

Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Integer Linear Program is NP-Complete

(Refer Slide Time: 0:16)

INTEGER LINEAR PROGRAM (ILP)



We will first see what is a linear Program.

Linear Programming

Find x_1, x_2, \dots, x_n that

maximizes $\sum_{i=1}^n c_i x_i$

subject to $a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$
 $a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$
 \vdots
 $a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$
 $x_i \geq 0 \quad \forall i$



Here a_{ij}, b_j, c_i are given as part of the LP instance.

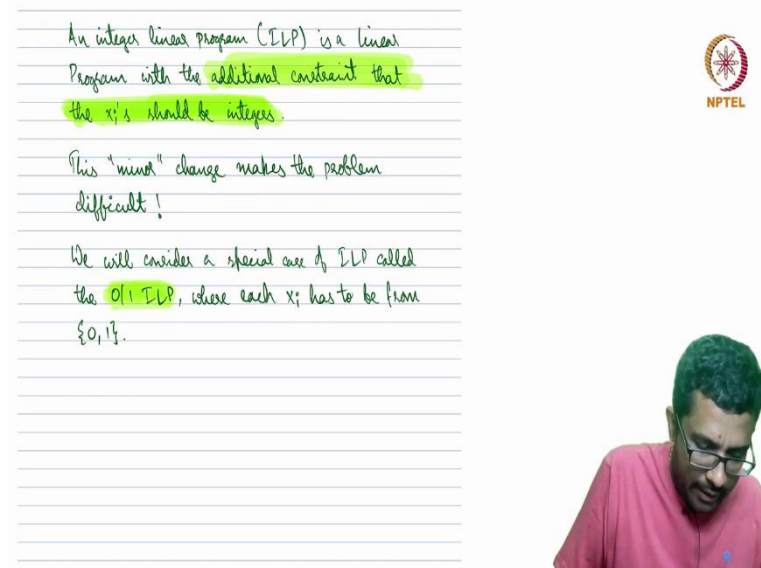
- Formalized during second world war, to plan expenditures of the army.
- Many applications - food, transportation, scheduling etc.

→ LP can be solved efficiently. The decision version is in P.

Dantzig, Khachiyan, Karumakar are some who have proposed algorithms.

An integer linear program (ILP) is a linear Program with the additional constraint that



Hello and welcome to lecture 56 of the course Theory of Computation. In this lecture, we will discuss another NP-complete problem called Integer Linear Programming (ILP). Before delving into ILP, let's understand Linear Programming (LP), a well-known technique with numerous applications.

Linear Programming involves maximizing an objective function subject to constraints. You choose variables x_1 to x_n to maximize the objective function, given constraints expressed as linear inequalities or equations. The constraints look like $a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$, $a_{21}x_1 + a_{22}x_2 \leq b_2$ and so on up to $a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$.

In matrix form, this can be represented as $Ax \leq b$, where A is an $m \times n$ matrix, x is an $n \times 1$ vector of variables, and b is an $m \times 1$ vector. The objective function to maximize is $\sum C_i x_i$, which can be written as the dot product $C \cdot x$, where C is a vector of coefficients.

Given A , b , and C , the goal is to find the values of x_1 to x_n that maximize the objective function while satisfying the constraints. Linear Programming was formalized during World War II, and it is used to optimize resource allocation under constraints.

For example, in wartime scenarios, there are constraints such as limited supplies and risks, but you want to maximize some benefit, like transporting food efficiently while avoiding enemy detection. Linear Programming has many applications, including diet optimization. For instance, to create balanced cat food, you need to ensure the right proportions of nutrients, such as protein and carbohydrates, within upper and lower limits.

Now, let's move on to Integer Linear Programming (ILP). ILP is similar to LP but with the additional constraint that some or all variables are required to be integers. This makes the problem more complex and is the focus of our discussion on NP-completeness.

To show that ILP is NP-complete, we need to prove two things: ILP is in NP: Given a proposed solution, we can verify in polynomial time whether it satisfies the constraints and optimizes the objective function. An NP-complete problem reduces to ILP in polynomial time: We can take a known NP-complete problem and transform it into an ILP problem.

For example, reducing the subset sum problem to ILP involves creating a system of linear inequalities that represent the subset sum instance. By demonstrating this reduction, we establish the NP-hardness of ILP, and by verifying solutions efficiently, we show that ILP is in NP.

At the same time, we have, let's say, 10 ingredients. The first ingredient provides some nutrients in certain quantities, the second ingredient provides other nutrients in other quantities, and so on. How much of these ingredients should I include so that all nutrients are in balanced quantities? Additionally, I want to optimize the cost since each ingredient has a certain cost. The goal is to select the ingredients to minimize the overall cost without compromising the nutritional limits.

In transportation, such as for an airline company, you want to optimize how your fleet operates. This involves scheduling constraints. Any real-world application with multiple constraints that aims to meet specific goals can often be formulated as a linear program. A linear program involves linear constraints and an objective function that you want to maximize or minimize.

For example, we want to maximize the sum of $C_i x_i$, subject to the constraints $Ax \leq b$. Sometimes, there are minimization versions. Linear programming can be solved efficiently in polynomial time using algorithms like the simplex method, developed by George Dantzig, which usually runs in polynomial time but isn't guaranteed. Later, Khachiyan and Narendra Karmarkar developed algorithms that are polynomial time.

However, if we add the constraint that all x_i must be integers, the problem becomes hard. In linear programming without the integer constraint, if a constraint isn't met, you can gradually

adjust x_1 or x_2 to meet it, making fine-tuning possible. Minor adjustments don't significantly change the outcome, allowing for smooth optimization.

When x_i must be integers, you lose this flexibility. If x_1 is 3, you can only move to 4 or down to 2, losing the smooth range for adjustments. This constraint makes the problem harder, not simpler, by reducing flexibility and making it more constrained.

Integer Linear Programming (ILP) involves the same setup as linear programming but with the additional constraint that variables must be integers. This small change transforms the problem from being solvable in polynomial time to being NP-complete, showing how a minor adjustment can significantly impact problem complexity.

So, this convey, this additional constraint changes the whole situation. And we will see that this additional constraint makes it NP-complete. So NP-complete means we generally believe it is not possible to solve it efficiently. Whereas without the integer constraint, the problem was in P. So, in fact, what we will see is a special case of integer linear program called 0-1 ILP. So in fact, here, what I am saying is I am not even saying that it should be integers, the x_i 's should be integers, I am saying that the x_i 's should be 0, 1. x_i 's should be 0 or 1.



(Refer Slide Time: 11:03)

decision version is in P.
Dantzig, Khachiyan, Kosmider are
some who have proposed algorithms.

An integer linear program (ILP) is a linear
Program with the additional constraint that
the x_i 's should be integers.

This "minor" change makes the problem
difficult!

We will consider a special case of ILP called
the 0-1 ILP, where each x_i has to be from
 $\{0, 1\}$.



Linear Programming

Find x_1, x_2, \dots, x_n that

$$\text{maximizes } \sum_{i=1}^n c_i x_i \geq t$$

subject to $a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$
 $a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$
 \vdots
 $a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$
 $x_i \geq 0 \quad \forall i$

Here a_{ij}, b_j, c_i are given as part of the LP instance.

→ Formalized during second world war, to plan circumstances of the army.



So, let me just state the decision version of the ILP, 0-1 integer linear program. So ILP is just short for integer linear program. So, the decision version of the general linear programming is that instead of asking to maximise or minimise, I already said you could ask is there a C, is there an x_i ' such that summation of $C_i X_i$ greater than t, in which case it becomes a yet another constraint. So, basically, you want to find some x_1, x_2, \dots, x_n which meet some set of constraints.

(Refer Slide Time: 11:38)

The decision version of 0/1 ILP is the following.

Does there exist $x_1, x_2, \dots, x_n \in \{0, 1\}$ such that

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_i \in \{0, 1\} \quad \forall i$$

Theorem: The decision version of 0/1 ILP is NP complete.



$$a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b$$

$$x_i \in \{0,1\} \quad \forall i$$



Thorem: The decision version of 0/1 ILP is NP complete.

Proof: (1) 0/1 ILP \in NP : Guess and verify.

(2) SUBSET-SUM \leq_p 0/1 ILP.

Given a SUBSET-SUM instance $S = \{s_1, s_2, \dots, s_k\}$ and target sum t , we can encode it into a 0/1 ILP as follows.

Does there exist x_1, x_2, \dots, x_k such that

Does there exist x_1, x_2, \dots, x_k such that

$$s_1x_1 + s_2x_2 + \dots + s_kx_k \leq t \quad \text{--- (1)}$$

$$-s_1x_1 + (s_2)x_2 + \dots + (s_k)x_k \leq -t \quad \text{--- (2)}$$

$$x_i \in \{0,1\} \quad \forall i.$$

It is easy to see that the inequality (2) is equivalent to the following

$$s_1x_1 + s_2x_2 + \dots + s_kx_k \geq t \quad \text{--- (3)}$$

$$(1) \& (3) \Rightarrow \sum_{i=1}^k s_i x_i = t$$

When $x_i \in \{0,1\}$, this is the same as asking if there is a subset of S that sums to t , i.e., it is the SUBSET-SUM problem.

$$x_i \in \{0,1\} \quad \forall i.$$

It is easy to see that the inequality (2) is equivalent to the following

$$s_1x_1 + s_2x_2 + \dots + s_kx_k \geq t \quad \text{--- (3)}$$

$$(1) \& (3) \Rightarrow \sum_{i=1}^k s_i x_i = t$$

When $x_i \in \{0,1\}$, this is the same as asking if there is a subset of S that sums to t , which is the SUBSET-SUM problem.

Thus 0/1 ILP is NP-complete.



Similarly, the decision version of the 0-1 integer linear program (0-1 ILP) is given matrices A and B. We need to determine if there exist x_1 to x_n in $\{0,1\}$ such that the constraints $Ax \leq b$ are satisfied. If the range is integers, it becomes general ILP.

Now, let's see that this problem is NP-complete. The decision version of 0-1 ILP is NP-complete. It is in NP because we can guess and verify: guess which x_i are 0 or 1, and verify whether they meet the constraints.

To show NP-hardness, we reduce the subset sum problem to 0-1 ILP. Given a set S and a target sum t, we want to build a 0-1 ILP instance such that the subset sum instance is a yes instance if and only if the 0-1 ILP instance is a yes instance.

Given a set S and a target sum t, we construct a 0-1 ILP as follows:

1. The first constraint is $\sum_{i=1}^k S_i x_i \leq t$.
2. The second constraint is $-\sum_{i=1}^k S_i x_i \leq -t$.

We also want all x_i to be in $\{0,1\}$. This setup ensures that if x_i are chosen such that $\sum_{i=1}^k S_i x_i = t$, both constraints will be satisfied.

Let's see why this works:

1. Given a subset sum instance, we write down the ILP with constraints $\sum_{i=1}^k S_i x_i \leq t$ and $-\sum_{i=1}^k S_i x_i \leq -t$.
2. Multiplying the second constraint by -1, we get $\sum_{i=1}^k S_i x_i \geq t$.
3. Combining $\sum_{i=1}^k S_i x_i \leq t$ and $\sum_{i=1}^k S_i x_i \geq t$ we get $\sum_{i=1}^k S_i x_i = t$.

This ensures that the sum of the selected S_i equals t. The x_i being 0 or 1 effectively picks some S_i values to include in the sum. This is equivalent to the subset sum problem, where you want to determine if there is a subset of S that sums to t.

Writing this instance of 0-1 ILP from the subset sum problem is straightforward and can be done in polynomial time. The correspondence between the subset sum instance and the 0-1 ILP instance is clear: the 0-1 ILP is a yes instance if and only if the subset sum problem has a subset summing to t.

(Refer Slide Time: 18:39)

INTEGER LINEAR PROGRAM (ILP)

We will first see what is a linear Program.

Linear Programming

Find x_1, x_2, \dots, x_n that

$$\text{maximizes } \sum_{i=1}^n c_i x_i$$

$$\text{subject to } a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

$$Ax \leq b$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_i \geq 0 \quad \forall i$$

→ LP can be solved efficiently. The decision version is in P.

Dantzig, Khachiyan, Karmarak are some who have proposed algorithms.

An integer linear program (ILP) is a linear Program with the additional constraint that the x_i 's should be integers.

This "minor" change makes the problem difficult!

We will consider a special case of ILP called the 0/1 ILP, where each x_i has to be from $\{0, 1\}$.

The decision version of 0/1 ILP is the following.

Does there exist $x_1, x_2, \dots, x_n \in \{0, 1\}$ such that

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

⋮

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_i \in \{0, 1\} \quad \forall i$$

Theorem: The decision version of 0/1 ILP is NP complete.

Proof: (1) 0/1 ILP ∈ NP : Given a set of variables



$$a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b$$

$$x_i \in \{0,1\} \quad \forall i$$



Theorem: The decision version of 0/1 ILP is NP complete.

Proof: (1) 0/1 ILP \in NP : Guess and verify.

(2) SUBSET-SUM \leq_p 0/1 ILP.

Given a SUBSET-SUM instance $S = \{s_1, s_2, \dots, s_k\}$ and target sum t , we can encode it into a 0/1 ILP as follows.



Does there exist x_1, x_2, \dots, x_k such that

Proof: (1) 0/1 ILP \in NP : Guess and verify.

(2) SUBSET-SUM \leq_p 0/1 ILP.

Given a SUBSET-SUM instance $S = \{s_1, s_2, \dots, s_k\}$ and target sum t , we can encode it into a 0/1 ILP as follows.



Does there exist x_1, x_2, \dots, x_k such that

$$s_1x_1 + s_2x_2 + \dots + s_kx_k \leq t \quad \text{--- (1)}$$

$$-s_1x_1 + (-s_2)x_2 + \dots + (-s_k)x_k \leq -t \quad \text{--- (2)}$$

$$x_i \in \{0,1\} \quad \forall i.$$



It is easy to see that the inequality (2) is equivalent to the following

It is easy to see that the inequality (2) is equivalent to the following

$$s_1x_1 + s_2x_2 + \dots + s_kx_k \geq t \quad \text{--- (3)}$$

$$(1) \& (3) \Rightarrow \sum_{i=1}^k s_i x_i = t$$

When $x_i \in \{0,1\}$, this is the same as asking if there is a subset of S that sums to t , which is the SUBSET-SUM problem.

Thus 0/1 ILP is NP-complete.



So, we talked about linear programming, which involves linear constraints and inequalities. Subject to these linear inequalities, we want to maximize a linear objective function by selecting appropriate values for x . Given a matrix A , vector b , and vector C , we want to find the x values that maximize the objective function while satisfying the constraints.

Then we discussed integer linear programming (ILP), where we added the restriction that the x_i 's must be integers. This additional constraint makes the problem harder. We specifically looked at the 0-1 integer linear program (0-1 ILP), a special case of ILP where the x_i 's can only be 0 or 1.

We showed that 0-1 ILP is NP-complete. Membership in NP is easy to see because we can guess and verify which x_i 's are 0 or 1 and check if they satisfy the constraints. We demonstrated the NP-hardness by reducing the subset sum problem to the 0-1 ILP problem. Given a subset sum instance, we can easily construct a 0-1 ILP instance, which is a yes instance if and only if the subset sum instance is a yes instance.

Thus, the reduction from subset sum to 0-1 ILP is straightforward, and it shows that 0-1 ILP is NP-complete. This completes our proof that 0-1 integer linear programming is NP-complete.

(Refer Slide Time: 19:50)



$$\textcircled{1} \& \textcircled{2} \Rightarrow \sum_{i \in I} s_i x_i = t$$

When $x_i \in \{0, 1\}$, this is the same as asking if there is a subset of S that sums to t , which is the SUBSET-SUM problem.

Thus 0-1 ILP is NP-complete.

This completes time complexity.

Next week - space complexity.



INTEGER LINEAR PROGRAM (ILP)

We will first see what is a linear Program.

Linear Programming

Find x_1, x_2, \dots, x_n that

$$\text{maximizes } \sum_{i=1}^n c_i x_i$$

$$\text{subject to } a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

$$Ax \leq b$$

\vdots

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_i \geq 0 \quad \forall i$$



And this also completes the time complexity part of the course, which is chapter 7 in the textbook. This also concludes week 11 of our course. In week 11, we explored many NP-complete problems. We saw that the clique problem is NP-complete, the vertex cover problem is NP-complete, and the Hamiltonian path problem is NP-complete. These problems are based on graphs.

We also mentioned a couple of other problems but did not provide detailed proofs, leaving them as exercises. We demonstrated that the subset sum problem is NP-complete, which involves determining if there is a subset of a set that sums to a target sum t . We then showed that the knapsack problem is NP-complete. Finally, we saw that the 0-1 integer linear programming problem is NP-complete.

One fascinating aspect of integer linear programming is that linear programming without the integer constraint is polynomial-time solvable. However, when the constraint is added that the numbers must be integers, the problem becomes NP-complete.

This completes the time complexity chapter. Next week, in week 12, we will begin an overview of space complexity. Just as we studied time complexity to understand the time required to solve or decide on a problem, space complexity focuses on the amount of memory needed to answer computational questions. This will be the focus for week 12.

That's all from me for week 11 and lecture 56. See you next week. Thank you.