Integer programming

Math 482, Lecture 32

Misha Lavrov

April 24, 2020

Integer linear programming

Definition

An *integer linear program* is a linear program in which some or all of the variables are constrained to have integer values only.

Integer linear programming

Definition

An *integer linear program* is a linear program in which some or all of the variables are constrained to have integer values only.

- Earlier in this class: bipartite matching.

  This is an integer program, but total unimodularity saved us and guaranteed integer optimal solutions.

Integer linear programming

Definition

An *integer linear program* is a linear program in which some or all of the variables are constrained to have integer values only.

- Earlier in this class: bipartite matching.

  This is an integer program, but total unimodularity saved us and guaranteed integer optimal solutions.

- Total unimodularity is important in integer programming, but doesn't often happen: usually, the integrality matters.

Integer linear programming

Definition

An *integer linear program* is a linear program in which some or all of the variables are constrained to have integer values only.

- Earlier in this class: bipartite matching.

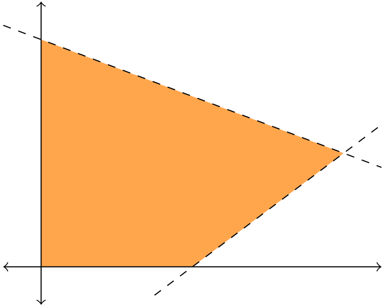
  This is an integer program, but total unimodularity saved us and guaranteed integer optimal solutions.

- Total unimodularity is important in integer programming, but doesn't often happen: usually, the integrality matters.

Some examples

Here is a completely ordinary linear program:

$$\begin{aligned} \underset{x,y\in\mathbb{R}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned}$$

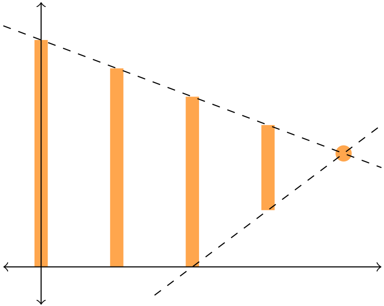


The optimal solution is $(x, y) = (4, \frac{3}{2})$.

Some examples

Now, change x to an integer variable:

$$\begin{aligned} \underset{x \in \mathbb{Z}, y \in \mathbb{R}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned}$$

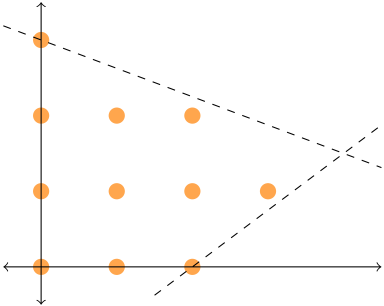


The optimal solution is still $(x, y) = (4, \frac{3}{2})$. Coincidentally, the integrality didn't matter.

Some examples

Now, make x and y both integers:

$$\begin{aligned} \underset{x,y\in\mathbb{Z}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned}$$

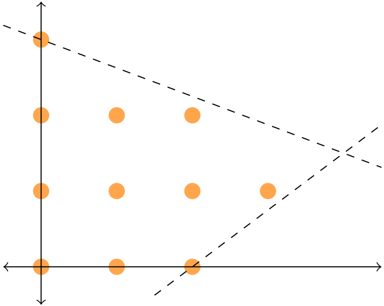


The optimal solutions are $(x, y) = (2, 2)$ and $(x, y) = (3, 1)$.

Some examples

Now, make x and y both integers:

$$\begin{aligned} \underset{x,y\in\mathbb{Z}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned}$$



The optimal solutions are $(x, y) = (2, 2)$ and $(x, y) = (3, 1)$.

Note that rounding $(4, \frac{3}{2})$ to the nearest integer won't give us an optimal or even feasible solution!

Difficulty of approximation

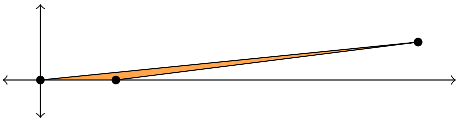
Optimal integer solutions can be arbitrarily far from optimal real solutions. Example: take the region

$$\left\{(x, y) \in \mathbb{R} : \frac{x-1}{998} \leq y \leq \frac{x}{1000}, x \geq 0, y \geq 0\right\}.$$

Difficulty of approximation

Optimal integer solutions can be arbitrarily far from optimal real solutions. Example: take the region

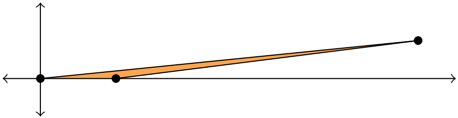
$$\left\{(x, y) \in \mathbb{R} : \frac{x-1}{998} \leq y \leq \frac{x}{1000}, x \geq 0, y \geq 0\right\}.$$



Difficulty of approximation

Optimal integer solutions can be arbitrarily far from optimal real solutions. Example: take the region

$$\left\{(x,y) \in \mathbb{R} : \frac{x-1}{998} \leq y \leq \frac{x}{1000}, x \geq 0, y \geq 0\right\}.$$

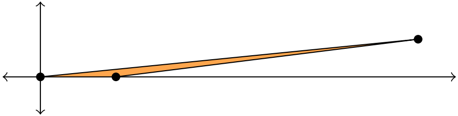


This has a vertex at $(x,y) = (500, \frac{1}{2})$. But the only integer points are $(0,0)$ and $(1,0)$.

Difficulty of approximation

Optimal integer solutions can be arbitrarily far from optimal real solutions. Example: take the region

$$\left\{(x, y) \in \mathbb{R} : \frac{x-1}{998} \leq y \leq \frac{x}{1000}, x \geq 0, y \geq 0\right\}.$$



This has a vertex at $(x, y) = (500, \frac{1}{2})$. But the only integer points are $(0, 0)$ and $(1, 0)$.

Even determining if a region contains *any* integer points can be difficult.

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**.

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.
- X_1 **OR** X_2 = **TRUE** when at least one of X_1, X_2 is **TRUE**, and **FALSE** otherwise.

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.
- X_1 **OR** X_2 = **TRUE** when at least one of X_1, X_2 is **TRUE**, and **FALSE** otherwise.
- **NOT**(**TRUE**) = **FALSE** and **NOT**(**FALSE**) = **TRUE**.

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.
- X_1 **OR** X_2 = **TRUE** when at least one of X_1, X_2 is **TRUE**, and **FALSE** otherwise.
- **NOT**(**TRUE**) = **FALSE** and **NOT**(**FALSE**) = **TRUE**.

We can use these to express logic puzzles such as Sudoku,

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.
- X_1 **OR** X_2 = **TRUE** when at least one of X_1, X_2 is **TRUE**, and **FALSE** otherwise.
- **NOT**(**TRUE**) = **FALSE** and **NOT**(**FALSE**) = **TRUE**.

We can use these to express logic puzzles such as Sudoku, but also combinatorial problems such as bipartite matching,

Logical constraints

Logical expressions have Boolean variables with values **TRUE** and **FALSE**. They are combined with logical operations:

- X_1 **AND** X_2 = **TRUE** when $X_1 = X_2 =$ **TRUE**, and **FALSE** otherwise.
- X_1 **OR** X_2 = **TRUE** when at least one of X_1, X_2 is **TRUE**, and **FALSE** otherwise.
- **NOT**(**TRUE**) = **FALSE** and **NOT**(**FALSE**) = **TRUE**.

We can use these to express logic puzzles such as Sudoku, but also combinatorial problems such as bipartite matching, graph coloring, and more.

Boolean satisfiability

Boolean satisfiability: the problem of determining if we can assign variables to Boolean variables $X_1, \ldots, X_n$ to make a logical expression true.

Boolean satisfiability

Boolean satisfiability: the problem of determining if we can assign variables to Boolean variables $X_1, \ldots, X_n$ to make a logical expression true.

(Example: does this Sudoku have a solution? Does this graph have a matching that covers all the vertices?)

Boolean satisfiability

Boolean satisfiability: the problem of determining if we can assign variables to Boolean variables $X_1, \ldots, X_n$ to make a logical expression true.

(Example: does this Sudoku have a solution? Does this graph have a matching that covers all the vertices?)

This is

- very hard: we can solve the problem by checking all 2^n assignments of $(X_1, \ldots, X_n)$, but we don't even know if there's an algorithm that takes $O(1.999^n)$ steps.

Boolean satisfiability

Boolean satisfiability: the problem of determining if we can assign variables to Boolean variables $X_1, \dots, X_n$ to make a logical expression true.

(Example: does this Sudoku have a solution? Does this graph have a matching that covers all the vertices?)

This is

- very hard: we can solve the problem by checking all 2^n assignments of $(X_1, \dots, X_n)$, but we don't even know if there's an algorithm that takes $O(1.999^n)$ steps.
- very important: if we have good heuristics for it, lots of real-life problems become easier to attack.

Boolean satisfiability and integer programming

Encode each Boolean variable X_i by an integer variable x_i with $0 \leq x_i \leq 1$: $X_i =$ **TRUE** corresponds to $x_i = 1$ and $X_i =$ **FALSE** corresponds to $x_i = 0$.

Boolean satisfiability and integer programming

Encode each Boolean variable X_i by an integer variable x_i with $0 \leq x_i \leq 1$: $X_i = \textbf{TRUE}$ corresponds to $x_i = 1$ and $X_i = \textbf{FALSE}$ corresponds to $x_i = 0$.

Then X_1 **OR** X_2 **OR** … **OR** X_k is equivalent to an inequality:

$$x_1 + x_2 + \cdots + x_k \geq 1.$$

We can write $\textbf{NOT}(X_i)$ as $(1 - x_i)$.

Boolean satisfiability and integer programming

Encode each Boolean variable X_i by an integer variable x_i with $0 \leq x_i \leq 1$: $X_i =$ **TRUE** corresponds to $x_i = 1$ and $X_i =$ **FALSE** corresponds to $x_i = 0$.

Then X_1 **OR** X_2 **OR** ... **OR** X_k is equivalent to an inequality:

$$x_1 + x_2 + \cdots + x_k \geq 1.$$

We can write **NOT**(X_i) as $(1 - x_i)$.

So a system of inequalities can represent a logical expression in "conjunctive normal form": an **AND** of **OR**s.

Boolean satisfiability and integer programming

Encode each Boolean variable X_i by an integer variable x_i with $0 \leq x_i \leq 1$: $X_i =$ **TRUE** corresponds to $x_i = 1$ and $X_i =$ **FALSE** corresponds to $x_i = 0$.

Then X_1 **OR** X_2 **OR** . . . **OR** X_k is equivalent to an inequality:

$$x_1 + x_2 + \cdots + x_k \geq 1.$$

We can write **NOT**(X_i) as $(1 - x_i)$.

So a system of inequalities can represent a logical expression in "conjunctive normal form": an **AND** of **OR**s.

Fact: all logical expressions can be put in this form. So integer programming can model all Boolean satisfiability problems!

Fixed costs

We can get additional power by mixing logical expressions with linear constraints.

Fixed costs

We can get additional power by mixing logical expressions with linear constraints.

Example 1: Fixed costs

A banana factory wants to ship bananas to grocery stores Illinois. It can rent a warehouse in Colorado, but this doesn't add a per-banana price: it costs \$1000, no matter how many bananas are stored.

Fixed costs

We can get additional power by mixing logical expressions with linear constraints.

Example 1: Fixed costs

A banana factory wants to ship bananas to grocery stores Illinois. It can rent a warehouse in Colorado, but this doesn't add a per-banana price: it costs \$1000, no matter how many bananas are stored.

- Add a variable $w \in \mathbb{Z}$ with $0 \leq w \leq 1$, represented a warehouse rental by $w = 1$.

Fixed costs

We can get additional power by mixing logical expressions with linear constraints.

Example 1: Fixed costs

A banana factory wants to ship bananas to grocery stores Illinois. It can rent a warehouse in Colorado, but this doesn't add a per-banana price: it costs \$1000, no matter how many bananas are stored.

- Add a variable $w \in \mathbb{Z}$ with $0 \leq w \leq 1$, represented a warehouse rental by $w = 1$.
- Cost in the objective function $1000w$.

Fixed costs

We can get additional power by mixing logical expressions with linear constraints.

Example 1: Fixed costs

A banana factory wants to ship bananas to grocery stores Illinois. It can rent a warehouse in Colorado, but this doesn't add a per-banana price: it costs \$1000, no matter how many bananas are stored.

- Add a variable $w \in \mathbb{Z}$ with $0 \leq w \leq 1$, represented a warehouse rental by $w = 1$.
- Cost in the objective function $1000w$.
- We can write other constraints in terms of w when they depend on the existence of a warehouse.

Combining constraints with Boolean variables

Example 2: Conditional constraints

The warehouse can store up to 100 red, yellow, or green bananas—but only if it is rented. Otherwise, it can't store any bananas.

Assume $r, y, g \geq 0$ are the number of bananas stored.

Combining constraints with Boolean variables

Example 2: Conditional constraints

The warehouse can store up to 100 red, yellow, or green bananas—but only if it is rented. Otherwise, it can't store any bananas.

Assume $r, y, g \geq 0$ are the number of bananas stored.

- The unconditional constraint: $r + y + g \leq 100$.
- The conditional constraint: $r + y + g \leq 100w$.

Combining constraints with Boolean variables

Example 2: Conditional constraints

The warehouse can store up to 100 red, yellow, or green bananas—but only if it is rented. Otherwise, it can't store any bananas.

Assume $r, y, g \geq 0$ are the number of bananas stored.

- The unconditional constraint: $r + y + g \leq 100$.
- The conditional constraint: $r + y + g \leq 100w$.
- This simplifies to the unconditional constraint if $w = 1$, but forces $r = y = g = 0$ if $w = 0$.

The big-number method

Example 3: The big-number method

If a warehouse is rented in Colorado, suddenly the banana company is subject to Colorado state laws, which say it can grow at most 50 blue bananas.

The big-number method

Example 3: The big-number method

If a warehouse is rented in Colorado, suddenly the banana company is subject to Colorado state laws, which say it can grow at most 50 blue bananas.

- The unconditional constraint: $b \leq 50$.

The big-number method

Example 3: The big-number method

If a warehouse is rented in Colorado, suddenly the banana company is subject to Colorado state laws, which say it can grow at most 50 blue bananas.

- The unconditional constraint: $b \leq 50$.
- The conditional constraint: $b \leq 50 + 1000000(1 - w)$.

The big-number method

Example 3: The big-number method

If a warehouse is rented in Colorado, suddenly the banana company is subject to Colorado state laws, which say it can grow at most 50 blue bananas.

- The unconditional constraint: $b \leq 50$.
- The conditional constraint: $b \leq 50 + 1000000(1 - w)$.
- This simplifies to the unconditional constraint if $w = 1$ (if there is a warehouse), and is effectively not present if $w = 0$ (if there is no warehouse).

The big-number method

Example 3: The big-number method

If a warehouse is rented in Colorado, suddenly the banana company is subject to Colorado state laws, which say it can grow at most 50 blue bananas.

- The unconditional constraint: $b \leq 50$.
- The conditional constraint: $b \leq 50 + 1000000(1 - w)$.
- This simplifies to the unconditional constraint if $w = 1$ (if there is a warehouse), and is effectively not present if $w = 0$ (if there is no warehouse).
- This method does not always work (only if there are practical limits on b) and very large values of the big number make the linear program worse to solve.