Lecture 15: Linear Programming Solvers

Sam McCauley

November 1, 2024

Williams College

Admin

- Assignment 2 over!
- Homework 4 back; great job!
- Homework 5 out. Last “homework” (one more assignment next week; then final project)
- Preregistration until Monday. Two particularly relevant courses:
  - Algorithmic Game Theory
  - Parallel Programming
- Questions?

In honor of last “leaderboard”

Linear Programming

A linear program consists of:

- a linear *objective function*, and
- a set of linear *constraints*.

Goal: achieve the best possible objective function value while satisfying the constraints

Solving Problems with Linear Programming

Example 3 (hard): Group Grading

- The CS TAs at Williams have decided that all TAs will help do the grading for all assignments due in a given week.
- Problem setup: they have n hour-long time slots during the week. Some time slots have more TAs available than others. Assignments will arrive as the week goes on.
- Assignments don't all take the same time to grade! In particular, there are m courses. It takes a certain amount of TA hours to grade a particular submission from a given course, and a given due date may have different numbers of arriving assignments.
- Goal: assign how many TAs should work on what course during a given hour
- Objective: minimize the *average* time it took to grade each assignment

Example 3 (hard): Group Grading

Inputs to the problem:

- Time slot i has t_i TAs available for grading
- Grading a single assignment from course j requires a total of h_j TA hours worth of time
- $w_{i,j}$ is the number of assignments from course j that arrive at time slot i
- Question: for each time slot i, how many (fractional) TAs should work on each course j to minimize the average time it takes each submission to be graded?

Example 3 (hard): Group Grading

How should we make our variables? (In other words, what does our solution look like?)

Let $x_{i,j}$ be the number of TAs working on course j in time slot i.

(It seems like we should also have variables for cost. We'll come back to that.)

Problem (Reminder)

- *t_i: TAs available at time i*
- *h_j: TA hours req. to grade an assgn. from course j*
- *$w_{i,j}$: number assignments from course j that arrive at time slot i*
- *Question: for each time slot i, how many TAs should work on each course j to minimize the average time it takes each submission to be graded?*

Example 3 (hard): Group Grading

Can we constrain $x_{i,j}$? What are the limits to how we can assign TAs?

Can't assign more TAs at time i than available: for all i, $\sum_j x_{i,j} \leq t_i$

Problem (Reminder)

- *t_i: TAs available at time i*
- *h_j: TA hours req. to grade an assgn. from course j*
- *$w_{i,j}$: number assignments from course j that arrive at time slot i*
- *$x_{i,j}$: (variable) for each time slot i, number TAs working on each course j to minimize the average time it takes each submission to be graded*

Example 3 (hard): Group Grading

How do we keep track of the work the TAs are doing? When $w_{i,j}$ arrives, if we have assignment $x_{i,j}$, how does that affect the final grading time?

First try: $x_{i,j} = w_{i,j} \cdot h_j$.

Issue: This requires *all* work that arrives at slot i to be completed at time i. Might not be possible!

Problem (Reminder)

- *t_i: TAs available at time i*
- *h_j: TA hours req. to grade an assgn. from course j*
- *$w_{i,j}$: number assignments from course j that arrive at time slot i*
- *$x_{i,j}$: (variable) for each time slot i, number TAs working on each course j to minimize the average time it takes each submission to be graded*

Example 3 (hard): Group Grading

What if we can't finish all the work in a given timeslot? We need to keep track of what spills over.

Let $r_{i,j}$ be the remaining work for course j after time slot i.

Problem (Reminder)

- t_i: *TAs available at time i*
- h_j: *TA hours req. to grade an assgn. from course j*
- $w_{i,j}$: *number assignments from course j that arrive at time slot i*
- $x_{i,j}$: *(variable) for each time slot i, number TAs working on each course j to minimize the average time it takes each submission to be graded*

Example 3 (hard): Group Grading

How much work is remaining? Well, during time slot i for course j, we assign $x_{i,j}$ TAs. This means they can do a total of $x_{i,j}$ work from course j.

Problem (Reminder)

- t_i: *TAs available at time i*
- h_j: *TA hours req. to grade an assgn. from course j*
- $w_{i,j}$: *number assignments from course j that arrive at time slot i*
- $x_{i,j}$: *(variable) for each time slot i, number TAs working on each course j to minimize the average time it takes each submission to be graded*
- $r_{i,j}$: *(variable) work remaining for course j after slot i*

Example 3 (hard): Group Grading

Time slot i starts with $r_{i-1,j}$ work remaining for course j. The TAs can perform $x_{i,j}$ work, and $w_{i,j}h_j$ new work arrives. Therefore, $r_{i,j} = r_{i-1,j} + w_{i,j} \cdot h_j - x_{i,j}$.

Problem (Reminder)

- *t_i: TAs available at time i*
- *h_j: TA hours req. to grade an assgn. from course j*
- *$w_{i,j}$: number assignments from course j that arrive at time slot i*
- *$x_{i,j}$: (variable) for each time slot i, number TAs working on each course j to minimize the average time it takes each submission to be graded*
- *$r_{i,j}$: (variable) work remaining for course j after slot i*

Cost?

- We want to minimize the *average time* it takes each submission to be graded.
- The total time all submissions of course j wait is $\sum_i r_{i,j}/h_j$
  - Each h_j of work remaining at the end of time slot 1 increases the total amount of time the assignments wait by 1.
- The total number of submissions is $\sum_i \sum_j w_{i,j}$
- Need $r_{i,j} \geq 0$!
- Objective function: minimize $\left(\sum_j \sum_i r_{i,j}/h_j\right) / \left(\sum_i \sum_j w_{i,j}\right)$

Example 3: Final LP

Remember that h_j is a constant!

Objective: $\min \left(\sum_j \sum_i r_{i,j}/h_j\right) / \left(\sum_j \sum_i w_{i,j}\right)$

Constraints:

For all i: $\sum_j x_{i,j} \leq t_i$

For all i and all j: $r_{i,j} \geq r_{i-1,j} + w_{i,j} \cdot h_j - x_{i,j}$

For all i and all j: $x_{i,j} \geq 0$ and $r_{i,j} \geq 0$

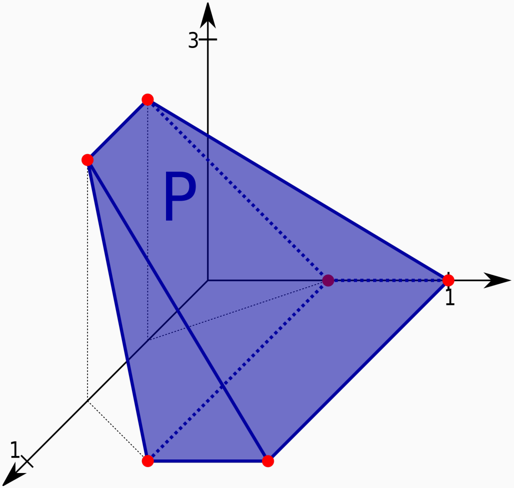
- What are the *variables*? What are the constants?
- Is this an LP? How many variables and constraints does it have?
- How can we go from a feasible LP solution to a real-world schedule?

Structure of Linear Programs

Canonical Form

- Without loss of generality, can always put all constants on the right; can ensure variable appears once per line
- Our solver *does require* that variables all appear on the left and constants all appear on the right.
- Some solvers need other constraints (like all $\leq$); ours doesn't

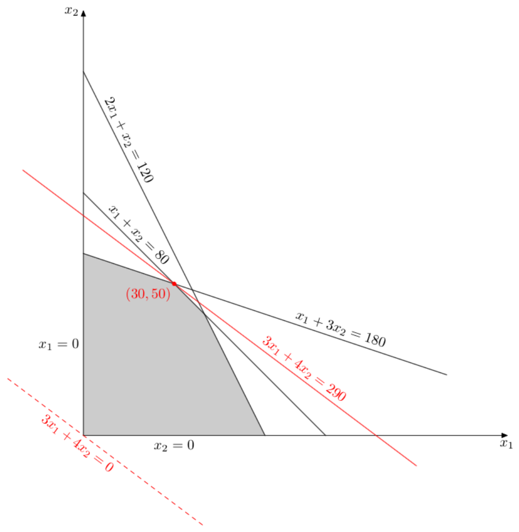
Extreme Points



- Where can a solution lie?
- Can't ever be *inside* the polytope of feasible solutions
- In fact, don't need to look along an edge of the polytope either
- Theorem: any LP has an optimal solution at an at *extreme point*
- Defn: does not lie on a line between two other points in the polytope (intuitively, a vertex of the polytope)

Solving Linear Programs

First Steps



- For small programs, draw them out and solve them
- This is not a bad tactic for solving these by hand

Some Theory on Solving LPs

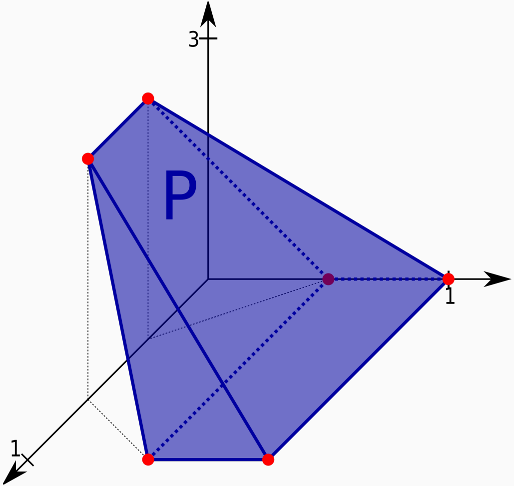
- $O(n)$ time for constant dimensions
- Also: polynomial time algorithm in general!
  - “Ellipsoid method” (Khachiyan 1979)
  - “Interior point methods” (Karmarkar 1984)
  - Best known currently: Cohen, Lee, Song, Zhang 2019
- We’ll learn about an algorithm that’s slower in the *worst case* (not polynomial time), but works extremely well in practice

LP Solving Using the Simplex Algorithm

Simplex algorithm:

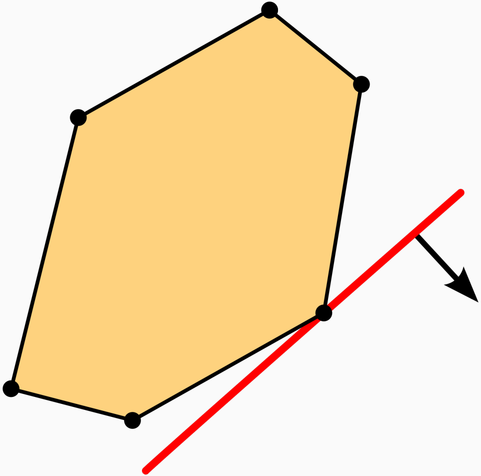
- Invented by Dantzig in 1947
- Simple, most common in practice
- Works extremely well on real-world data
- Exponential time in the worst case
- We will just see just the basics of this algorithm

How do we search through extreme points?



- From one extreme point, we can follow an edge to another
- Pros: local!
- Has a nice algebraic formulation
- But when do we know that we have the best solution?

Going through extreme points



- One option: keep track of which ones we’ve seen, stop once we’ve seen all of them
- Takes up lots and lots of space!
- Not very efficient
- No opportunities for heuristics:
    - even if we see the solution early, need to search through all of them

Key Lemma

Lemma 1

An extreme point is an optimal solution if every adjacent extreme point has a strictly worse objective value.

- That is to say: a local maximum is always a global maximum!
- Adjacent roughly means: connected by a line
- More formally (you don't need to know this vocab): "adjacent" extreme points can be determined by loosening one constraint and tightening another
- Called a "pivot"

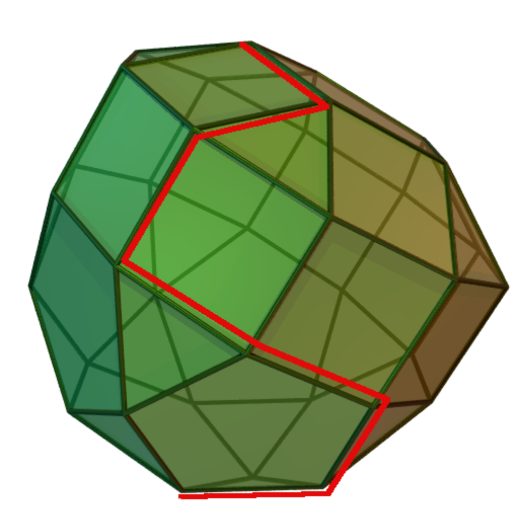
The Simplex Algorithm

- Start at some extreme point
- While there is an adjacent extreme point with the same or better objective function:
  - Go to that extreme point
- Then: Return current extreme point

Does this work?

- By our lemma, if it finishes, the value it returns is correct.
- When might it not finish? What obstacles might it find?
- First: need to find the initial extreme point
  - Significant area of research; usually easy in practice
- Can the algorithm loop infinitely?
  - Yes. Also significant area of research, can generally be avoided in practice (and can always be avoided in theory).

Simplex Algorithm



- This is what simplex does:
- Greedily searches through points
- Does not keep track of previous points
- Very good at getting to the right place quickly in practice

Where to pivot?

- Simplex performance depends on what extreme point we go to next (“pivot rule”)
- How can we choose?
- One option: greedily choose best objective function
  - Not bad, but not as good as you’d think
- 70 years of optimization have gotten us really effective rules
- Some work well for certain types of problems (i.e. network flows)

How fast is it?

We can't get stuck in local minima; can't get stuck in an infinite cycle. Does this mean it's fast in terms of the number of variables and constraints?

- Classic result: there exists an LP with n variables and n constraints such that simplex can take $\Omega(2^n)$ time (Klee Minty 1972)
  - (But *subexponential* pivot rule by Hansen and Zwick in 2015!)
- Can be exponential even if all constants are in $\{1, 2, 3, 4\}$
- Good news: bad cases are very very carefully crafted, extremely rare in practice

Using an LP Solver

LP Solver in this course

- GLPK: open source solver
- Can be called from C, or from python, or used as a standalone program
  - We'll be using as a standalone program
  - *Arguably* easier. (Downside: can't program the generation of the LP. Have to write it out by hand.)
  - If you really want to use the C or python version you can but I think it's ultimately harder for these problems and I don't recommend it
- Industrial solvers may have better performance than GLPK, especially for specific types of LPs. They can be very expensive.

What does GLPK do

- *Best effort* to solve the problem (uses very optimized simplex, plus some other stuff)
- Gives you the best solution it found, tells you whether or not it's optimal.
  - Remember that simplex knows when it arrives at an optimal solution
  - (More advanced techniques can also be used)
- So far: basically solves everything I've tried instantly, optimally
- Full disclosure: I've used this program a few times but I don't know it in and out, especially corner cases

Formatting LP in this class

- We'll be using the CPLEX format
- Pretty much looks like writing the LP in text
- Note: any inequalities may be written as strict inequalities: you can write $<$ rather than $<=$. But $<=$ is always meant!!

CPLEX LP format summary: objective function

- (Must) start with objective function
- write `maximize` or `minimize`
- Then just write the function! (Can name it if you want with name:)
- Example: `minimize obj:  - y1 + 2 bananas - 3.5 y3`
- Or: `minimize obj:  -y1 + 2bananas - 3.5y3`
- Number next to the variable means multiplying

CPLEX LP format summary: Constraints

- Must write `subject to`
- Then, one constraint per line (again, can name)
- Must have one constant on right side of equation

```
Subject To
one:  y1 + 3 a1 - a2 - b >= 1.5
y2 + 2 a3 + 2 a4 - b >= -1.5
two :  y4 + 3 a1 + 4 a5 - b <= +1
.20y5 + 5 a2 - b = 0
```

CPLEX LP format summary: bounding variables

- Special section to give bounds on individual variables
- Useful! (and optional; variables are positive by default)
- Write `bounds` then a sequence of bounds (one per variable)
- `+inf` and `-inf` for infinity; `free` for unbounded variable

```
 bounds
-inf <= a1 <= 100
200 <= a2 <= 300
-100 <= a3
bananas <= 100
x2 = +123.456
x3 free
```

CPLEX LP format summary: finishing it up

- Starting next week: another section for specifying integer variables
- Don't need that section for now!
- Then write `end` keyword
- Can comment *single lines* like so:

```
\* This is a comment *\
```

Running the LP Solver

- `glpsol --cpxlp [LP file] -o [desired output file]`
- `glpsol --cpxlp mylp.lp -o mylp.out`
- Outputs solution to output file (text format!! Despite the extension)
- Also outputs a bunch of information to the command line
- Let’s look at an example!

GLPK Example Usage

Example 1: Diet

- You need to eat 46 grams of protein and 130 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 your diet goals? First, let's formulate the LP together on the board

GLPK Example

Now let's make the file

- Start with objective function
- Then `subject to`, then constraints
- Finally, `bounds` followed by bounds
- Then `end`

Tips

- Last year feedback: easiest homework conceptually, but most tedious
  - I did simplify some parts considerably, but there's a cost to using tools this powerful—they're harder to set up and may not be as user-friendly
- Use a good text editor! A chance to use things like find and replace (regex?), multiple cursors/vertical selections, etc.
- Use *helper variables* to keep things simple. If you're going to write `2 x + 2 y + 2 z` a lot, might want to set `sum = 2 x + 2 y + 2 z` and use `sum` instead
- You can write little python programs that output text for your LP file. Could help considerably if used properly!
- Debugging outputs from GLPK are almost useless. Try different pieces of the line (or file) to narrow down where the issue is