15-853:Algorithms in the Real World

Linear and Integer Programming II
- Ellipsoid algorithm
- Interior point methods

Ellipsoid Algorithm

First polynomial-time algorithm for linear programming (Khachiyan 79)

Solves

find x
subject to $Ax . b$

i.e find a feasible solution

Run Time:

$O(n^4L)$, where L = #bits to represent A and b

Problem in practice: always takes this much time.

Reduction from general case

To solve:

maximize: $z = c^Tx$
subject to: $Ax . b, \quad x = 0$

Convert to:

find: x, y
subject to: $Ax . b,$

$-x . 0$
$-yA . -c$
$-y . 0$
$-cx + by . 0$

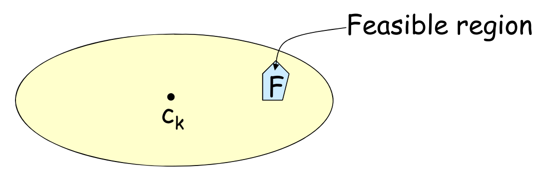
Ellipsoid Algorithm

Consider a sequence of smaller and smaller ellipsoids each with the feasible region inside.

For iteration k:

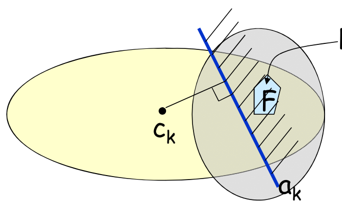
c_k = center of E_k

Eventually c_k has to be inside of F, and we are done.



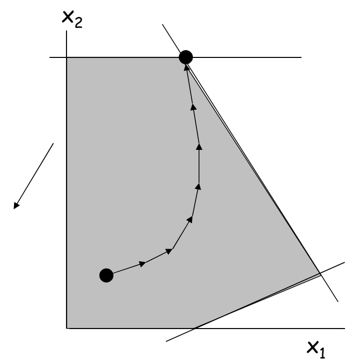
Ellipsoid Algorithm

To find the next smaller ellipsoid:
- find most violated constraint a_k
- find smallest ellipsoid that includes constraint



$$\frac{Vol(E_{k+1})}{Vol(E_k)} = \frac{1}{2^{1/(2n+1)}}$$

Interior Point Methods



Travel through the interior with a **combination** of
1. An **optimization** term (moves toward objective)
2. A **centering** term (keeps away from boundary)

Used since 50s for nonlinear programming.

Karmakar proved a variant is **polynomial time** in 1984

Methods

Affine scaling: simplest, but no known time bounds
Potential reduction: O(nL) iterations
Central trajectory: O($n^{1/2}$ L) iterations

The time for each iteration involves solving a linear system so it takes polynomial time. The "real world" time depends heavily on the matrix structure.

Example times

	fuel	continent	car	initial
size (K)	13x31K	9x57K	43x107K	19x12K
non-zeros	186K	189K	183K	80K
iterations	66	64	53	58
time (sec)	2364	771	645	9252
Cholesky non-zeros	1.2M	.3M	.2M	6.7M

Central trajectory method (Lustic, Marsten, Shanno 94)
Time depends on Cholesky non-zeros (i.e. the "fill")

Assumptions

We are trying to solve the problem:

minimize $z = c^T x$

subject to $Ax = b$

$x = 0$

Outline

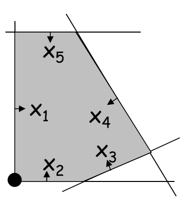
1. Centering Methods Overview
2. Picking a direction to move toward the optimal
3. Staying on the Ax = b hyperplane (projection)
4. General method
5. Example: Affine scaling
6. Example: potential reduction
7. Example: log barrier

Centering: option 1

The "analytical center":

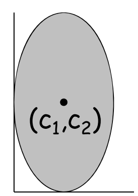
Minimize: $y = -\Sigma_{i=1}^{n} \lg x_i$

y goes to @ as x approaches any boundary.



Centering: option 2

Elliptical Scaling:



$$\frac{x_1^2}{c_1^2} + \frac{x_2^2}{c_2^2} = 1$$

Dikin Ellipsoid

The idea is to bias spaced based on the ellipsoid.
More on this later.

Finding the Optimal solution

Let's say f(x) is the combination of the "centering term" c(x) and the "optimization term" $z(x) = c^T x$.

We would like this to have the same minimum over the feasible region as z(x) but can otherwise be quite different.

In particular c(x) and hence f(x) need not be linear.

Goal: find the minimum of f(x) over the feasible region starting at some interior point x_0

Can do this by taking a sequence of steps toward the minimum.

How do we pick a direction for a step?

Picking a direction: steepest descent

Option 1: Find the steepest descent on x at x_0 by taking the gradient: $\nabla f(x_0)$

Problem: the gradient might be changing rapidly, so local steepest descent might not give us a good direction.

Any ideas for better selection of a direction?

Picking a direction: Newton's method

Consider the truncated taylor series:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

To find the **minimum** of f(x) take the derivative and set to 0.

$$0 = f'(x_0) + f''(x_0)(x - x_0)$$

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

In matrix form, for arbitrary dimension:

$$x = x_0 - (F(x))^{-1} \nabla f(x)^T \qquad F(x) = \nabla \times \nabla f(x)$$

Hessian

Next Step?

Now that we have a direction, what do we do?

Remaining on the support plane

Constraint: $Ax = b$

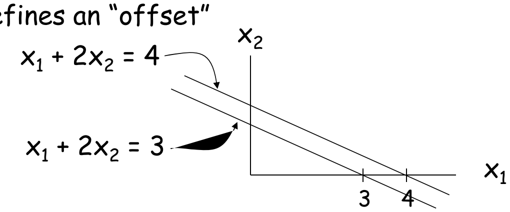
A is a $n \times (n + m)$ matrix.

The equation describes an m dimensional hyperplane in a n+m dimensional space.

The hyperplane basis is the null space of A

A = defines the "slope"

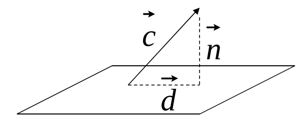
b = defines an "offset"



Projection

Need to project our direction onto the plane defined by the null space of A.

$$\begin{aligned}\vec{d} &= \vec{c} - \vec{n} \\ &= c - A^T(AA^T)^{-1}Ac \\ &= (I - A^T(AA^T)^{-1}A)c \\ &= Pc\end{aligned}$$



$$P = \left(I - A^T\left(AA^T\right)^{-1}A\right) = \text{the "projection matrix"}$$

We want to calculate Pc

Calculating Pc

$\boxed{Pc} = (I - A^T(AA^T)^{-1}A)c = \boxed{c - A^Tw}$

where $A^Tw = A^T(AA^T)^{-1}Ac$

giving $AA^Tw = AA^T(AA^T)^{-1}Ac = Ac$

so all we need to do is solve for w in: $\boxed{AA^Tw = Ac}$

This can be solved with a sparse solver as described in the graph separator lectures.

This is the workhorse of the interior-point methods.

Note that AA^T will be more dense than A.

Next step?

We now have a direction c and its projection d onto the constraint plane defined by $Ax = b$.

What do we do now?

To decide how far to go we can find the minimum of f(x) along the line defined by d. Not too hard if f(x) is reasonably nice (e.g. has one minimum along the line).

Alternatively we can go some fraction of the way to the boundary (e.g. 90%)

General Interior Point Method

Pick start x_0
Factor AA^T
Repeat until done (within some threshold)

- decide on function to optimize f(x) (might be the same for all iterations)
- select direction d based on f(x) (e.g. with Newton's method)
- project d onto null space of A (using factored AA^T and solving a linear system)
- decide how far to go along that direction

Caveat: every method is slightly different

Affine Scaling Method

A biased steepest descent.
On each iteration solve:

minimize $c^T y$
subject to $Ay = 0$
$yD^{-2}y$. 1

Dikin ellipsoid

$$D = \begin{bmatrix} x_1 & 0 & 0 & \\ 0 & x_2 & 0 & \\ 0 & 0 & x_3 & \\ & & & \ddots \end{bmatrix}$$

Note that:

1. y is in the null space of A and can therefore be used as the direction d.
2. we are optimizing in the desired direction c^T

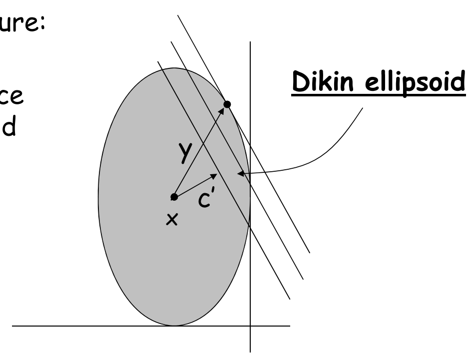
What does the Dikin Ellipsoid do for us?

Affine Scaling

Intuition by picture:

Ax = b is a slice of the ellipsoid

c' = Pc



Note that y is biased away from the boundary

How to compute

By substitution of variables: $y = Dy'$

minimize: $c^T Dy'$
subject to: $ADy' = 0$
$y'D^T D^{-2} Dy'$. 1 ($y'y'$. 1)

The sphere $y'y'$. 1 is unbiased.
So we project the direction $c^T D = Dc$ onto the nullspace of B = AD:

$y' = (I - B^T(BB^T)^{-1}B)Dc$

and

$y = Dy' = D(I - B^T(BB^T)^{-1}B)Dc$

As before, solve for w in $BB^T w = BDc$ and

$y = D(Dc - B^T w) = D^2(c - A^T w)$

Affine Interior Point Method

Pick start x_0

Symbolically factor AA^T

Repeat until done (within some threshold)

- $B = A D_i$
- Solve $BB^T w = BDc$ for w (use symbolically factored AA^T…same non-zero structure)
- $d = D_i(D_i c - B^T w)$
- move in direction d a fraction α of the way to the boundary (something like $\alpha = .96$ is used in practice)

Note that D_i changes on each iteration since it depends on x_i

Potential Reduction Method

minimize: $z = q \ln(c^T x - by) - \Sigma_{j=1}^{n} \ln(x_j)$

subject to: $Ax = b$

$x = 0$

$yA + s = 0$ (dual problem)

$s = 0$

First term of z is the **optimization** term

The second term of z is the **centering** term.

The objective function is **not** linear. Use hill climbing or "Newton Step" to optimize.

$(c^T x - by)$ goes to 0 near the solution

Central Trajectory (log barrier)

Dates back to 50s for nonlinear problems.

On step i:

1. **minimize**: $cx - \mu_k \Sigma_{j=1}^{n} \ln(x_j)$, s.t. $Ax = b$, $x > 0$
2. **select**: $\mu_{k+1} \cdot \mu_k$

Each minimization can be done with a constrained Newton step.

μ_k needs to approach zero to terminate.

A primal-dual version using higher order approximations is currently the best interior-point method in practice.

Summary of Algorithms

1. Actual algorithms used in practice are very sophisticated
2. Practice matches theory reasonably well
3. Interior-point methods dominate when
   A. Large n
   B. Small Cholesky factors (i.e. low fill)
   C. Highly degenerate
4. Simplex dominates when starting from a previous solution very close to the final solution
5. Ellipsoid algorithm not currently practical
6. Large problems can take hours or days to solve. Parallelism is very important.