Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 52

Hello everyone, this is the second lecture of week 11. For the past two weeks and the first lecture in week 11, we have been discussing linear programming. We have discussed two algorithms: one of them is the simplex method and the other is the affine scaling method. From this lecture onwards, we will learn about Karmarkar's algorithm, Karmarkar's method.

I had mentioned in passing about Karmarkar's method in some of the previous lectures. If you recall, this algorithm was given by Narendra Karmarkar. He was a graduate from IIT Bombay; he finished his B.Tech at IIT Bombay in 1978. He gave this algorithm which you can say challenged the might of the simplex algorithm. If you look at the history of the algorithms for linear programming, Dantzig came up with the simplex method in 1946 and the ellipsoid method was actually discovered later. The advantage of the simplex method over the ellipsoid method is because in practice the simplex method was much faster for most practical linear programs, but in the worst case the ellipsoid method was better.

We have seen an example, a pathological example where the simplex method actually takes 2^n iterations. That is the worst case for the simplex method, but the ellipsoid method has a guarantee that for any linear program the number of iterations is polynomial in n, polynomial in the number of variables.

It does not blow up like 2^n or anything of that sort. But people did not use the ellipsoid method. When given a choice between the simplex method and the ellipsoid method, people were always using the simplex method because for most practical purposes the simplex method was much, much faster than the ellipsoid method.

This was the case until Narendra Karmarkar proposed this algorithm called Karmarkar's method, which is popularly known as Karmarkar's method or Karmarkar's algorithm. This again has the worst-case guarantee of being polynomial in n and it performs better than the ellipsoid method. In many practical cases Karmarkar's algorithm was at least performing close to the complexity of the simplex method, or sometimes performing better also. You can say that people switched to different kinds of algorithms only after Karmarkar's method was formulated in 1984, designed in 1984.

Just to give the timeline of how the algorithms came out, note that first Dantzig designed the simplex algorithm or simplex method in 1946, so it is quite old. The ellipsoid method, I do not recall the year, but was discovered much later, was designed much later.

So if you compare the simplex and ellipsoid methods, the ellipsoid method provides a worst-case guarantee, whereas the simplex method was faster for most practical settings. People were always using the simplex method. All this until Karmarkar designed this method in 1984. This gives a worst-case guarantee, I should say a better worst-case guarantee than the ellipsoid method, and performed either close to the simplex method or sometimes better. This was clearly much better than the ellipsoid method, so if you compare practical purposes sometimes it was better than the simplex method and even when the simplex method was better, Karmarkar's was still performing close to what the simplex method would have given. In many places after Karmarkar gave his algorithm, people started employing Karmarkar's algorithm rather than the simplex method. But the simplex method still continues to be in use.

Now you must actually be asking about the affine scaling method. We skipped the ellipsoid method because nobody uses it now. We have an algorithm that gives a better worst-case guarantee, which is Karmarkar's algorithm, and even for practical purposes the simplex method was much better. So the ellipsoid method is just in academic circles now. But affine scaling is something that people study widely even today. When was it discovered and what is its performance? The affine scaling method was actually discovered after Karmarkar's algorithm. It is, you can say, a simplification of Karmarkar's algorithm. I have not read about worst-case guarantees or anything of that sort for the affine scaling method. Karmarkar's algorithm comparatively is a little more difficult to understand. Affine scaling is, you can say, a simplified version which helps us understand how Karmarkar's algorithm works. The intuition is much clearer with the affine scaling method than with Karmarkar's algorithm.

That is the reason I first covered the affine scaling method, so that you understand what is going on. I will discuss Karmarkar's algorithm by first discussing the differences between the affine scaling method and Karmarkar's algorithm before actually discussing the algorithm in detail. I will also write this down: the affine scaling method is a simplified version of Karmarkar's method or Karmarkar's algorithm designed in 1985. The simplification was because this gives a better intuition of what is actually happening than Karmarkar's method. It gives a better intuition to understand the algorithm. That is roughly what Karmarkar's algorithm is about. I wanted to discuss this since learning different kinds of algorithms, we need at least a rough comparison between these algorithms so that you understand which one to use when you are facing a problem in a practical setting, an optimization problem, a linear programming problem in a practical setting.

Let us start with the **differences between Karmarkar's algorithm and the affine scaling method**. You can recall that in the affine scaling method we convert the given problem into the standard form. That was the first step. The first step is just to convert the given LP to its standard form. That is how the affine scaling method starts. But here you convert it into a slightly different form which Karmarkar calls the canonical form. I will tell you what a canonical form is. But before that, just to put things in perspective, the form that you convert in the affine scaling method is the **standard form**, that is

minimize $c^T x$

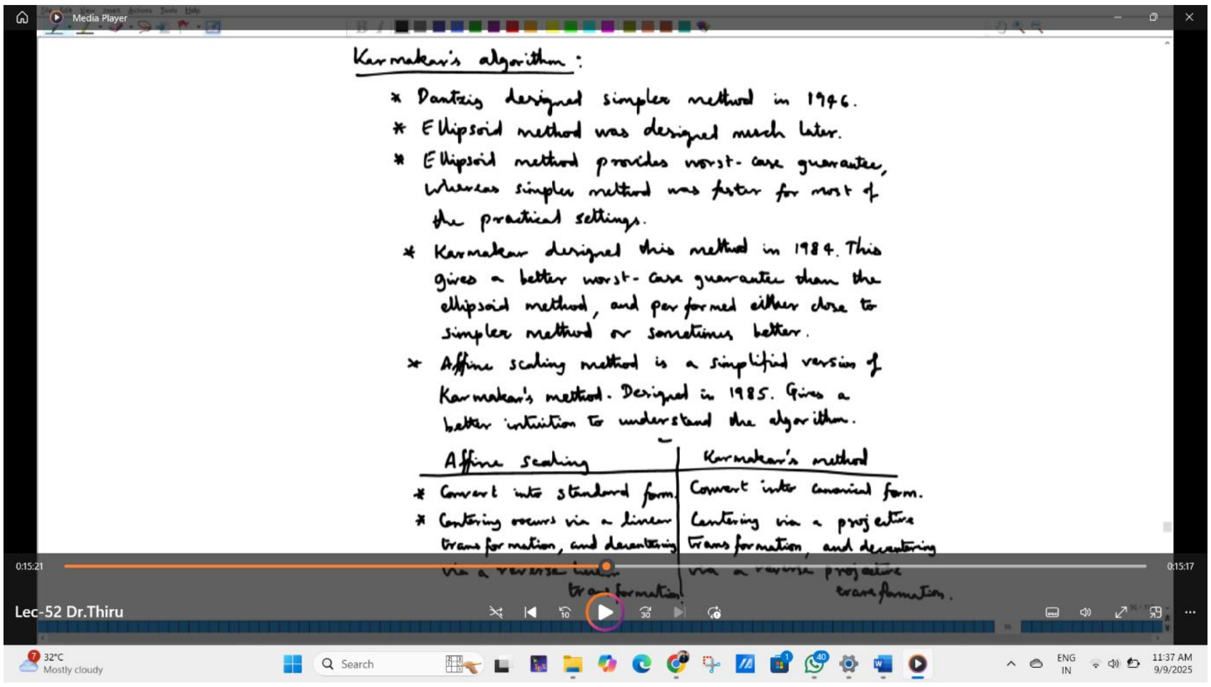
subject to

$A x = b, x \geq 0.$

But here you convert it into a slightly different form called the canonical form. That is the first difference.

If you recall what you do in affine scaling, you first center the initial point or whatever point you have, then you find the descent direction and project it onto the feasible set, and then you move over there, and then decenter it. That is the procedure. The first step is the centering process. The second step is you find the direction and move over there. And the third step is the decentering process. This is the formal algorithm. The first step is centering, the second is moving along a direction, and the third is decentering. The other thing that you have to see is the centering process actually uses a linear transformation, multiplying with a matrix, so that is a linear transformation, and the decentering process also uses another linear transformation; you can call it the reverse linear transformation, but nevertheless both are linear transformations. But here what you use is actually called a projective transformation and not a linear transformation. In affine scaling, centering occurs via a linear transformation and decentering via a reverse linear transformation. In Karmarkar's method, centering actually occurs via a projective transformation and decentering via a reverse projective transformation. You may not immediately understand what a projective transformation is, but I would like to indicate all of this before going into the method because without this you may not understand what is going on at all. Basically, you can say that these are the differences between the affine scaling method and Karmarkar's method: instead of standard form, you convert it into a canonical form, and instead of linear transformation and reverse linear transformations, you use a projective transformation and a reverse projective transformation. Based on this, there are other issues that you have to handle.

(Refer Slide Time 15:21)



Karmakar's algorithm :

* Dantzig designed simplex method in 1946.
* Ellipsoid method was designed much later.
* Ellipsoid method provides worst-case guarantee, whereas simplex method was faster for most of the practical settings.
* Karmakar designed this method in 1984. This gives a better worst-case guarantee than the ellipsoid method, and performed either close to simplex method or sometimes better.
* Affine scaling method is a simplified version of Karmakar's method. Designed in 1985. Gives a better intuition to understand the algorithm.

Affine scaling	Karmakar's method
* Convert into standard form.	Convert into canonical form.
* Centering occurs via a linear transformation, and decentering via a reverse linear transformation.	Centering via a projective transformation, and decentering via a reverse projective transformation.

The other differences come because of these two differences. Let us get started with the method now, given that I have given an overview of Karmarkar's method. First, let me describe the canonical form. A linear program is said to be in its **canonical form** when you are

minimizing $c^T x$

subject to

the conditions that $A x = 0$, $1^T x = 1$, and $x \geq 0$.

You can see that it is slightly different from the standard form. The standard form, if you recall, is of the form

min $c^T x$ with respect to x subject to the conditions that $A x = b$ and $x \geq 0$.

Here you can see that there is a small change: instead of

$A x = b$, you have $A x = 0$ and $1^T x = 1$.

The other conditions are there anyway, where you have A is an $m \times n$ matrix with rank m.

This is not the binding condition.

The binding condition is that the solution of this problem should actually, or rather not the solution, the optimal value of the objective function should be 0.

The optimal value, which is $c^T x^*$, should be equal to 0.

That actually is the more binding constraint in the canonical form.

The other ones are much simpler; it is handleable, but this condition that the optimal value $c^T x^*$ should be equal to 0.

What it says is that you must actually know the answer in some sense.

If you do not know the answer, you must know the optimal value. You may not know the solution x^*, but you must know that when you substitute the solution, the optimal value of the objective function should go to 0, should be actually equal to 0.

That is the canonical form that Karmarkar's algorithm asks for. Now let us actually try and understand how to convert a given problem to satisfy all of these conditions.

Suppose you are given a linear program in general form. If you are given in general form, it will be of this form:

minimize $c^T x$

subject to,

$\bar{A} x \leq \bar{b}$ and $\tilde{A} x = \tilde{b}$.

That is the very general form. You know how to convert this into standard form, which is of the following form:

minimize $c^T x$

subject to $A x = b$ and $x \geq 0$.

This c and the original c could be different. Let me actually call this $\hat{c}$ or something.

This will be of the form $A x = b$ and $x \geq 0$.

This is the standard form with A an $m \times n$ matrix with rank of A equal to m.

Now how do I convert this into the so-called canonical form? I am just going to introduce one more variable. Let us assume that now it is in standard form. I am just going to discuss converting a problem in standard form to a problem in canonical form.

If this is of the form, let us say, you are minimizing $c_1x_1 + c_2x_2 + \ldots + c_nx_n$.

What I am going to do is introduce a new variable, x_{n+1}, and I am going to fix that to be equal to 1. Possibly I will write down the form and then explain how both of them are equivalent. I am claiming that the canonical form that I have written and the standard form that I have written are both equivalent. Why is that the case?

Note that I have introduced a constraint $x_{n+1} = 1$. That means the constraint here can actually be written as $A x - b = 0$, which is nothing but $A x = b$.

And $x_{n+1} = 1$ means that $x_{n+1} \geq 0$ is a redundant constraint, but that is because we want it in the form that we are looking forward to.

That is the reason I have written $x_{n+1} \geq 0$ anyway.

And I have also included this part - $v^* x_{n+1}$, where v^* is actually the optimal value of the objective function of the standard form.

If $x_{n+1} = 1$, then the optimal value of the objective function here will be actually equal to 0.

That is the reason how the problem written in the standard form and the problem that I have written in the canonical form are both equivalent. You can see that it is clearly in the canonical form because this is a problem in n+1 variables.

It is of the form that the first constraint that I have written is of the form $A x = 0$.

Just that the new matrix now, you can write that as $\hat{A} x = 0$ where $\hat{A}$ is $[A, -b]$.

You augment -b with A.

You also have $1^T x = 1$ where you have $[0, 0, \ldots, 0]$ (n times) and a 1.

That will give you $x_{n+1} = 1$, $1^T x = 1$, and of course $x \geq 0$.

This is clearly in the canonical form, and the problem given in the standard form and the problem given in the canonical form are both equivalent to one another. Any given LP can actually be converted into a canonical form. You can see that just from this. What you are going to do is first convert it into standard form and add a dummy variable $x_{n+1} = 1$, and then do this simple conversion: instead of $A x = b$, you write $A x - b x_{n+1} = 0$, and the objective function is $c^T x - v^* x_{n+1}$ where v^* was the optimal value of the standard form.

The canonical form is very clear. But let me tell you there is a catch.

The catch is actually with v^*.

You do not know v^* in general. Given a linear program in standard form, it is not that some genie comes and gives us the optimal value v^*.

That never happens. You can write this when you know v^*.

If you do not know v^*, you cannot write this. That is correct, but we will take care of that as we go along. This is the way you first convert it into canonical form.

We will now go ahead to the next step. We have converted the given problem to the canonical form. Now, how do we do the centering, decentering, and all that? That is something we will go ahead with at this point. This is done. Instead of converting into standard form, we have converted into canonical form. The next step is to take an initial point and do centering, but not using a linear transformation, but using a projective transformation.

Let us do that, but before that I will have to tell you what a projective transformation is. Assume that the transformation is given by the following equation.

Before that, capital X is the diagonal matrix of x. And y is given by $X^{-1}x / (1^T X^{-1} x)$.

Please recall that 1 is a vector of all ones. This is the projective transformation. This is slightly different from the linear transformation, which was just capital $X^{-1}x$.

That was something we saw. You can recall that in affine scaling, y_k is just $X_k^{-1} x_k$.

Similarly, here, if capital X_k is the diagonal matrix of x_k, then $y_k = X_k^{-1} x_k / (1^T X_k^{-1} x_k)$. Something that is very clear is that $1^T y_k$ is actually equal to 1.

That is because if you put 1^T, you get cancellation and you get 1.

Now how do you get back x_k, little x_k, from little y_k? That is also important; we will derive that now. Note that capital $X_k y_k = x_k / (1^T X_k^{-1} x_k)$.

That implies that x_k is actually capital $X_k y_k * (1^T X_k^{-1} x_k)$. But this is not complete because this depends on x_k. We will correct this shortly.

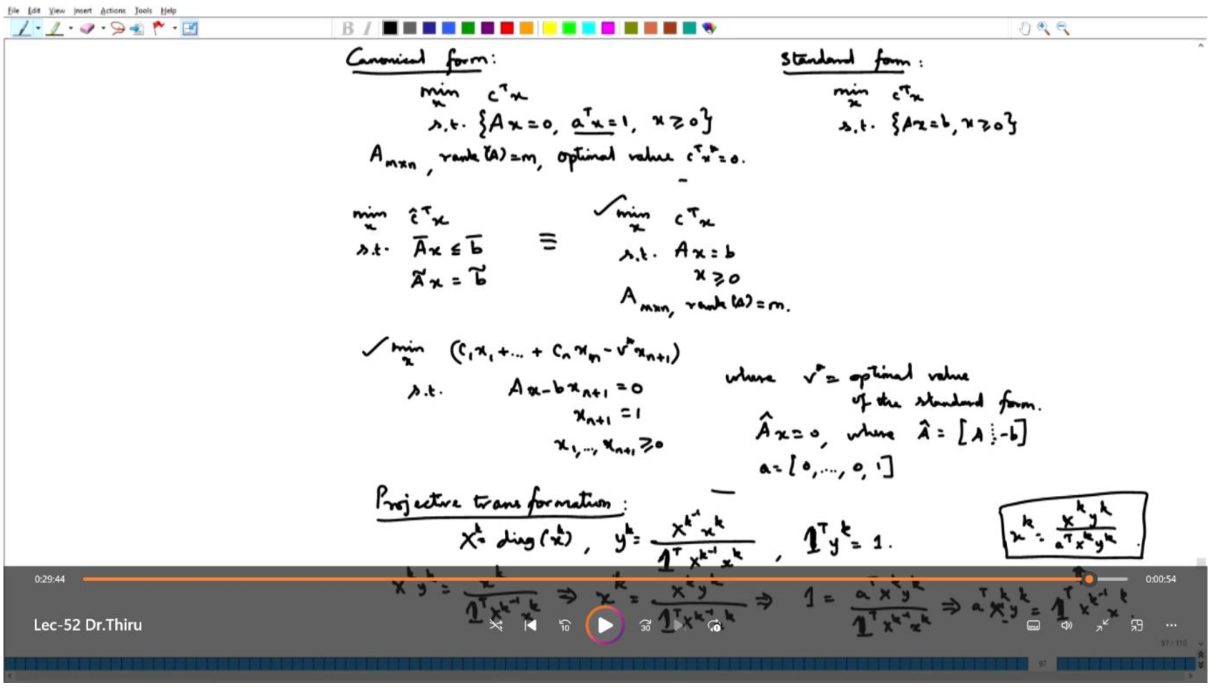
Note that $1^T x_k$ should be 1. That is coming from this constraint. So $1^T x_k = 1$. That means that $1 = 1^T (X_k y_k * (1^T X_k^{-1} x_k))$. That implies that $1^T X_k y_k = 1^T X_k^{-1} x_k$.

So I can replace $1^T X_k^{-1} x_k$ by $1^T X_k y_k$.

And because of which we actually have $x_k = (X_k y_k) / (1^T X_k y_k)$.

This is the projective transformation by which x is converted to y, and this is the reverse projective transformation where y is converted to x.

(Refer Slide Time 29:44)



Canonical form:

$$\min_x \; c^T x$$
$$\text{s.t. } \{Ax = 0,\ \underline{a^T x = 1},\ x \geq 0\}$$
$$A_{m\times n},\ \text{rank}(A) = m,\ \text{optimal value } c^T x^* = 0.$$

Standard form:

$$\min_x \; c^T x$$
$$\text{s.t. } \{Ax = b,\ x \geq 0\}$$

$$\min_x \; \hat{c}^T x \quad \text{s.t. } \bar{A}x \leq \bar{b},\ \tilde{A}x = \tilde{b} \quad \equiv \quad ✓\ \min_x \; c^T x \quad \text{s.t. } Ax = b,\ x \geq 0$$
$$A_{m\times n},\ \text{rank}(A) = m.$$

$$✓\ \min_x \; (c_1 x_1 + \dots + c_n x_n - v^* x_{n+1})$$
$$\text{s.t. } Ax - b x_{n+1} = 0$$
$$x_{n+1} = 1$$
$$x_1, \dots, x_{n+1} \geq 0$$

where v^* = optimal value of the standard form.

$$\hat{A}x = 0,\ \text{where } \hat{A} = [A \,\vdots\, -b]$$
$$a = [0, \dots, 0, 1]$$

Projective transformation:

$$X^k = \text{diag}(x^k),\quad y^k = \frac{X^{k^{-1}} x^k}{\mathbb{1}^T X^{k^{-1}} x^k},\quad \mathbb{1}^T y^k = 1.$$

$$\boxed{x^k = \frac{X^k y^k}{a^T X^k y^k}}$$

$$X^k y^k = \frac{x^k}{\mathbb{1}^T X^{k^{-1}} x^k} \Rightarrow x^k = \frac{X^k y^k}{\mathbb{1}^T X^{k^{-1}} x^k} \Rightarrow 1 = \frac{a^T X^k y^k}{\mathbb{1}^T X^{k^{-1}} x^k} \Rightarrow a^T X^k y^k = \mathbb{1}^T X^{k^{-1}} x^k.$$

What we have seen until now in Karmarkar's method is we saw how to convert a given LP into the canonical form, and then we have also learnt what a projective transformation is. We will shortly write down the algorithm properly in the next lecture. Thank you.