Lecture 16: Integer Linear Programming

Sam McCauley

November 5, 2024

Williams College

Admin

- Slightly updated wording posted for Homework 5
- Assignment 3 will be out Thursday; will look like Homework 5 but will have integer-constrained problems
- I'll try to grade Homework 5 very quickly so that you have time to get feedback before submitting
- Friday: lots of MIP examples; some from me and some in *group work*
- Questions?

Homework 5 Comments

- I released some updated wording for Problem 3 in Homework 5. No big changes!
  - A "week" is a 7-day period in the n days. In part (b), when you actually make the LP, there is only one week
  - When I say that half of the ℓ_j hours worked must be by employees who can work the cash register, I mean half the hours (not half the employees—I don't think you could encode that using an LP)
  - Half of the ℓ_j extra hours, *and* all of the original 8 hours, must be done by cash register employees.

Outside resources

- (ChatGPT etc. is not allowed on these assignments; it’s only allowed for code.)
- Sometimes I get the impression that it’s very tempting to use ChatGPT
- For what it’s worth: it seems very bad at giving LPs and ILPs
- It will give you a very good-looking one, and write beautiful language about why it’s correct, but it will make no sense
- It “makes no sense” in a very unique way; I don’t think it’s very good at generating a first draft (or a mediocre answer) like it often can be for simple code

Plan for Today

- Wrap up last lecture
- What is an integer linear program?
- Lots of examples!
- Some information about how *integer* linear program solvers work

One more LP

- Let’s solve a difficult optimization problem with our LP solver
- Idea: a middle school closed. Students from 6 different areas need to be assigned to the three other existing middle schools in the area. How can we do that?
- Need to assign all students; make sure students are reasonably well-balanced; minimized cost of transportation

Setting up school problem

- Three schools with capacities 900, 1100, 1000
- Grades 6, 7, and 8
- # students from each grade assigned to a school must consist of between 30% and 36% of the school's total enrollment.
- Let's look at numbers in terms of what students are from each area, and how much it costs to get students from an area to a school. Then we'll create the LP on the board, and finally input it into an LP solver.

School Problem Numbers

Area	6th	7th	8th	School 1	School 2	School 3
1	144	171	135	$300	0	$700
2	222	168	210	-	$400	$500
3	165	176	209	$600	$300	$200
4	98	140	112	$200	$500	-
5	195	170	135	0	-	$400
6	153	126	171	$500	$300	0

- Three schools with capacities 900, 1100, 1000
- # students from each grade assigned to a school must consist of between 30% and 36% of the school's total enrollment.
  - (Can't give one school all eighth graders.)
- minimize total cost

Looking at the Solution

- Let's look at the solution
- Can use `grep` to help process the text
- Powerful tool for searching in files
  - Usage: `grep [expression] [file or files]`
  - Returns all lines in the files containing the expression
  - The expression can just be a string, or can be a regular expression

Our solution is fractional!

- What can we do?
- Round up or down; make sure constraints are still met
- How much can this affect our cost?
- Each school will probably end up with ≈ 1 student away from optimal. Unlikely to be more than \$1000 or so off.
- Is this strategy a good idea for other problems?
  - Works well when rounding has a small impact on solution quality
  - Not so well when it has a large impact
  - Often cannot guarantee any optimality (we're suddenly getting a decent-looking solution rather than the best solution)

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

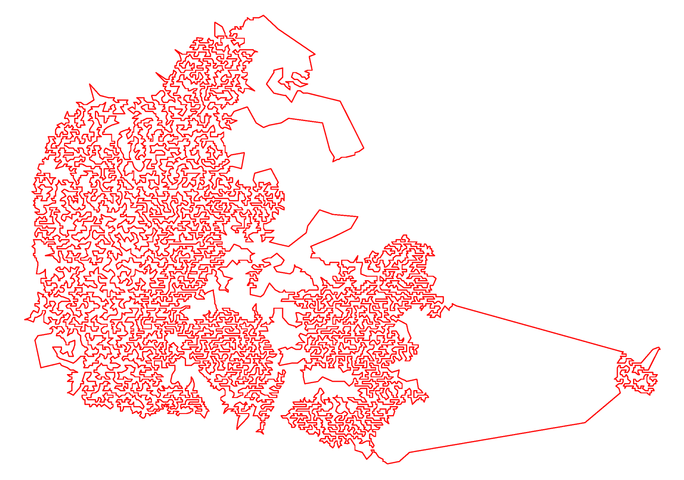
Why it’s Useful

- 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



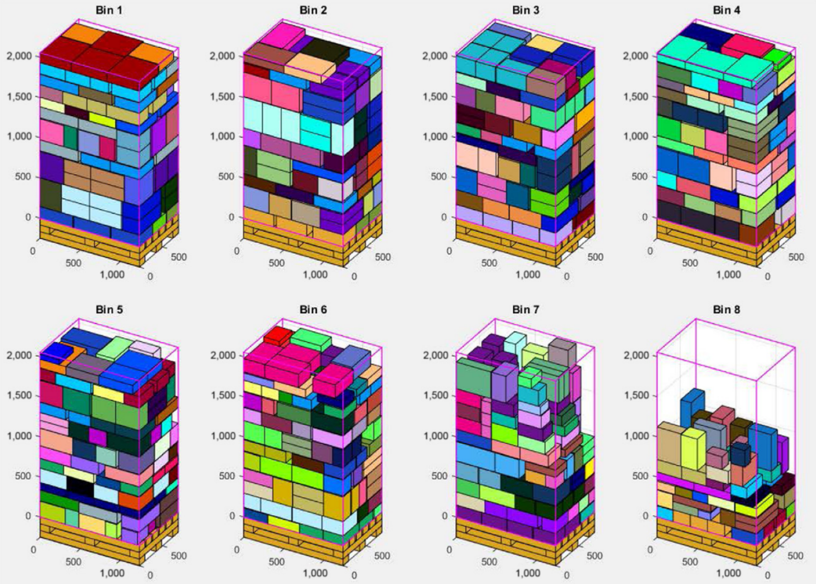
This is an *optimal* TSP instance with tens of thousands of points.

(Literally) Packaging Items

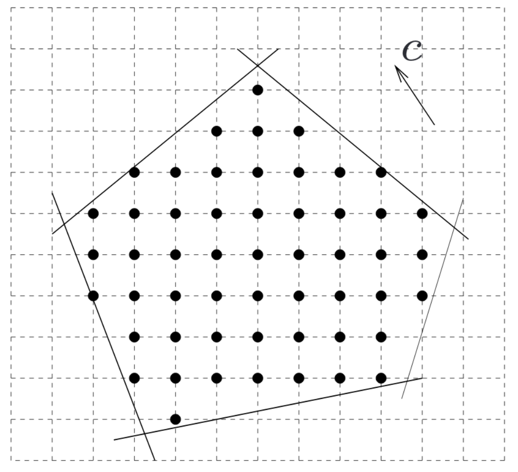
- Real-world example of something you may way to solve with an MIP:
- Pack items onto pallets (bins)
- Items are 3 dimensional, can be rotated
- Items may not have integral sizes
- Constraints for what goes on top of what

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

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, general are just normal LP variables)
- Default: general
- Let’s look at the diet problem as an ILP

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?
- (That gave a “pretty good” solution for the middle schools problem.)
- 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?
- Probably an interesting final project to look into it more and figure out the tradeoffs
  - Are there inputs it works on? Is it ever better than meet in the middle?
  - Does the extra control calling it from `C` help?

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
- Other methods?
    - Yes, but this one generalizes easily to allow for further constraints
    - Random 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?
- $x_{i,j} = 1$ if doctor i is assigned to hospital j, $x_{i,j} = 0$ otherwise
- Constraints?
  - $x_{i,j} \in \{0, 1\}$
  - For all i: $\sum_{j=1}^{n} x_{i,j} = 1$ (every doctor has one hospital)
  - For all j: $\sum_{i=1}^{n} x_{i,j} = 1$ (every hospital has one doctor)

Doctor Assignments: ILP

Constraints:

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

Objective? (Recall: goal is minimize total distance)

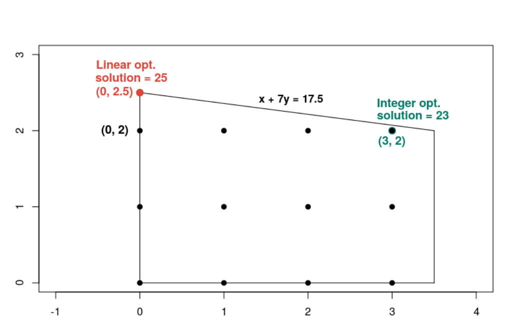
- $\min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} d_{i,j}$

Solving ILPs and MIPs

First thought: Can we Round?

- *LP relaxation:* just remove the integer constraints
- $e_{i,j} \in \{0, 1\}$ becomes $e_{i,j} \geq 0$ and $e_{i,j} \leq 1$.
- How badly can this do?

Rounding MIPs

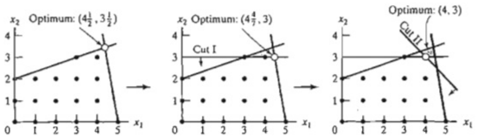


From Google OR Tools Documentation

- Can do *arbitrarily* badly, even for simple ILPs
- May work effectively if the problem has a special structure that makes rounding effective
- Example: the diet problem is probably solved fairly well by rounding (will only be off by 1 unit of each food)

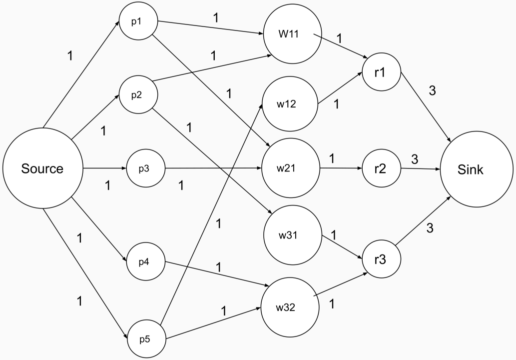
Second Method: Cutting ILPs

FIGURE 9.10
Illustration of the use of cuts in ILP



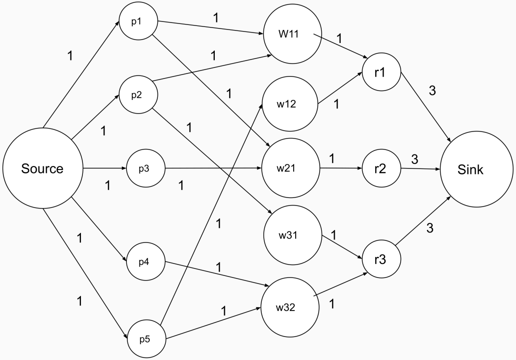
- Cut the LP without removing integer solutions
- After enough cuts, can round and get a good solution!
- Not always possible, but surprisingly effective methods in practice for some types of problem
- Many MIP solvers find these cuts for you

Third Method: Prove the LP has integral soln



- Broad class of LPs are guaranteed to give optimal solutions
- We won't go over in this class except on this slide!
- Example for linear algebra people: if your constraint matrix is totally unimodular then there exists an optimal integer solution

Third Method: Prove the LP has integral soln



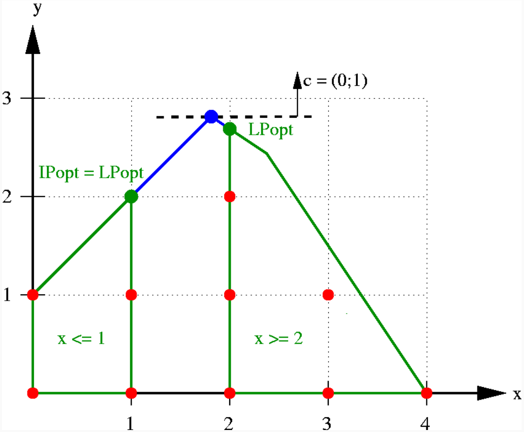
- Broad class of LPs are guaranteed to give optimal solutions
- We won't go over in this class except on this slide!
- Example for flow-reduction-lovers: if you write a flow problems as an LP where all constraints are integers, there exists an optimal integer solution

Main MIP Solving Method: Branch and Bound

Branch and Bound

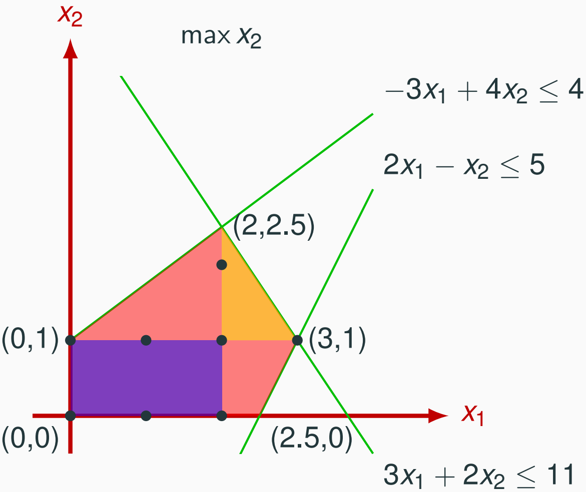
- Two towers: really wanted to “rule out” some of the search space
- Maintain worst-case guarantees
- Branch and bound: a less-problem-specific way to do the same thing
- This is a large *class* of algorithms; I’m giving a high level description of the idea

Branching



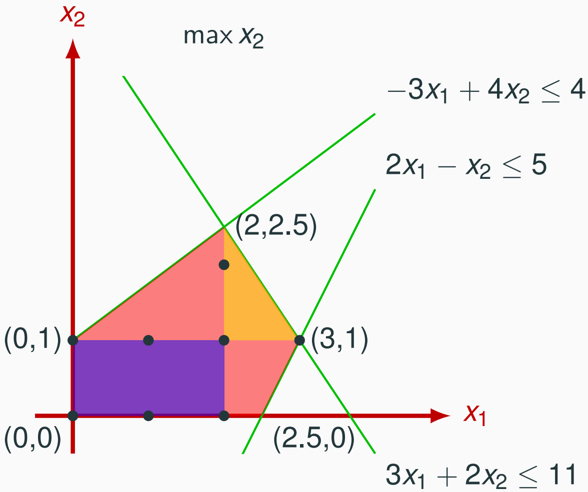
- First, we divide the problem into several subproblems
- Visualization is useful: just partition the feasible region into several pieces
- So far, still need to search through all of them (same as brute force)

Branching and Bounding



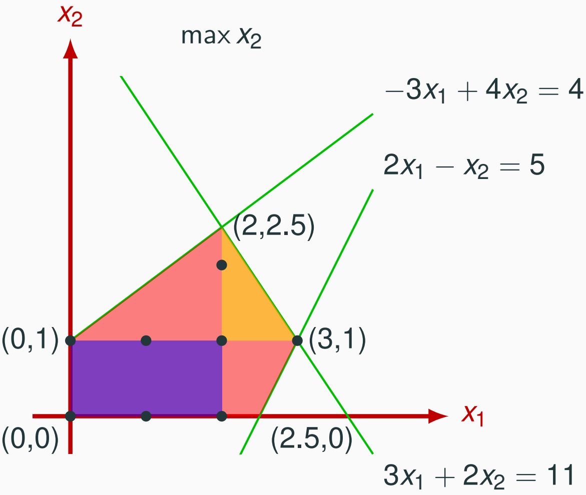
- Partition region
- Find best solution in orange piece
- When can we avoid searching in purple?

Branching and Bounding



- Upper bound best solution in purple
- If best possible soln in purple is worse than best soln in orange, can *safely skip* it

Branching and Bounding



Safe to skip: *always* still gives an optimal solution.

But, can't skip anything in worst case.

What do we need?

- Way to get a good solution in orange region: recurse!
- Base case: can just do a simple greedy method if the region is small enough.
- Way to upper bound best solution in purple region??
  - Relax to an LP! Might not give a good upper bound, but will give *an* upper bound (Recall: LPs are relatively fast to solve)
  - Duality can help (we won’t talk about in this class)

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces
- Let us rule out big parts of the search space
- "Everything in here has a bad objective function, so we can skip it." (This is the *bound* part)
- Many practical problems have large parts that are easy to skip. (If we're stacking groceries on pallets, no need to spend time looking at solutions with bread on the bottom.)
- The more we branch (find good solutions), the more we can bound (rule out parts of the search space whose solutions are suboptimal)

Branch and Bound in Practice

- Advanced methods to figure out how to split into pieces; how much to search each piece before doing more bound calculations
- The better your choices, the more you can rule out
- Other methods (greedy, LP cuts, duality, heuristic search, etc.) can be integrated into this method

Branch and Bound in Practice

- Solvers are sometimes optimized for a given problem
- Dedicated solvers for TSP, Knapsack, that make branching decisions and use bounding methods particularly effective for that problem
- This is how you get the optimal, giant TSP tours
- Also some general-purpose solvers

Branch and Bound Summary

- Always gives an optimal solution *eventually*
- May not find it quickly on tricky problems
- Can be very fast even on reasonably hard, reasonably large instances.

Solvers

These solvers have both LP and MIP solvers (using different algorithms):

- GLPK (simplex, branch and bound). Open source. Standalone program is fairly easy to use; can also access from C.
- CPLEX - IBM software for MIPs. Old but reliable. Proprietary. Effective, but can be difficult to work with
- COIN-OR - open source solver
- Google OR tools - wrapper for COIN-OR. Has a really nice TSP and Knapsack solvers. More user friendly