Lecture 20: Integer Linear Programming

Sam McCauley

November 21, 2025

Williams College

Admin

- Assignment 7 posted; look over topic ideas presented there so you're ready to email me on the Monday after Thanksgiving
- Practice problems for the final out soon; graded assignments out right after
- TA feedback form posted; please fill out if you've talked to Alex this semester
- Colloquium today: Cryptography!
- Questions?

Wrapping up LPs

What makes an LP Solution Efficient?

- One question I often get: how *efficient* is your LP?
- Short answer: I'd like your LPs to have polynomial size (in the size of the input to the original problem)
- In this class, I do not think we will look at any problem which can be effectively solved with an exponential-sized LP

LPs and Real-World Solutions

- I often ask you to write a recipe to create an LP for any specific problem instance
- What happens when there is no solution to the instance?
  - The LP is *infeasible*. The solver will tell you that!
- What happens when you can achieve arbitrarily good solutions?
  - The LP is *unbounded*. The solver will tell you that too!

Integer Linear Programming

Definitions

- Integer Linear Program (ILP): has linear constraints and objectives, but all variables are required to be *integers*
- Mixed Integer Linear Program (MIP): linear constraints and objectives. Some variables are required to be integers, some variables are continuous
- Sometimes I might call something an “ILP” when it is actually an MIP. Sorry in advance. (The distinction is not particularly important in this class.)

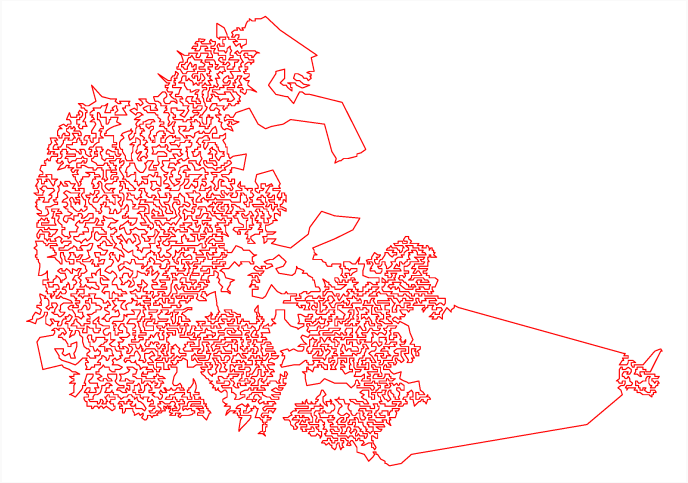
Why ILP is Useful

- Benefits from some structure (not as much as LP)
- Efficient solvers in practice
- Extremely widely applicable

Some good and bad news

- Solving an ILP or an MIP is *NP-hard*
- Bad news: this means that we can't give a guarantee to solve an ILP efficiently
- Good news: if an ILP solver tends to be efficient in practice, we can use it to solve real-life NP hard problems
- Can guarantee optimal solutions!
    - It just may take a long time to get them...

Travelling Salesman



This is an *optimal* solution for a TSP instance with tens of thousands of points; obtained using MIP-like techniques.

Travelling Salesman

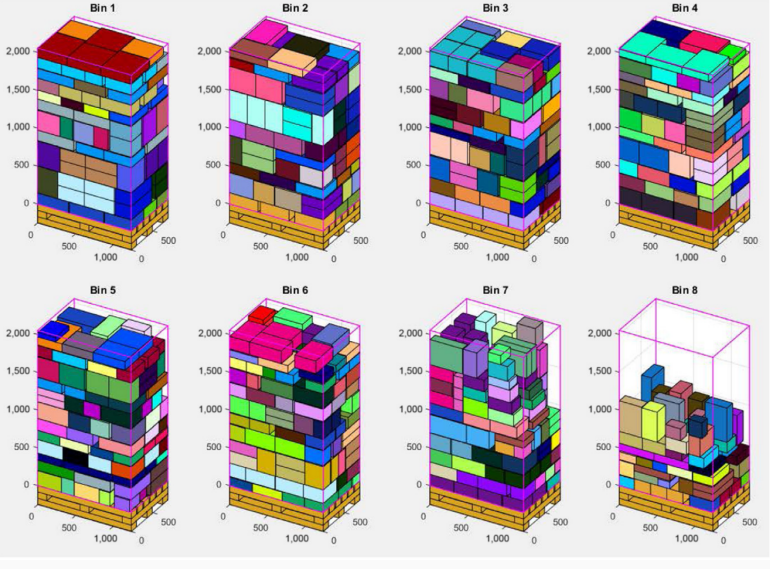
The best-known worst case solution ($O(n^2 \cdot 2^n)$ time) would not work for 100 points, even if all computers in the world were run for a century.

(Literally) Packaging Items

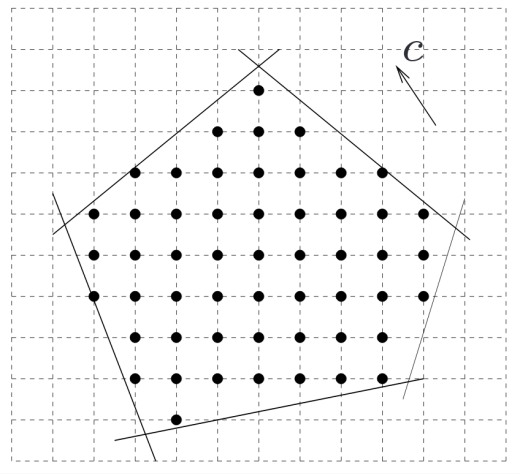
- Real-world example of something you may way to solve with an MIP:
- Pack items onto pallets (bins)
    - Items are 3 dimensional, can be rotated
    - Items may not have integral sizes
    - Constraints for what goes on top of what

Packaging Items

From Elhedhli, Gzara, and Yildiz 2019:



Visualization of an ILP



(Figure from L. Vandenberghe)

- Still a polytope given by inequalities
- But now, we're restricted to integer grid points

ILP and MIP Examples: Diet Revisited

Simple MIP

Diet problem from last lecture, but peanuts and rice only come in 100g bags. Chicken we may order as many grams as we want.

Objective: $\min 1.61p + .79r + .7c$

$$25.8p + 2.5r + 13.5c \geq 46$$
$$16.1p + 28.7r \geq 130$$
$$p \geq 0, r \geq 0, c \geq 0$$
$$p, r \in \mathbb{Z}$$

This means p and r are integers.

Using GLPK for ILP and MIP

- after “bounds” section (or after “constraints” section if no bounds)
- can write `general`, `integer`, or `binary`
- Then list variables of that type. (Binary variables must be `0` or `1`; integer must be integer; general are just normal LP variables)
- Default: general
- Let’s look at the diet problem as an ILP

ILP and MIP Examples: Two Towers

Two Towers!

- Get a list of heights (let's forget about taking square roots—it's OK if the heights are not integers) $h_1, \ldots h_n$
- Want to divide into two towers T_1 and T_2 to minimize $|\sum_{i \in T_1} h_i - \sum_{j \in T_2} h_j|$.
- How can we do this using an ILP?

Two Towers as an ILP

- Idea: build the smaller tower, make it as large as possible (but less than half total height)
- Variables $x_1, \ldots, x_n$. We have $x_i \in \{0, 1\}$ for all i. Goal: $x_i = 1$ if i is in the smaller tower
- Objective: $\max \sum_{i=1}^{n} x_i h_i$
- Constraints:

$$\sum_{i=1}^{n} x_i \, h_i \leq \frac{1}{2} \sum_{i=1}^{n} h_i$$

$$x_i \in \{0, 1\} \text{ for all } i = 1, \ldots, n$$

Why does this work?

- Every x_i is 0 or 1
- The total height of all items i with $x_i = 1$ is less than half the height (so it's the smaller tower), and is as large as possible
- So every assignment of 0 and 1 to x_i corresponds to a two tower solution. The ILP solution picks the best one.

First attempt: rounding

- Can we solve this as an LP and then round the solution?

- No! LP is trivially solvable with all but one variable being an integer.

How does GLPK do on two towers?

- It seems to give wrong answers for larger inputs (suboptimal, or even over the threshold)
- Appear to be some precision issues.
- Might not come up if we call GLPK from `C` rather than using the `CPLEX` format?
- If you want, you can have your Assignment 7 be implementing this ILP using the `C` or python interface into GLPK and investigate what's going wrong

ILP and MIP Examples: Doctor Assignments

Doctor Assignments

- Let's say we have n doctors and n hospitals
- Want to match doctors to hospitals
- Doctor i lives distance $d_{i,j}$ from hospital j
- Goal: match doctors with hospitals to minimize total driving distance
- Side question: can we solve this algorithmically without using ILP/MIP?
  - Yes, but this one generalizes easily to allow for further constraints
  - One example: two doctors are in a relationship and they need to be matched to hospitals that are within a certain distance of each other.

Doctor Assignments: ILP

- What should our variables be?
- $x_{ij} = 1$ if doctor i is assigned to hospital j, $x_{ij} = 0$ otherwise
- Constraints?
  - $x_{ij} \in \{0, 1\}$
  - For all i: $\sum_{j=1}^{n} x_{ij} = 1$ (every doctor has one hospital)
  - For all j: $\sum_{i=1}^{n} x_{ij} = 1$ (every hospital has one doctor)

Doctor Assignments: ILP

Constraints:

- $x_{ij} \in \{0, 1\}$
- For all i: $\sum_{j=1}^{n} x_{ij} = 1$
- For all j: $\sum_{i=1}^{n} x_{ij} = 1$

Objective? (Recall: goal is minimize total distance)

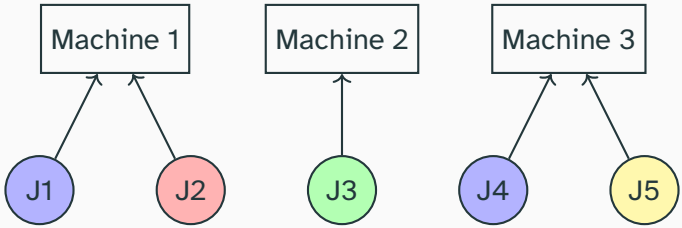
- $\min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} d_{ij}$

Scheduling With ILP

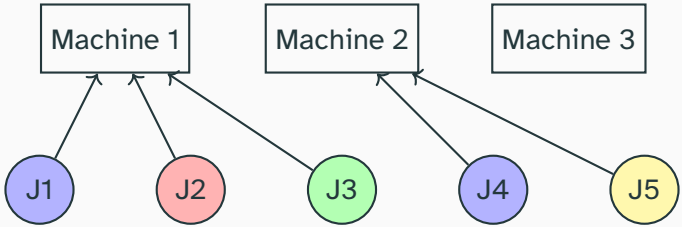
Scheduling

- (Aside: scheduling is a major application of ILPs. Lots of different techniques; this is just one simple example.)
- Assign n unit-cost jobs to machines.
- Each job j_i has a type t_i. Two jobs of the same type cannot be assigned to the same machine.
- How can we schedule the jobs with the minimum number of machines?

Scheduling



Scheduling



This solution is better. (Yes, you can solve this with a greedy approach. Our goal today is to get MIP practice.)

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?
- Probably: keep track of what job is assigned to what machine
- $s_{i,m} = 1$ if job i is assigned to machine m
- How many machines do we need?
- At most n. So have n^2 variables: $s_{i,m} \in \{0, 1\}$, for $1 \leq i \leq n$ and $1 \leq m \leq n$.

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Want every job assigned to exactly one machine
- For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Two jobs of the same type can't be assigned to the same machine
- Rephrased: for every machine m, no two jobs of the same type can be assigned to m

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- For every machine i, no two jobs of the same type can be assigned to i
- For all $1 \leq m \leq n$,for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$
- (Up to n^3 constraints. Also: constraints depend on the input.)

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?
- Let c_m be the cost of machine m. Want $c_m = 1$ if there is a job assigned to machine i, $c_m = 0$ otherwise.
- $\min \sum_{m=1}^{n} c_m$
- Constraint for c_m?
- For all jobs i and all machines m, $c_m \geq s_{i,m}$

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input
- The *computation time* is the time it takes to go from an ILP/MIP recipe to a `.lp` file
- In other words: the time to calculate all the *constants*!
- We also want this to be *polynomial* in the size of the original problem input
- I will not ask you to calculate these values. I am going over this because any ILP/MIP you give should have polynomial size and polynomial computation time.

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

What is the size of this ILP?

$n + n^2 = O(n^2)$ variables, at most $n^2 + n^3 + n = O(n^3)$ constraints. Multiplying, total size is $O(n^5)$

So the size is polynomial in n.

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

Computation time?

Polynomial. (All the constants can be calculated in $O(1)$ time.)

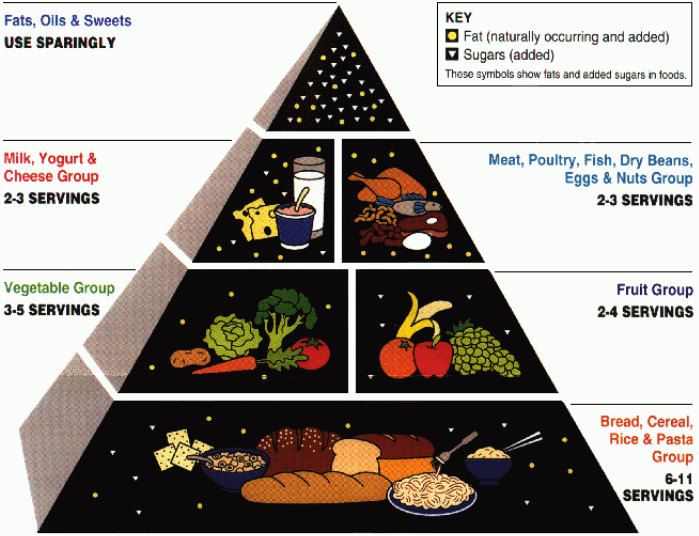
More specifically, this can be caluclated in $O(n^5)$ time.

Using ILP to Help With Choices

Revisiting Discussion from Last Class

- Idea here: we talked about how LPs can only really “AND” constraints
- With ILP and MIP, can do something much more like “OR”:
  - One of these constraints must be satisfied, or
  - Pick one of these items (in an assignment)
- Simple example: optimal eating while being able to choose your diet

Food Pyramid



Choice of diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day

- 100g Peanuts: 25.8g of protein, 16.1g carbs, $1.61

- 100g Rice: 2.5g protein, 28.7g carbs, $.79

- 100g Chicken: 13.5g protein, 0g carbs, $.70

What is the cheapest way you can hit one of these diet goals?

MIP for Choice of Diet

- How to encode which diet I choose?
- $x_1 = 1$ if I choose the first diet; $x_2 = 1$ if I choosed the second diet; $x_3 = 1$ if I choose the third diet
- Make sure I choose exactly one diet?
- $x_i \in \{0, 1\}$
- $x_1 + x_2 + x_3 = 1$

MIP for Choice of Diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- How can I encode this?
- Previously: $25.8p + 2.5r + 13.5c \geq 46$ ...
- Hint: if $x_1 = 0$, I want to do something to these constraint so that they're *always* satisfied
- $25.8p + 2.5r + 13.5c + 46(1 - x_1) \geq 46$

Choice of diet LP

- Diet options:
  - 46 g protein; 130 g carbs; or
  - 20 g protein; 200 g carbs; or
  - 100 g protein; 30 g carbs
- 100g Peanuts: 25.8g protein, 16.1g carbs, $1.61
- 100g Rice: 2.5g protein, 28.7g carbs, $.79
- 100g Chicken: 13.5g protein, 0g carbs, $.70

$\min 1.61p + .79r + .7c$

- $25.8p + 2.5r + 13.5c + 46(1 - x_1) \geq 46$;
- $16.1p + 28.7r + 130(1 - x_1) \geq 130$
- $25.8p + 2.5r + 13.5c + 20(1 - x_2) \geq 20$;
- $16.1p + 28.7r + 200(1 - x_2) \geq 200$
- $25.8p + 2.5r + 13.5c + 100(1 - x_3) \geq 100$;
- $16.1p + 28.7r + 30(1 - x_2) \geq 30$
- $x_1 + x_2 + x_3 = 1$
- $p, r, c \geq 0$; $p, r \in \mathbb{Z}$; $x_i \in \{0, 1\}$

Technique summary

- When want to choose one of several constraints to satisfy:
- Multiply the indicator variable for whether or not you choose by a large enough constant to make the constraint trivial
- Need to be able to bound the constraint to do this!
- What happens with rounding when you use this technique?