Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 55

Hello everyone. This is the fifth lecture of week 11. Recall that we were learning about Karmarkar's method in the past few lectures. We learnt the procedure completely and we have also now worked out an example. In this lecture, we will work out the so-called pathological example that we had actually discussed in the simplex case. If you recall the simplex algorithm, we had actually mentioned about one of the problems where if the value of n increases, the number of steps in the simplex method is 2^n.

Let us see that example once. This was the example if you could recall. This was one of the issues that were pointed out in the simplex method. We are maximizing this particular quantity

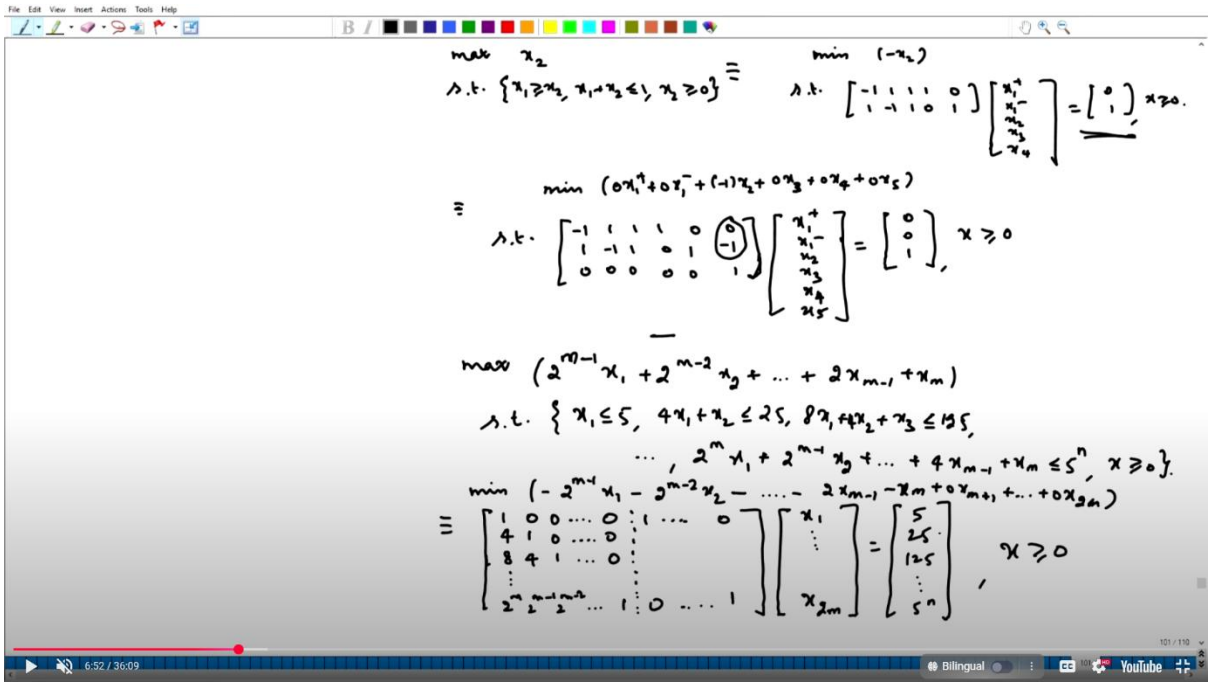
$2^{m-1} x_1 + 2^{m-2} x_2 + \cdots + 2 x_{m-1} + x_m$

subject to m conditions as well: $x_1 \leq 5$, $4 x_1 + x_2 \leq 25$, $8 x_1 + 4 x_2 + x_3 \leq 125$, $\cdots$,

and this condition $2^m x_1 + 2^{m-1} x_2 + \cdots + 4 x_{m-1} + x_m \leq 5^m$,

and x greater than or equal to 0, meaning all x_1 to $x_m \geq 0$.

(Refer Slide Time 6:52)



This took 2^m steps in the simplex method, and we have actually come to Karmarkar's method saying that it is much faster than the simplex method, at least in the pathological cases. It would be good if we can work out this pathological case and show that we are actually not performing really bad as we are performing in the simplex method. Let us check; let us take up the same problem. I will write down this problem. The problem is this:

maximize $2^{m-1} x_1 + 2^{m-2} x_2 + \cdots + 2 x_{m-1} + x_m$.

This is what you are maximizing, and it is subject to m conditions again.

It is $x_1 \leq 5$, $4 x_1 + x_2 \leq 25$, $8 x_1 + 4 x_2 + x_3 \leq 125$, $\cdots$,

and the final constraint is this: $2^m x_1 + 2^{m-1} x_2 + \cdots + 4 x_{m-1} + x_m \leq 5^m$, and $x \geq 0$.

This is the problem that we are going to solve. We will first write this in the standard form as we did in the affine scaling method. Since there are m constraints, all of them inequality constraints, we should have m more slack variables. You will have A to be the following: it will be 1, and many zeros, m zeros to be very precise, because there is x_1 to x_m, and here you will fill up later. This will be 4, 1, 0, and 0; this will be 8, 4, 1, and all zeros;

this goes up to 2^m, 2^{m-1}, 2^{m-2}, $\cdots$, and finally it will be 1.

These are the constraints, and here you will have an identity.

It is 1, 0; fine, these can be filled up. And you have x_1 to x_{2m}.

For each of them, you have added a slack variable. So you have 2m variables now. And here you have 5, 25, 125, $\cdots$, 5^m. And x is not equal to 0. I should have written the minimum at the top. I will write that.

This is minimize $-2^{m-1} x_1 - 2^{m-2} x_2 - \cdots - 2 x_{m-1} - x_m$.

If you want, you can write $+ 0 x_{m+1} + \cdots + 0 x_{2m}$ because you have 2m variables.

This is the standard form. I now have to convert this to the canonical form where you actually add a row with all 0s and a 1 below. That is one thing that you do. You also add one more variable and call it as 1. And the B on the right side becomes -B within the A matrix. That is something that we have worked out in the previous example as well. The 0, 1 here, this part has become 0, -1 in this place. You will have this B matrix on the right coming in. The standard form becomes like this:

minimize $-2^{m-1} x_1 - \cdots - 2 x_{m-1} - x_m + 0 x_{m+1} \cdots + 0 x_{2m}$.

Here you have a slightly different matrix. This is 1, 0, $\cdots$, 0, 1, 0, $\cdots$, 0, and here you have -5.

Have 4, 1, $\cdots$, 0, 0, 1, $\cdots$, 0, this is -25; go ahead;

this will be 2^m, 2^{m-1}, $\cdots$, this is 1,

and this will be 0, 0, and 1, -5^m; and

this is going to be 0, 0, all 0s, and you still have a 1 at the end.

This is the A matrix, and here you will have x_1, x_2, ⋯, x_{2m}, and x_{2m+1} that is equal to 1.

Here everything is 0 except the last one where it is actually equal to 1. And you have $x \geq 0$ for all 2m + 1. So $x_1 \geq 0$, $x_2 \geq 0$, ⋯, $x_{2m+1} \geq 0$.

I will also change this as 0 x_{2m+1} because that is also there. This is the final problem that we are going to solve. I am going to write this matrix for a general value of m. That means I will change the values of m and show how many iterations it takes in Karmarkar's algorithm when you just implement this particular problem. Let us go ahead to the implementation part. This was the algorithm that we implemented in Karmarkar's method for some other problem. The later part we are going to copy down. But since we are actually going to write down the matrix A, c, x everything for general values of m, that will take a little bit of time. We will do that. I will write down the text which is Karmarkar's method for a pathological case. It is actually a pathological example for the simplex setting, but nevertheless it is a pathological example everywhere.

We will first initialize m = 3, which is the simplest, which is a very simple case. The simplest case is m = 1, but we will start with this. Now I am going to rewrite A, c, and x; everything else is going to be the same. The rest of the code is going to be the same. Let us write it for a general value of m.

Let us initialize c to be an array of zeros. I can write it as np.zeros(2m + 1).

So x and c, if you can recall, they have m + 1; you have x_1, x_2, ⋯, x_{2m+1}. Since it is $c^T x$, both c and x will have 2m + 1.

I will first initialize c to be 2m + 1 zeros and x to be 2m + 1 ones.

For A, I will initialize it to be zeros of size m + 1 by 2m + 1. You have m + 1 rows: m rows and the last one row, so it is m + 1, and the number of columns is 2m + 1. I have initialized the sizes.

Now I will change the values accordingly. For c, please note that x_1 to x_m has some values, but after that it is 0s. I do not have to change. Please note this.

I have got the A matrix is fine. Your c matrix is this: -2^{m-1}, -2^{m-2}, -2^{m-3}, ⋯, -2, -1, and then you have zeros. This is m + 1 zeros.

This is m value. This m + 1 zeros can be left out as such, and I will just code for the others. You can see that $c_0 = -2^{m-1}$, $c_1 = -2^{m-2}$, and so on up to $c_{m-1} = -2^0$.

So I will write this as $c_i = -1 \cdot 2^{m-1-i}$, so that if i = 0 it is 2^{m-1}, if i = 1 it is 2^{m-2}, and when it is m - 1 it is 2^0.

So for x_i, we have to choose an interior point.

I think we did that a little while ago for the affine scaling method. Let us check if we have it as such. We have it. If you recall, we initialized it; we initialized x_1 to x_m to be 1. And the remaining ones in such a way that A x = b. Here we will have to do it in such a way that A x = 0.

So again, x, this is x_0. I said we will have 1's for the first m values. This is m 1's. We need not disturb this. We have 1's anyway. And this one is 4 because 1 + 4 - 5 = 0.

See the first row. You have 1, and you have 4 - 5 = 0.

In the next row, I am writing this as 20 because 4 + 1 = 5, 5 + 20 = 25, 25 - 25 = 0.

And the next one is it will be 8, 4, and 1. So 8 + 4 + 1 = 13.

So 13 you have to add 112. So you have 13 + 112 = 125 - 125 will give you 0.

That is how you are going to write. And the last one is 1. If it is for particular values of m, I can find out what these values are. But how do I write it in general? That is one question that we need to address. I will leave this as an exercise, but you can be convinced that the numbers are actually the following. It is 5 power.

I will write this as m - i. So $5^{m-i+1} - 2^{m-i+1} + 3$.

That is, I have to write i - m then. It is i - m + 1, and this is also i - m + 1.

Let us check for certain values. I claim that for x_i it is this when i is m, m + 1, up to 2m - 1. For 2m it is 1.

We can leave that. Let us check this for certain values of m. What is x_m? $x_m = 5^1 = 5 - 2^2 = 4$ and + 3.

So it is 5 - 4 + 3 = 1 + 3 = 4.

That is correct; it is 4. What is x_{m+1}? It is 25 - 8 + 3, which is 20.

What is x_{m+2}? It is 125 - 16 + 3.

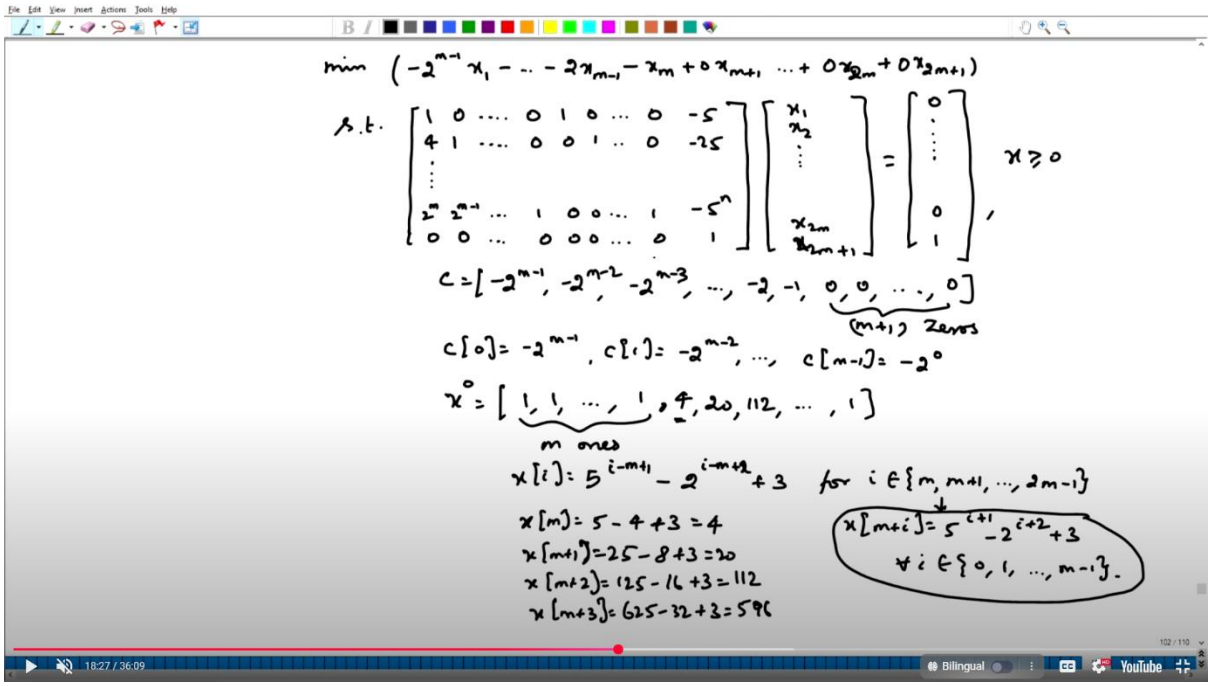
So you have 128 - 16 = 112. And $x_{m+3} = 625 - 32 + 3$.

So you will have 628 and yes this is 596. And so on. I will leave you to check why this is the right answer. But nevertheless, you can also check from the previous code that this number was 596.

I am just going to replace x_i, not every i; it is actually m + i. So since it is m + i, m + i - m is just i, so I will write it as $5^{i+1} - 2^{i+2} + 3$.

So instead of m, so if you substitute from 0 to m - 1, I can substitute i as m + i. I can rewrite this as $x_{m+i} = 5^{i+1} - 2^{i+2} + 3$ for all i in 0 to m - 1.

(Refer Slide Time 18:27)



I have just changed the way I write i.

So that is what I have written down here: $5^{i+1} - 2^{i+2} + 3$.

And now we will write A. I have written down c and x in general for all values of m, but for A we have to struggle to some extent. The m + 1th row we will write later because we just have to replace one entry by one; we will concentrate on the other entries. We will first put this diagonal as one and this diagonal as one.

So that will be $A_{i,i} = 1$ and $A_{i,i+m} = 1$.

Those two diagonals are done. We will also finish this.

So that is -5^{i+1}. So that is $A_{i,2m}$. So you start with 0, that is column number 2m, so that is just $-1 \cdot 5^{i+1}$.

Now we still have to fill these diagonals. Recall that this diagonal is 1, that is ok; the diagonal just below the diagonal of 1's you have 4's; below that you have 8; below that you have 16; below that you have 32; and so on. We will do the general filling now. This will be another loop. If it is ok, we will start actually from; so it is just up to m - i. So if it is 0, we have filled the 0th column here. Since it is the first column, we are starting from row 0 and row is 1, but the column is 0. This has to be; the column number has to be j. And this is; so what we have to fill is 2^{j+1}. We will start from 1 and then go up to m - i.

This will be i + j, i, and this is 2^{j+1}.

To explain what is happening: note that when i = 0, and we note that we are starting j from one. The first entry will be A of 1, 0 = 4. So that is what we have here: A of 1. This is 0th row and 0th column. The first row and 0th column has 4.

So $A_{1,0} = 4$. And this one is 2, 1.

So when you put i = 1, that is in the next cycle. You will have $A_{2,1}$, but this will be 4 because j = 1 again. And similarly, 3,2; 4,3; everything will be 4.

If you are looking for 8, here you will see that A of, so it starts with i = 2, I mean when, sorry j = 2. So when i = 0, it is $A_{2,0}$, which is 8, 2^3. And the next one will be $A_{3,1}$, which is again 8, and so on. We have filled c, x, and A. And this c and x can be removed. We have not written down the final row of A, so that's one thing that we have to write as well.

The last row where you have all zeros and a 1. So that has to be filled up with A of, so that is m + 1, 2 · m, sorry, m, 2 · m, that is actually 1.

So that is just one entry that we need to change. Little a is the last row. n is the shape of 0. So that is the reason I started with m, because n we are using for something else, 2m + 1 basically, that is n. Capital X is diag of x, k = 0, and we have the μ and the rest of the algorithm. Let us check if this algorithm works for m = 3.

Double parenthesis, yes, so we have the answer. The answer is 0, 0, 125, 5, 25, 0, and 1. That is something we know already. Good. We have got that in 214 iterations. Let us note that down. This is Karmarkar's algorithm.

Let us write down a table for the value of m: what is the number of steps that Karmarkar's method takes and what are the number of steps that the simplex method takes. For m = 3, we know that simplex takes $2^3 = 8$. But Karmarkar's has taken 214 iterations. That is much more than 8. We were told that Karmarkar's method performs much better than the simplex algorithm. Maybe it did not work for m = 3.

Let us try for slightly higher values of m. Let us actually first start with m = 5, which is slightly higher. Let us see how many iterations Karmarkar's method takes. It has taken 600 iterations. Yes, but the answer is 0, 0, 0, 0, 3125, 5, 25, 125, 625, 0, and 1. Good. For 5, it has taken 600 iterations, and we know that simplex takes $2^5 = 32$.

Still simplex is performing better. Let us do it for a slightly higher value, say 10. For m = 10, let us see how many iterations does our Karmarkar's algorithm take. It has taken 4262 iterations. Again, you can see that the first four are 0, the next four are 0, this is 0, and this is something very high. That is expected; it is about 5^{10}.

That is expected to be very high. But it has taken 4262 iterations, but the simplex algorithm will take $2^{10} = 1024$. Still simplex is performing better. But one thing that you should have noted is that the number of steps Karmarkar's algorithm took in comparison with simplex was much higher. This is roughly 4 times of simplex when you put m = 10. That is understandable. Let us try some higher values as well, maybe m = 12, to see how many iterations does

Karmarkar's method take. It has taken roughly 7488 for 12, and simplex would have taken 4096, since 2^{12} is 4096.

Let us possibly stop with 15. Let us check how many iterations does Karmarkar's algorithm take. We will do it for m = 14. The final one that we will attempt now. We will try with m = 14 and check how many iterations that does Karmarkar's method take. It takes 11588, and what is 2^{14}? If you have computed already, you would have got 16384. So at least there is some value of m for which Karmarkar's algorithm takes a lesser number of iterations than the simplex algorithm. This is not surprising because when we say 2^n, 2^n seems to be a very small number when you actually consider low values of m.

For example, take m = 3. So 2^3 is just 8. 8 is a very trivial number. If the simplex algorithm is just taking 8 iterations, 8 iterations is very low. Why did we really worry about saying, oh 2^m? Maybe 2^m is not large if m is small, but it shoots up really very badly when m is large. I am not sure how many students would have heard of the P class and NP class in the theory of complexity. When we discuss the complexity of algorithms, we say certain algorithms as polynomial time algorithms which is P, and NP is actually non-deterministic polynomial. I would not like to elaborate on this since it is beyond the scope of this course, but I will just give you an idea of what this polynomial time and those algorithms which do not follow the so-called polynomial time algorithms.

2^n is certainly not polynomial. And Karmarkar's algorithm is said to have a time complexity of $O(m^{1.5} n^2 L)$, where m is the number of constraints, n is the number of variables, and L is the number of bits as input. In this case, the number of constraints is m + 1, and the number of variables is 2m + 1.

Let us do a comparison of this Karmarkar's method's $m^{1.5} n^2$ and the simplex method. Suppose it takes one second to compute the answer in the simplex method when m = 14. That means 2^{14} iterations take 1 second. 2^{15} iterations will take 2 seconds, 2^{16} will take 4 seconds; all that is fine. How many seconds will 2^{28} iterations take? It will actually take 2^{14} seconds. 2^{14} is 16384 seconds. One hour is 3600 seconds.

This is slightly less, but about 4.5 hours. When m = 14, we assume that it completes in one second. The computer is fast; it computes in one second. But suppose you give m = 28, the simplex algorithm to converge to the answer takes 4.5 hours. It is so slow. I do not have to explain that anymore. This is about the simplex algorithm. Now, suppose we take Karmarkar's algorithm in this case. In Karmarkar's algorithm, of course, the number of iterations is less, but let us say there also it takes one second; maybe it is slightly less than one second, but let us say it takes one second. So in Karmarkar's algorithm, for m = 14, it takes one second. Now, what happens to m = 28?

This was O(m power, so m is the number of constraints, so that is 15 power. So it is $15^{1.5}$, and n was 2m + 1, which is 29, 29^2 into L. For 28, it will be $O(29^{1.5}$, and 57^2 L).

29 by 15 is roughly 2.

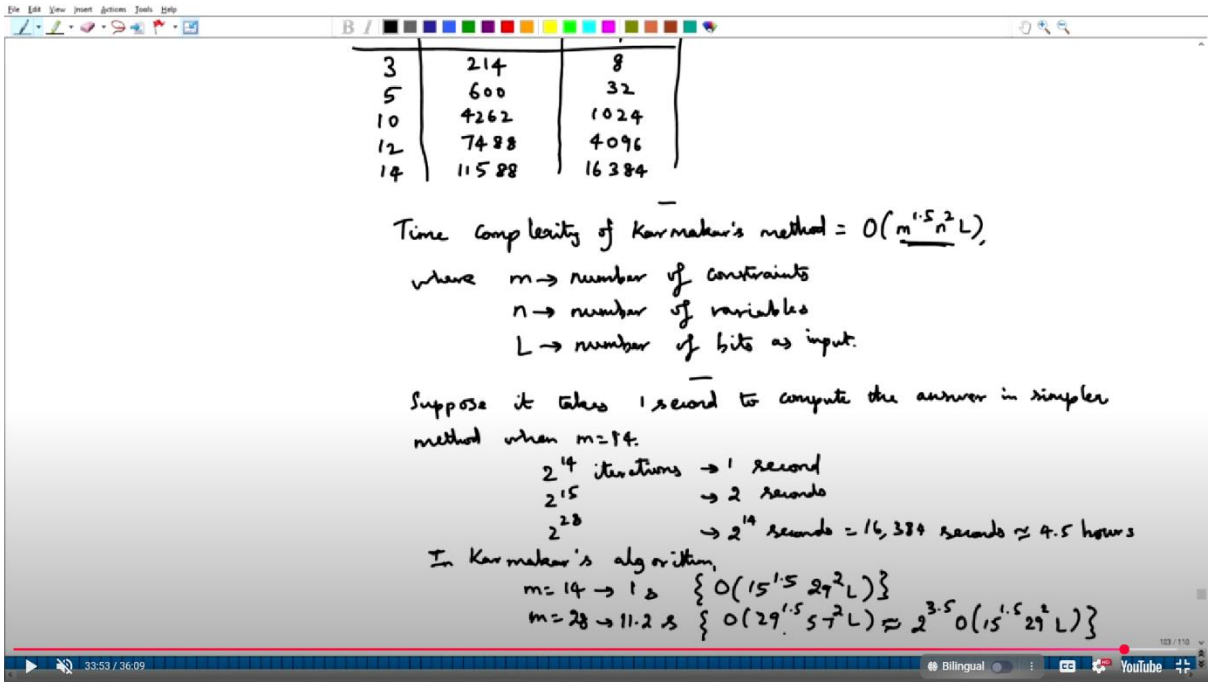
I had slightly less, but it is ok. I will write it as approximately. So it is $2^{1.5}$, and here also it is roughly 2. So it is $2^{3.5}$ $O(15^{1.5}\ 29^2$ into L).

What is $2^{3.5}$? $2^{3.5} = 8 \cdot \sqrt{2}$, which is about 1.4, so about 11.2 is what I am getting. Maybe I will write just s; I think it is fine.

What this tells us is that when you use Karmarkar's algorithm for large values of m, whatever we work out in class is like some small problems. When you work out large problems like what we worked out in this example, Karmarkar's algorithm performs much, much faster than the simplex algorithm. We just saw this. In the simplex algorithm, when m = 28, it is taking 4.5 hours, whereas in Karmarkar's algorithm, it just takes 11.2 seconds.

When m is large, you can see that Karmarkar's algorithm performs much better than the simplex algorithm that we see here.

(Refer Slide Time 33:52)



That is the reason people use Karmarkar's algorithm when at least for cases when the problems are large. Of course, the simplex method is still in use because for small problems, we can still find an exact answer when you use the simplex method. But I suppose Karmarkar's algorithm gives a better algorithm in terms of speed when you have large problems.

We will end our discussion on linear programming with this. Recall that we have discussed the simplex method, affine scaling, and Karmarkar's algorithm. In the next week, we will discuss an application. We have discussed so many algorithms now. I will explain some application and then discuss in detail how you apply these algorithms in that application in the last week of this course. Thank you.