 Notes by Steven Czerwinski

- Duality
- Ellipsoid Algorithm
- Interior Point Methods
  1. Affine scaling
  2. Potential reduction
  3. Central trajectory

1 Duality

Duality refers to the property that linear programming problems can be stated both as minimization and maximization problems. For any problem, there is a *primal* and a *dual*. The primal usually refers to the most natural way we describe the original problem. This can either be a minimization or maximization problem, depending on the specifics. The dual represents an alternative way to specify the original problem, such that it is a minimization problem if the primal is a maximization and vice versa. Solving the dual is equivalent to solving the original problem.

For example, suppose we have the following:

Primal problem: Maximize $z = cx$
such that $Ax \leq b$ and $x \geq 0$
(n equations, m variables)

Then, for our dual, we have: (*notice the change in m and n*)

Dual problem: Minimize $z = yb$
such that $yA \geq c$ and $y \geq 0$
(m equations, n variables)

So why do we care about the dual for a problem? One reason is that we often find that the dual is easier to solve. The dual can also be used for *sensitivity analysis* which determines how much the solution would change with changes in c, A or b. Finally, it can be used in conjunction with the primal problem to help solve some of the interior point methods.

1.1 Duality: General Case

In converting maximization problems to minimization problems and vice versa, we find that every variable in the primal produces a constraint in the dual. Likewise, every constraint in the primal produces a variable in the dual.

Maximization Problem		**Minimization Problem**
Constraints		Variables
$\leq$	$\Leftrightarrow$	≥ 0
$\geq$	$\Leftrightarrow$	≤ 0
$=$	$\Leftrightarrow$	unrestricted
Variables		Constraints
≥ 0	$\Leftrightarrow$	$\geq$
≤ 0	$\Leftrightarrow$	$\leq$
unrestricted	$\Leftrightarrow$	$=$

Table 11: Rules for converting between Primals and Duals

We also find there is correspondence between the variable's restrictions in the primal, with the constraint's equality sign in the dual. For example, if a variable must be greater than 0 in a minimization problem, then the constraint the variable produces in the dual will be stated as a "less than" relationship. Similarly, there is a correspondence between the constraints' equality signs in the primal to the restrictions on the variables in the dual. Table 11 summaries these correspondences.

1.2 Example

We are given the following maximization problem:

$$\text{maximize:} \quad 4x_1 + 2x_2 - x_3$$

$$\begin{aligned} \text{s.t.} \quad & x_1 + x_2 + x_3 = 20 \\ & 2x_1 - x_2 \geq 6 \\ & 3x_1 + 2x_2 + x_3 \leq 40 \\ & x_1, x_2 \geq 0 \end{aligned}$$

From this description, we easily determine the following:

$$\begin{aligned} x &= \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ c &= \begin{bmatrix} 4 & 2 & -1 \end{bmatrix} \\ b &= \begin{bmatrix} 20 \\ 6 \\ 40 \end{bmatrix} \\ A &= \begin{bmatrix} 1 & 1 & 1 \\ 2 & -1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \end{aligned}$$

We can now produce the dual:

$$
\begin{aligned}
\text{minimize:}\quad & 20Y_1 + 6Y_2 - 40Y_3 \\
\text{s.t.}\quad & Y_1 + 2Y_2 + 3Y_3 \geq 4 \\
& Y_1 - Y_2 + 2Y_3 \geq 2 \\
& Y_1 + Y_3 = -1 \\
& Y_2 \leq 0, Y_3 \geq 0
\end{aligned}
$$

The coefficients come from transposing A. To determine the equality/inequality type we use the rules in Table 11. For example the first inequality $Y_1 + 2Y_2 + 3Y_3 \geq 4$ corresponds to variable x_1 in the primal. Since $x_1 \geq 0$ in the primal, the inequality type is also $\geq$ in the dual.

Notice, since the first constraint is produce by multiplying A's first column vector with Y, it corresponds to the x_1 variable in the primal (because the first column in A contains all of x_1's coefficients). Therefore, the first constraint in the dual must have an "equal" relationship since x_1 is unrestricted in the primal.

1.3 Duality Theorem

The Duality theorem states:

If $\overline{x}$ is feasible for **P** (given below) and $\overline{y}$ is feasible for **D** then: $c\overline{x} \leq \overline{y}b$ and at optimality, $c\overline{x} = \overline{y}b$. We call $\overline{y}b - c\overline{x}$ the *Duality Gap*. The magnitude of the duality gap is a good way to determine how close we are to an optimal solution and is used in some of the interior point methods.

$$
\begin{aligned}
\mathbf{P}\text{: Max}\quad & z = cx \\
\text{s.t.}\quad & Ax \leq b \\
& x \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{D}\text{: Min}\quad & Z = yb \\
\text{s.t.}\quad & yA \geq c \\
& y \geq 0
\end{aligned}
$$

2 Ellipsoid Algorithm

Created by Khachian in '79, the ellipsoid algorithm was the first known polynomial-time algorithm for linear programming. (There were other poly-time algorithms around, but they had not been proven to be so.) The ellipsoid algorithm does not solve linear programming problems of the general form, but rather it solves any problem stated in the following way:

Find x subject to $Ax \leq b$ (i.e. find any feasible solution).

However, we will show that any linear programming problem can be reduced to this form.

If L equals the number of bits used to represent A and b, then the running time of the algorithm is bound by $O(n^4L)$. Unfortunately the algorithm takes this much time for most problems (it is very unusual that it finds a solution any faster). Simplex's worse case running time is larger than $O(n^4L)$, but for most problems it does much better. Therefore Simplex

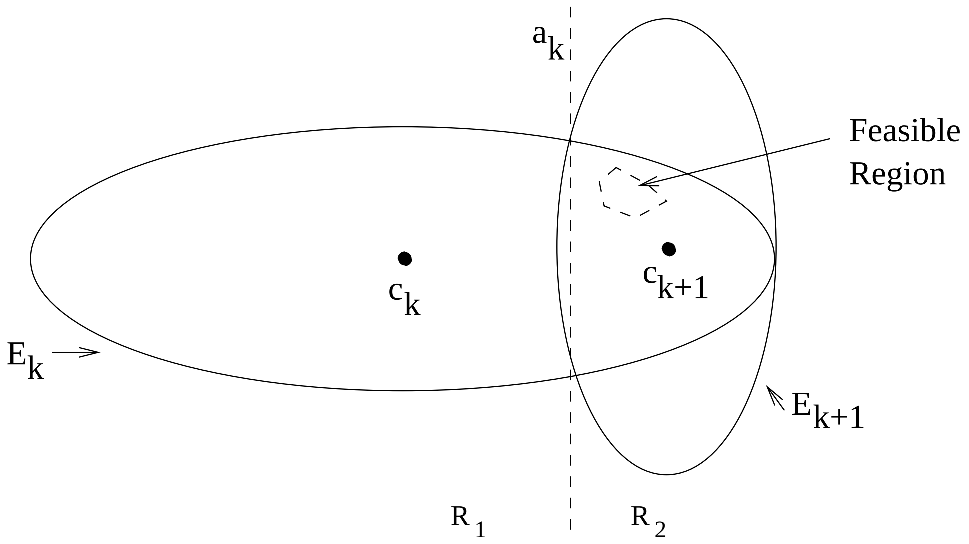

Figure 47: An iteration in the Ellipsoid Algorithm

ends up being better than the Ellipsoid algorithm in practice for most problems even though its worst-case running time is worse.

2.1 Description of algorithm

The idea behind the ellipsoid algorithm is that we start with some ellipsoid E_0 which encloses an area in the solution space, and is guaranteed to contain the feasible region of the solution. Basically, at each iteration of the algorithm, we create new ellipsoids E_k that are smaller than the previous, but still are guaranteed to contain the feasible region. Eventually, we will find a point that satisfies the problem (recall that we are satisfied with any feasible point).

Specifically, for each iteration k of the algorithm we do the following:

1. Determine $c_k =$ the center of E_k.
2. Determine $a_k =$ the constraint most violated by c_k.
3. Use the a_k hyperplane to split E_k into two regions. One of these regions R_1 contains c_k, and the other R_2 contains the feasible region.
4. We then construct E_{k+1} such that it contains all points in $E_k \cap R_2$, but is smaller than E_k.

Figure 47 illustrates these steps.

For n variables Khachian showed the following property of this algorithm:

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

2.2 Reduction from general case

As stated, we can convert the general form of linear programming problems to a form suitable to use the ellipsoid algorithm on.

Suppose we have the following problem stated in the general form: Maximize cx subject to $Ax \leq b$, and $x \geq 0$. We can convert it to the form $A'x' \leq b'$ as shown below:

Find any x, y such that

$$\begin{aligned} Ax &\leq b && (3) \\ -x &\leq 0 && (4) \\ yA &\leq -c && (5) \\ -y &\leq 0 && (6) \\ -cx + by &\leq 0 && (7) \end{aligned}$$

Here x and y together form x', the combination of the left sides of the equations form A', and the combination of the right sides of equations form b'. Equations (3) and (4) come from the original problem, and equations (5) and (6) come from its dual. Equation (7) will only be true at a single point when the original problem is optimized (the duality gap must be non-negative). In this way, if any solution to $A'x' \leq b'$ is found, then it will be the optimal solution for the original problem.

3 Interior Point Methods

Interior point methods are considered the first practical polynomial time methods for linear programming. Unlike simplex, the interior point methods do not touch any boundaries until the end when they find the optimal solution. Instead, these methods work by traveling through the interior of the feasible region (see Figure 48) by using a combination of the following two terms:

Optimizing term This moves the algorithm towards the objective.

Centering term This keeps it away from the boundaries.

Generally, interior point methods take many fewer steps than the simplex method (often less than 100), but each step is considerably more expensive. The overall relative cost of simplex and interior point methods is very problem specific and is effected by properties beyond just the size of the problem. For example the times depend strongly on the matrix structure.

Interior point methods have been used since the 50s for nonlinear programming. Roughly categorized, the methods include the following:

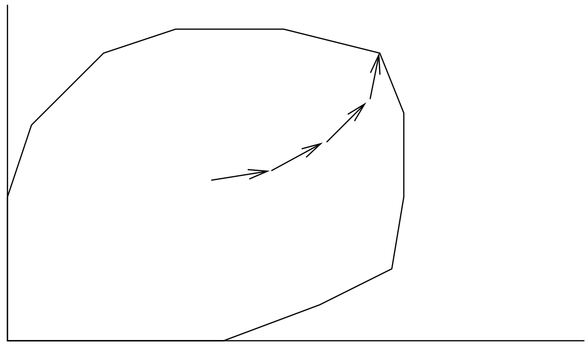

Figure 48: Walking on the interior points

	Fuel	Continent	Car	Initial
Size	13x31k	9x57k	43x107k	19x12k
Non-zeros	186k	189k	183k	80k
Iterations	66	64	53	58
Cholesky non-zeros	1.2M	.3M	.2M	6.7M
Time	2364s	771s	645s	9252s

Table 12: Performance of the central trajectory method on 4 benchmarks

Name	**Number of iterations**
Affine scaling	no known time bounds
Potential reduction	$O(nL)$
Central trajectory	$O(\sqrt{x}L)$

Karmarkar proved polynomial time bounds for his method (potential reduction) in 1984. Table 12 shows some example running times for a central trajectory method (Lustig, Marsten, and Shanno 94) on four different benchmark problems. As the table shows, the time spent in each iteration is greatly influenced by the number of Cholesky non-zeros present in the matrix.

In the following discussion we will assume we are trying to solve a linear program of the form

$$\begin{array}{rl} \text{minimize:} & cx \\ \text{subject to:} & Ax = b \\ & x \geq 0 \end{array}$$

3.1 Centering

There are several different techniques to compute the centering term for interior point methods. This section describes a few of those techniques.

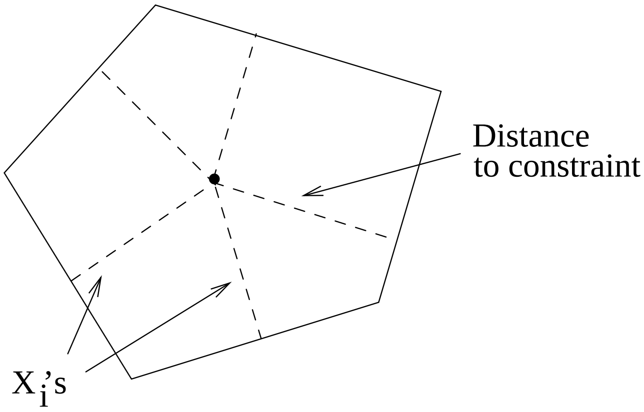

Figure 49: The analytical center

3.1.1 Analytical center

We can use analytical centering by minimizing

$$\sum_{i=1}^{m} \lg x_i$$

If $x_i = 0$ are the boundaries of the feasible region (we assume they are), then x_i are the distances to the boundaries from our current point. See Figure 49.

3.1.2 Elliptical Scaling

In elliptical scaling, we start with a small ellipsoid centered around the current point. We then expand the ellipsoid in each dimension of the problem. The expansion in a dimension stops when the ellipsoid touches a constraint. Since our inequality constraints are of the form $x \geq 0$, the equation will be of the form:

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

The ellipsoid will then be a measure of how uncentered the point is and can be used to scale space to bias our selection of a direction toward the longer dimensions of the ellipsoid, and thus away from constraints which are close by (the shorter dimensions). See Figure 50. We will see how this is used in the affine scaling method.

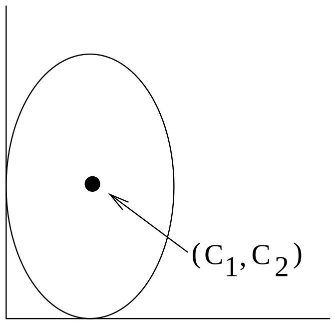

Figure 50: Elliptical Scaling

3.2 Optimizing techniques

3.2.1 Projection

If our problem is constrained to $Ax = b$, then we know that our feasible region is contained in a hyperplane. If A has n rows and m columns, then it is contained in a $m - n$ dimensional hyperplane in m dimensional space. (Assuming the rows in A are independent.) The hyperplane basis is called the *null space* of A We can also view A as being the "slopes" for the hyperplane, while b represents the "offsets".

We can use this to find the optimal solution for the problem. If $\vec{c}$ is the direction of the optimal solution, then we should take a step in the direction of $\vec{d}$. See Figure 51. We must do this because $\vec{c}$ would take us off the hyperplane and therefore, out of the feasible region. Instead, we project c into the hyperplane, and step in that direction.

Here's how we compute $\vec{d}$:

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

where $P = (I - A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1}A)$, the projection matrix.

Calculating $P \cdot c$ (the projection) must be done at each iteration of the method since, as we will see, A will change. So, solving for the projection matrix is a major computational concern.

We find that it is more efficient to solve $AA^{\mathrm{T}}w = Ac$ for w. We can then use w to calculate the projection because of the following relationship:

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

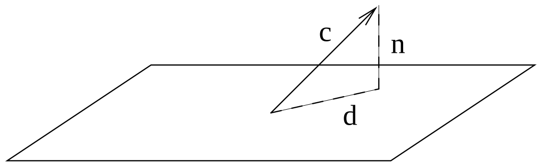

Figure 51: Projection into the null space

Computing $AA^{\mathrm{T}}w = Ac$ is more efficient because it can be solved using a sparse solver (e.g. Cholesky Factorization). Otherwise, calculating P directly can generate a dense matrix, which is problematic both in terms of runtime and in the space it would require.

3.2.2 Newton's Method

We can use Newton's Method to help determine the direction in which we should travel in the interior of the feasible region to optimize the solution. Often, the function we are trying to optimize will not be linear in x, and could be composed of transcendental functions such as $\lg or \ln$, as in the central trajectory case. We can approximate the minimum of these functions using a Newton's method, and more easily determine the direction of the optimal solution. We can see how this is done through a little math.

First, we will just look at the general case of minimizing some 1-dimensional function $f(x)$. We can write $f(x)$ as a Taylor expansion around some point x_0.

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

To get the minimum of the function, we take the derivative and set it to 0, which gives:

$$\begin{aligned} 0 &= f'(x_0) + f''(x_0)(x - x_0) \\ x &= x_0 - \frac{f'(x_0)}{f''(x_0)} \end{aligned}$$

In multiple dimensions we can write this in matrix form as

$$x - x_c = \vec{d} = (F(x_c))^{-1}\nabla f(x_c)^{\mathrm{T}}$$

where $f(x)$ is now a function of n variables, x_c is the point we are expanding about, and $F(x_c) = \nabla \times \nabla f(x)$ evaluated at x_c, an $n \times n$ matrix which is called the Hessian matrix, i.e.

$$F(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1 x_1} & \frac{\partial f}{\partial x_1 x_2} & \cdots & \frac{\partial f}{\partial x_1 x_n} \\ \frac{\partial f}{\partial x_2 x_1} & \frac{\partial f}{\partial x_2 x_2} & \cdots & \frac{\partial f}{\partial x_2 x_n} \\ & & \cdots & \\ \frac{\partial f}{\partial x_n x_1} & \frac{\partial f}{\partial x_n x_2} & \cdots & \frac{\partial f}{\partial x_n x_n} \end{bmatrix}$$

$\nabla f(x_c)$ is a vector of the first derivatives of $f(x)$ evaluated at x_c.

If we constrain the newton step to be on a hyperplane (i.e. $Ax = b$), it is called a *constrained Newton step*. In general we first pick a direction $\vec{d}$ based on a Newton step (or other method) and then project the step onto the constraining hyperplane, as discussed in the previous section.

3.3 Affine scaling method

The affine scaling method is basically a biased steepest descent search (biased away from the boundaries of the feasible region). Suppose we had the following problem to solve:

$$\begin{aligned} \text{minimize:}\quad & cx \\ \text{s.t.}\quad & Ax = b, \\ & x \geq 0 \end{aligned}$$

To solve this using the affine scaling method, at each iteration we solve:

$$\begin{aligned} \text{minimize:}\quad & cz \\ \text{s.t.}\quad & Az = 0, \\ & zD^{-2}z \leq 1 \end{aligned}$$

where D is a matrix whose only non-zero terms are along its main diagonal. The terms on this diagonal are the coordinates of the current point: $x_1, x_2, x_3, \cdots$. The equation $zD^{-2}z \leq 1$ is the definition for the interior of an ellipsoid, as described earlier (see Figure 50).

We update our x at each iteration by setting $x_{k+1} = x_k + \alpha * z$, where z is the direction we found in the above minimization problem, and α is selected to bring us close to the boundary, but not all the way there.

In this process, the cz minimization acts as the optimization term, while the $zD^{-2}z \leq 1$ acts as the centering term. In order to understand how this works, we should realize $Ax = b$ is a slice of the ellipsoid. In fact, it is the same hyperplane that we described in the ellipsoid scaling method. In order to minimize the function, we should step in the c' direction, where c' is the projection of c into the hyperplane. However, the affine scaling method steps in the direction of z.

How do we get z? We construct hyperplanes perpendicular to c, and keep moving them back until we find one such hyperplane that intersects the ellipsoid only once. We then construct z to be the difference between the current point and the intersection point. By stepping in the direction of z, we are centering the point because z will tend to point to constraints that are far away. In fact, as the current point comes closer and closer to a constrain, z will point further and further away from that constraint.

See Figure 52 for a picture of this construction of z, c', and the hyperplanes.

3.3.1 Computation at each iteration

In order to ease the computation of z, we make a substitution of variables, by setting $z = Dz'$. Then, we have

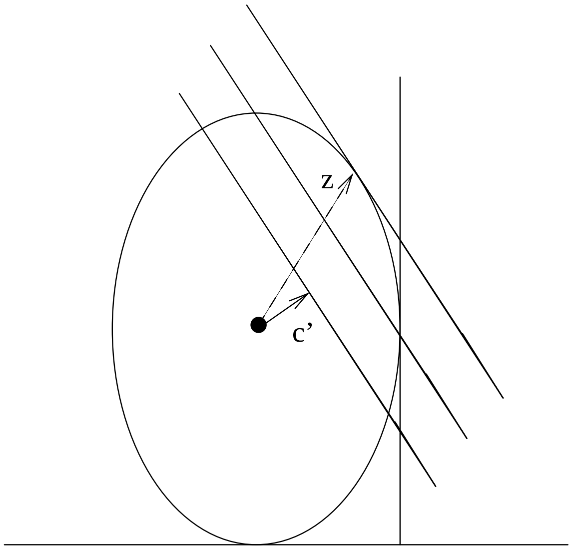

Figure 52: An iteration of the Affine Scaling Method

$$\begin{array}{rl} \text{minimize:} & cDz' \\ \text{s.t.} & ADz' = 0, \\ & z'^{\mathsf{T}} D^{\mathsf{T}} D^{-2} D z' \leq 1 \end{array}$$

The last equation simplifies to $zz \leq 1$, which is just a sphere. With this in mind, we can say our direction z' is unbiased and throw out the $zz \leq 1$ constraint. To find the minimum we now just take a steepest descent step on cDz' that keeps us within the hyperplane defined by $ADz' = 0$. This leads us to,

$$z' = -\underbrace{(I - B^{\mathsf{T}}(BB^{\mathsf{T}})^{-1}B) \cdot cD}_{\texttt{projection}}$$

where $B = AD$. To get z in our original space we just substitute back $z = Dz'$. The vector z' is negative since we are trying to find a minimum and therefore descending.

3.3.2 Overall computation

Now that we know what we have to compute during each iteration, let's step back and see how the entire process works. In the following algorithm we will assume we find the projection by using Cholesky factorization to solve the linear system, as described in Section 3.2.1. The linear equation could also be solved using other techniques, such as iterative solvers.

1. Find a starting feasible interior point x_0
2. Symbolically factor AA^{T}. Since the non-zero structure of BB^{T} on each step is going to be the same as AA^{T}, we can precompute the order of the operations in the Cholesky

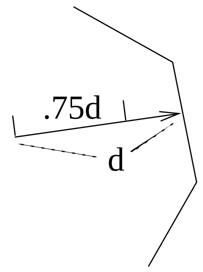

Figure 53: Picking an alpha

factorization, along with its non-zero structure. Since we are only doing this once, we can potentially spend some time trying to minimize the "fill" (the number of additional non-zero's that appear in the Cholesky factorization that did not appear in AA^{T}).

3. Repeat until done

   - $B = AD_i$, where D_i is a diagonal matrix with the x_i along the diagonal.
   - Solve for w in $BB^{\mathsf{T}}w = BDc$ by multiplying out BDc and factoring BB^{T} with help from the precomputed symbolic factorization.
   - $z' = -(Dc - B^{\mathsf{T}}w)$
   - $z = D_i z'$ (substituting back to the original space)
   - $x_{i+1} = x_i + \alpha z$

As mentioned, we should pick α so that the step is taken some fraction of the way to the nearest boundary. See Figure 53. In practice α is selected so that it goes .96 of the way to the boundary.

3.4 Potential reduction

In the potential reduction method, we reword the minimization problem to be

$$\begin{array}{rl} \text{minimize:} & q\ln(cx - by) - \sum_{j=1}^{n} \ln(x_j) \\ \text{s.t.} & Ax = b,\ x \geq 0 \\ & yA + s = 0,\ s \geq 0 \end{array}$$

As we can see, this method makes use of the problem's dual. The minimization criteria is partly based on the duality gap $(cx - by)$. Also, the s variables are the slack variables.

The first quantity in the minimization $(q\ln(cx - by))$ represents the optimizing term. It will go to $-\infty$ at the optimal solution. Therefore, this term will dominate near the optimal solution.

$\sum_{j=1}^{n} \ln(x_j)$ is the analytical center term.

In order to optimize the solution at each iteration, we can either do a constrained Newton step, or a transformation with a steepest descent. This second method is what Karmarkar used in his version of the potential reduction method.

There are several advantages of this method over the affine scaling method:

- There are guaranteed bounds on the running time.
- The duality gap used in the minimization allows the user to specify an error tolerance for the solution.
- It is more efficient in practice.

3.5 Central trajectory

The central trajectory is the method of choice these days. It falls under the category of Log Barrier methods, which date back to the 50s where they were used to solve nonlinear problems.

To perform central trajectory, at each iteration step k, we do the following:

1. $\begin{array}{rl} \text{minimize:} & cx - \mu_k \sum_{j=1}^{n} \ln(x_j) \\ \text{s.t.} & Ax = b, x > 0 \end{array}$

   (approximate using a constrained Newton step)

2. Select $\mu_{k+1} \leq \mu_k$

This will terminate when μ_k approaches 0.

It turns out that with the "appropriate" schedule of μ_k, the central trajectory approach reduces to the potential reduction approach. We can therefore view it as a generalization.

Currently the primal-dual version of the central trajectory approach using "higher order" approximations (rather than Newton's) seems to be the best interior-point algorithm in practice.

4 Algorithm summary

1. Overall, the current state of the art uses very sophisticated algorithms to solve linear programming problems in practice. Even the linear algebra packages used to perform the computations use complicated math in order to optimize computation time.

2. Practice matches theory reasonably well.

3. We find interior-point methods dominant when:

   - Large n
   - Small Cholesky factors
   - The problem is highly degenerate, meaning the many of the constraints cross each other.

4. Ellipsoid algorithms are not currently practical, since they always take the worse case time.
5. Large linear programming problems take hours/days to solve. Parallelism is very important to speed up the optimization.
6. There are many interior-point methods, but they all share the same intuition.

We looked at a slide that showed the running times of large models. For some interior point methods, $AA^ty = Ac$ was too dense, so the programs ran out of memory and couldn't complete their tasks.

Some algorithms use interior point methods to get close to a solution, then snap to a boundary and use simplex methods to finish off the answer.

5 General Observation

Mathematics has become increasingly important for "Real World" algorithms. Just over the last 10 years, the complexity of math employed in algorithms has increased dramatically.

- Compression
  1. Information theory
  2. Markov models
  3. Transforms (for lossy compression)
- Cryptography
  1. Group theory
  2. Number theory
- Linear Programming
  1. Matrix algebra
  2. Optimization
  3. Graph theory