf{x}) = \sum_{j=1}^{m} \frac{\mathbf{a}_j^T}{b_j - \mathbf{a}_j^T \mathbf{x}}$$

$$\mathbf{H}(\mathbf{x}) = \sum_{j=1}^{m} \frac{\mathbf{a}_j \otimes \mathbf{a}_j^T}{(b_j - \mathbf{a}_j^T \mathbf{x})^2}$$

- In the nondegenerate case, $\mathbf{H}$ is positive definite, and hence invertible

Back to Karmarkar

- In the original coordinates, Karmarkar's algorithm updates x as

$$\mathbf{x} \to \mathbf{x} + \Delta\lambda\, \mathbf{H}^{-1}\mathbf{c}$$

...a discrete version of the differential equation [Gill et al.]

$$\frac{\mathrm{d}\mathbf{x}_\lambda}{\mathrm{d}\lambda} = \mathbf{H}^{-1}\mathbf{c}$$

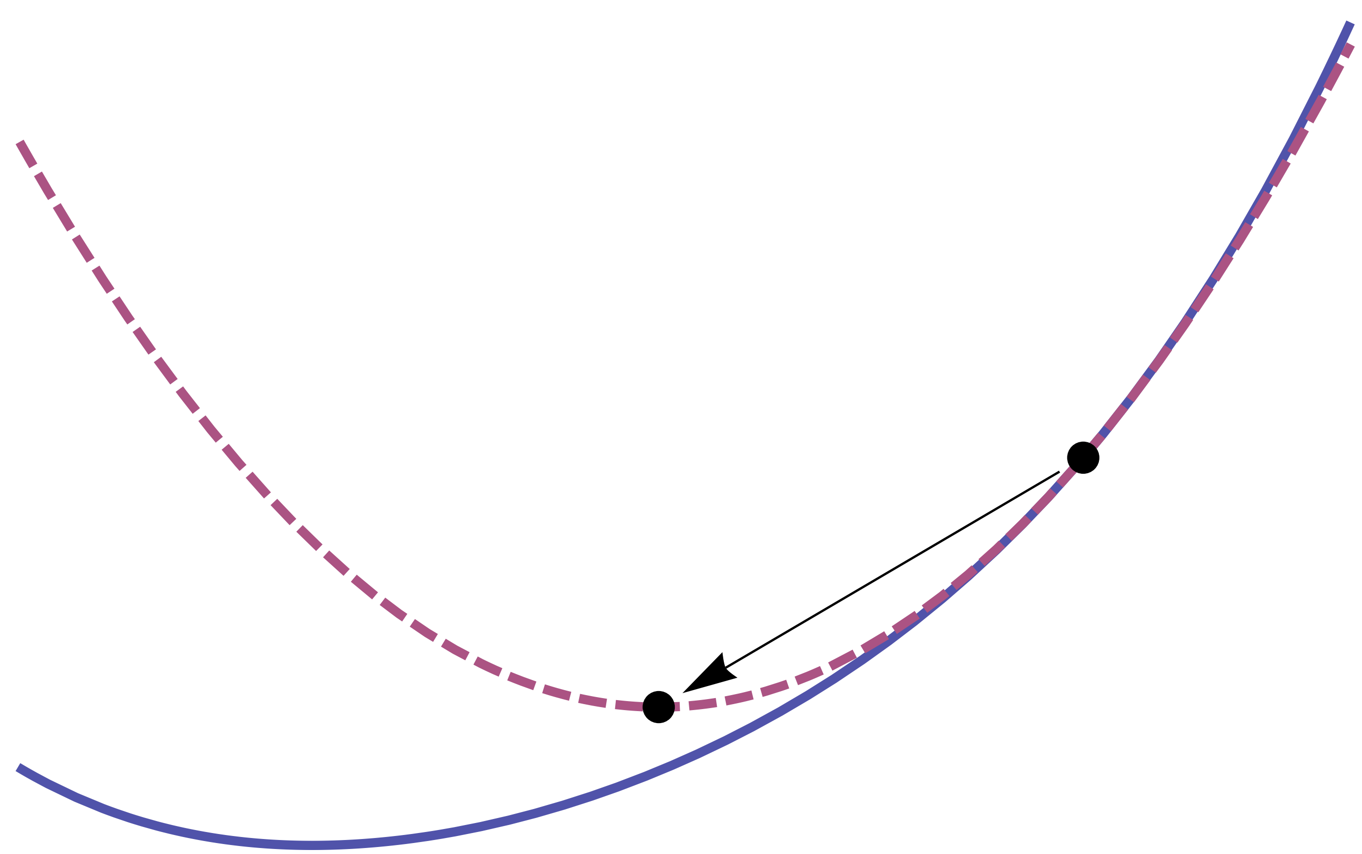
- The repulsive force~1/r, so we need λ~1/ϵ to get within ϵ of the optimum

- We get within $\epsilon=2^{-n}$ of the optimum when $\lambda\sim 2^n$

- Each step is large enough to multiply λ by a constant, so poly(n) steps

Newton's method

- Use a second-order Taylor series to predict the minimum



Newton's method

- In one dimension, minimizing

$$V(x+\delta) \approx V(x) + V'\delta + \frac{1}{2}V''\delta^2$$

gives

$$\delta = -\frac{1}{V''}V'$$

- In higher dimensions, minimizing

$$V(\mathbf{x}+\delta) \approx V(\mathbf{x}) + (\nabla V)^T\delta + \frac{1}{2}\,\delta^T\mathbf{H}\,\delta$$

gives

$$\delta = -\mathbf{H}^{-1}\,\nabla V$$

The Newton flow

- Every point in the polytope is the equilibrium for some buoyancy,

$$\lambda \mathbf{c} = \nabla V_{\text{facets}}$$

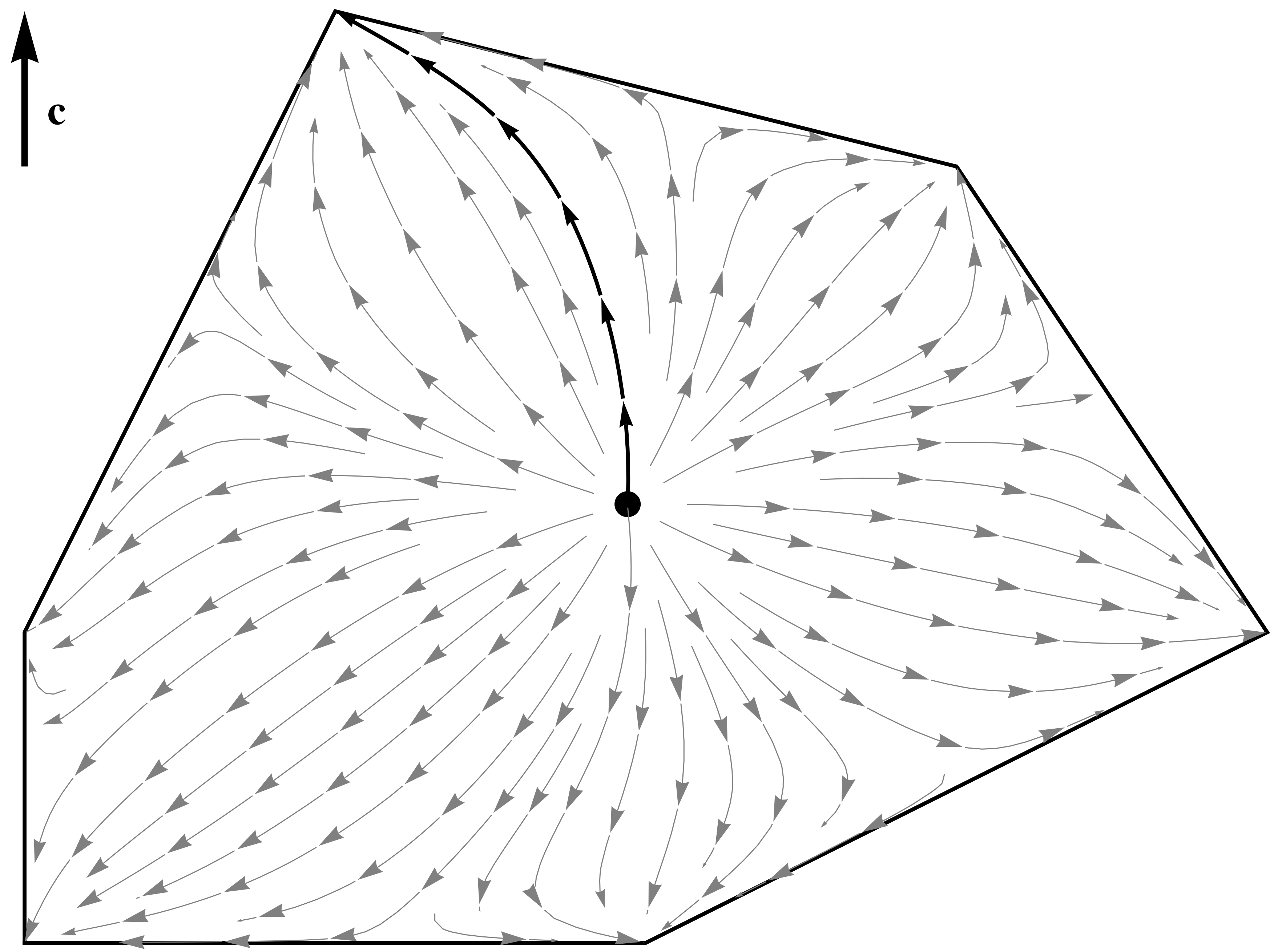
- Moving away from the minimum of ∇V_{facets} gives

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -\delta = \mathbf{H}^{-1}\nabla V_{\text{facets}} = \lambda \mathbf{H}^{-1}\mathbf{c}$$

  which is proportional to our previous differential equation.

- Each trajectory of this *reverse Newton flow* maximizes $c^T x$ for some c.
  All that matters are the initial conditions! [Anstreicher]

The Newton flow



Semidefinite programming

- Maximize a linear function of the entries of a matrix A, subject to

$$A \succeq 0 \quad \text{i.e.,} \quad \mathbf{v}^T A \mathbf{v} \geq 0$$

- The boundary is the set of singular matrices:

$$\{A : \det A = 0\}$$

- Add a barrier potential which tends to $+\infty$ as A becomes singular:

$$V = -\ln \det A$$

- This gives a repulsive force

$$F = -\nabla V = (A^{-1})^T$$

Conclusion

- Differential equations help us analyze the performance of algorithms on random structures, and thus prove things about those structures
- Some of our favorite algorithms are best viewed as discrete versions of continuous algorithms
- Computer science needs every kind of mathematics:
  - Number theory (cryptography)
  - Fourier analysis (the PCP theorem)
  - Group representations (expanders and derandomization)
  - and anything else we can get our hands on

Shameless plug

- Oxford University Press, 2010

THE NATURE
of COMPUTATION

Cristopher Moore
Stephan Mertens

Acknowledgments