Lecture 14: Linear Programming and Optimization

Sam McCauley

October 29, 2024

Williams College

Admin

- How is Assignment 2 going?
- Reminder: I'm only giving extensions if absolutely necessary this week. Start now!
- Homework 4 back tomorrow
- Questions?

What is an algorithmic problem?

- Constraints
- Objective
- What if we had a *single tool* that could solve *any* problem with certain kinds of constraints and objectives?

Next section of the course

- Frameworks to phrase algorithmic problems
- Allow practical solutions for a wide variety of otherwise-intractable problems
- “Optimization” problems that come up frequently in practice
- This topic is much older and much much broader than anything else we’ve covered
- Focus for this class: using linear programming and integer linear programming (and their solvers) to obtain optimal solutions to difficult problems. (Won’t be focusing on structure, mathematical properties.)

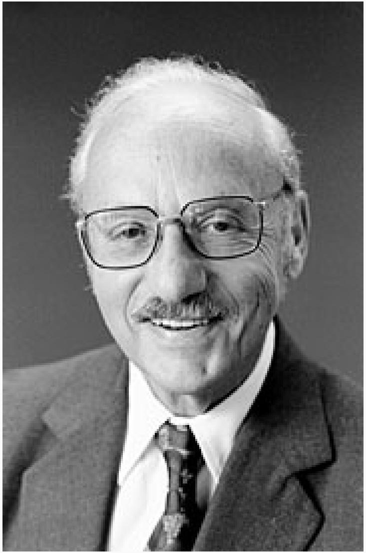
Context

> *I have a strong interest in the question of where mathematical ideas come from, and a strong conviction that they always result from a fairly systematic process—and that the opposite impression, that some ideas are incredible bolts from the blue that require "genius" or "sudden inspiration" to find, is an illusion.*
>
> Timothy Gowers

History

- Starts with a legend

George Dantzig



- Father of Linear Programming
- Worked for military during World War 2
- Invented the simplex algorithm

Linear Programming

A *linear program* consists of:

- a linear objective function, and
- a set of linear constraints.
- (We'll discuss what we mean by linear in a moment.)

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

Why linear programming

- Black-box tools to solve important optimization problems that would be otherwise intractable
- Probably the most powerful tool you'll learn about to solve difficult algorithmic problems
  - More powerful (in a sense) than dynamic programming
  - Strictly *generalizes* network flows
  - Essentially gives a free method to solve continuous optimization problems—as well as some others
- 2004 survey: 85% of fortune 500 companies report using linear programming

What do I mean by “linear”?

- Let’s say our variables are $x_1, \ldots x_n$.
- A linear function is the sum of a subset of these variables, each (possibly) multiplied by a constant.
- Linear inequality: this can be set $\geq$, $\leq$, or $=$ a final constant.
- Example: $4x_1 - 3x_2 \leq 7$ is linear
- Example 2: $4x_1x_2 + x_1 = 3$ is not
- Example 3: $|\sqrt{x_3} - x_7| \geq 5$ is not

Linear Programming

A linear program consists of:

- a linear objective function, (min or max) and
- a set of constraints, which are linear inequalities.
- Goal: achieve the best possible objective function value while satisfying *all of* the constraints
- Note that variables need not be integer or positive

Example of a Linear Program

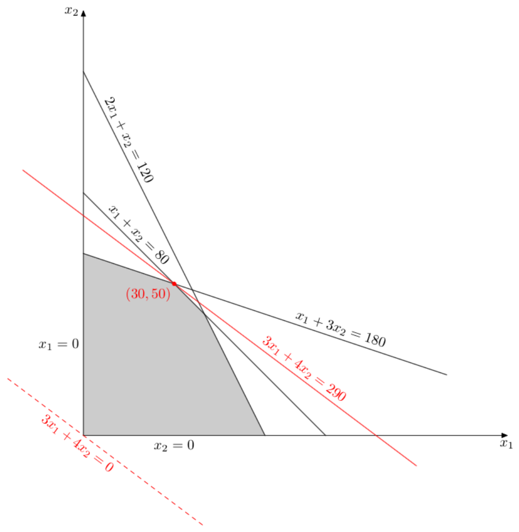
Objective:

$$\max \quad 3x_1 + 4x_2$$

Subject to:

$$\begin{aligned}
2x_1 \quad &+ \quad x_2 && \leq 120 \\
x_1 \quad &+ 3x_2 && \leq 180 \\
x_1 \quad &+ \quad x_2 && \leq 80 \\
x_1 \quad & && \geq 0 \\
& \quad\; x_2 && \geq 0
\end{aligned}$$

Feasibility



- An LP is *feasible* if there exists an assignment of variables that satisfies the constraints
- Nontrivial result: feasibility is not trivial to determine. In the worst case, it is as difficult as solving the entire LP.

Matrix Representation

Objective:

$$\begin{bmatrix} 3 & 4 \end{bmatrix}$$

Subject to:

$$\begin{bmatrix} 2 & 1 \\ 0 & 3 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 120 \\ 180 \\ 80 \\ 0 \\ 0 \end{bmatrix}$$

- Can represent with a matrix and vector
- Useful!
- I don't plan to use this representation again in this class

Visual representation

- We can plot these inequalities
- Works best for instances with 2 or 3 variables
- We'll use extensively as it gives good intuition

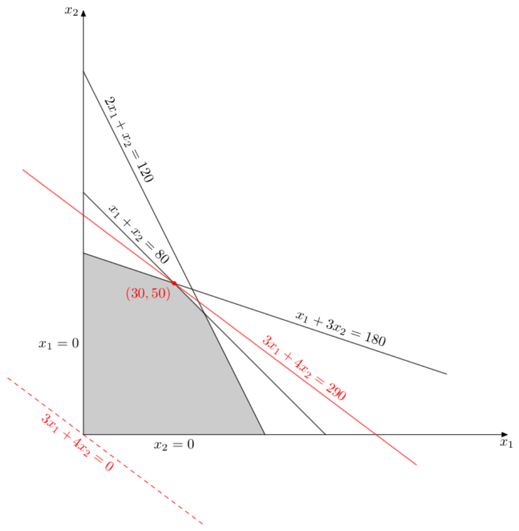
Plotting an LP

Objective:

$$\max \quad 3x_1 + 4x_2$$

Subject to:

$$\begin{aligned} 2x_1 &+ x_2 &\leq 120 \\ x_1 &+ 3x_2 &\leq 180 \\ x_1 &+ x_2 &\leq 80 \\ x_1 & &\geq 0 \\ & x_2 &\geq 0 \end{aligned}$$



Why are we looking at this?

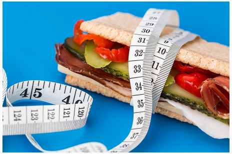
- Let's say I gave you a tool that could solve *any* linear program
- Guarantees correct, optimal solutions!
- Frequently very fast in practice
- Our goal: use this tool to solve computational problems

Why are we looking at this?

- Many problems can be phrased as a linear program
  - We'll start with some slightly contrived problems to build intuition, and eventually get to problems with more real-world importance.
- Linear programs can be solved efficiently
- For today: take as a given that efficient solving is possible. How can we use linear programming to solve these problems?
- Essentially a reduction: similar to using Network Flow to solve problems

Solving Problems with Linear Programming

Optimization Problems



Example 1: Diet

- You need to eat 46 grams of protein and 130 grams of carbs every day
- Available foods:
  - 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?

Diet Problem

How can we phrase this as a linear program? (I'll write the problem on the board; you should think about how you would do it.)

- Let p be the amount of peanuts, r be the amount of rice, and c be the amount of chicken you buy.
- Then what is our *objective function*?
- Answer: The price is $1.61p + .79r + .7c$
- Do we want to maximize or minimize this?
- $\min 1.61p + .79r + .7c$

Diet Problem Constraints

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

- Protein: $25.8p + 2.5r + 13.5c \geq 46$
- Carbs: $16.1p + 28.7r \geq 130$
- Anything else?
- $p \geq 0$, $r \geq 0$, $c \geq 0$

Reminder:

- 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

Diet Problem Solution

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

- Protein: $25.8p + 2.5r + 13.5c \geq 46$
- Carbs: $16.1p + 28.7r \geq 130$
- $p \geq 0$, $r \geq 0$, $c \geq 0$

Solution: $p = 0$, $r = 2.9216...$, $c = 2.86636...$

So we want to buy about 293g of rice, and 287g of chicken, for total cost \$4.32

Diet Problem Solution

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

- Protein: $25.8p + 2.5r + 13.5c \geq 46$
- Carbs: $16.1p + 28.7r \geq 130$
- $p \geq 0$, $r \geq 0$, $c \geq 0$

Solution: $p = 0$, $r = 2.9216...$, $c = 2.86636...$

So we want to buy about 293g of rice, and 287g of chicken, for total cost \$4.32

Example 2: Extending the Diet

- What if I wanted to limit the amount of rice I eat to 100g?
  - Add a constraint: $r \leq 1$
- What if I wanted a *balanced* diet—the amount of any pair of foods is within 50 grams of each other?
  - First: how would we write these constraints if we don't require that they are linear?
  - $|r - c| < .5$, $|c - p| < .5$, $|r - p| < .5$
  - Then: how can we use a sequence of constraints to achieve this?
  - if $|x - y| < c$ then $x - y < c$ and $y - x < c$. So:
  - $r - c < .5$, $c - p < .5$, $r - p < .5$, $c - r < .5$, $p - c < .5$, $p - r < .5$

Example 3: Facility Location (Harder Problem)

- Given coordinates for n roommates $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Goal: find location for a router that minimizes the average distance to each roommate
- Distance from (x, y) to (x_i, y_i) is $|x - x_i| + |y - y_i|$
- Cannot have distance more than 10 from any roommate

Example 3: Facility Location

Objective:

Constraints:

$$(x-3)+(y-4) \leq 10$$
$$(-x+3)+(y-4) \leq 10$$
$$(-x+3)+(-y+4) \leq 10$$
$$(x-3)+(-y+4) \leq 10$$
$$(x-13)+(y-5) \leq 10$$
$$(-x+13)+(y-5) \leq 10$$
$$(-x+13)+(-y+5) \leq 10$$
$$(x-13)+(-y+5) \leq 10$$

Can't make objective function. Idea: add new variables!

- Given roommates at $(3,4)$ and $(13,5)$
- Goal: find location for a router that minimizes the average distance to each roommate
- Distance from (x,y) to (x_i,y_i) is $|x-x_i|+|y-y_i|$
- Cannot have distance >10 from any roommate

Example 3: Facility Location

Objective: $\min d_1 + d_2$

Constraints:

$$
\begin{aligned}
(x-3)+(y-4) &\leq d_1 \\
(-x+3)+(y-4) &\leq d_1 \\
(-x+3)+(-y+4) &\leq d_1 \\
(x-3)+(-y+4) &\leq d_1 \\
(x-13)+(y-5) &\leq d_2 \\
(-x+13)+(y-5) &\leq d_2 \\
(-x+13)+(-y+5) &\leq d_2 \\
(x-13)+(-y+5) &\leq d_2 \\
d_1 &\leq 10 \\
d_2 &\leq 10
\end{aligned}
$$

- Given roommates at $(3, 4)$ and $(13, 5)$
- Goal: find location for a router that minimizes the average distance to each roommate
- Distance from (x, y) to (x_i, y_i) is $|x - x_i| + |y - y_i|$
- Cannot have distance > 10 from any roommate

Example 3: Facility Location

Objective: $\min d_1 + d_2$

Constraints:

$$
\begin{aligned}
x + y - d_1 &\leq 7 \\
-x + y - d_1 &\leq 1 \\
-x - y - d_1 &\leq -7 \\
x - y - d_1 &\leq -1 \\
x + y - d_2 &\leq 18 \\
-x + y - d_2 &\leq -8 \\
-x - y - d_2 &\leq -18 \\
x - y - d_2 &\leq 8
\end{aligned}
$$

- Given roommates at $(3, 4)$ and $(13, 5)$
- Goal: find location for a router that minimizes the average distance to each roommate
- Distance from (x, y) to (x_i, y_i) is $|x - x_i| + |y - y_i|$
- Cannot have distance > 10 from any roommate

Proving Correctness

- How can we show that the above LP works?
- Idea: an LP is feasible if and only if it corresponds to a correct router placement
- 1st: if there exists a feasible LP solution has values d_1, d_2, x, y then there exists a router placement at (x, y) with distance *at most* d_1 and d_2 from roommates 1 and 2, with $d_1 \leq 10$ and $d_2 \leq 10$
- 2nd: any placement of a router at location (x, y), with distance $d_1 \leq 10$ and $d_2 \leq 10$ from the first and second roommate respectively corresponds to a feasible LP solution with variables d_1, d_2, x, y
- If we can prove these claims then solving this LP solves the router placement problem: we get the min total distance placement

Proving Correctness

Lemma 1

If there exists a feasible LP solution with variables d_1, d_2, x, y then a router at (x, y) has distance at most d_1 and d_2 from roommates 1 and 2, with $d_1 \leq 10$ and $d_2 \leq 10$

Proof: Router at (x, y) has distance $\hat{d}_1 = |x - 3| + |y - 4|$ from roommate 1. Because the LP soln is feasible, we have:

$$(x - 3) + (y - 4) \leq d_1 \qquad (-x + 3) + (y - 4) \leq d_1$$
$$(-x + 3) + (-y + 4) \leq d_1 \qquad (x - 3) + (-y + 4) \leq d_1$$

Since $\hat{d}_1$ is equal to the left side of one of these equations, $\hat{d}_1 \leq d_1$. Furthermore, since the LP solution is feasible, $d_1 \leq 10$, so $\hat{d}_1 \leq 10$.

Same argument works for roommate 2

Proving Correctness

Lemma 2

Any placement of a router at location (x, y), with distance $d_1 \leq 10$ and $d_2 \leq 10$ from the first and second roommate respectively corresponds to a feasible LP solution with variables d_1, d_2, x, y

Proof summary: We have $d_1, d_2 \leq 10$ by definition. We need to show the roommate constraints are satisfied. Let's focus on d_1. We have $d_1 = |x - 3| + |y - 4|$.

For any x, y we have:

$$\begin{aligned} x - 3 &\leq |x - 3| & \qquad -x + 3 &\leq |x - 3| \\ y - 4 &\leq |y - 4| & \qquad -y + 4 &\leq |y - 4| \end{aligned}$$

Substituting, all equations for d_1 are satisfied.

Proving Correctness

- Therefore, the best LP solution gives the best router placement!
- So we can solve this problem by solving an LP

Router Example Discussion

- Can we add new roommates?
  - Yes!
- New constraints? (E.g. can't have the router in a certain portion of the house, or can't be too close to one of the roommates)
  - Yes—if they're linear

Taking a step back

- Useful: can generalize (weighting, additional constraints, additional dimensions)
- Some intuition: what can you encode with an LP?
  - *Continuous*: cannot explicitly require integer values
  - *AND*: can add new constraints. But not *OR*: can't just select one to satisfy
  - (Example: distance absolute value worked because $d_1 \geq 3 - x$ AND $d_1 > x - 3$. Cannot do something like $d > 5$ OR $d < 3$.)

Examples of problems that are harder or impossible to generalize to an LP:

- Peanuts come in packs; can only buy an integer number
- Buying *two* routers for the house. (Each roommate needs to connect to one OR the other)

Taking a step back

Things to note

- Can (and often want to) create new variables when making an LP
- Each *instance* of the problem may require a new LP
- Example: for a general roommate at (x_1, y_1) instead of $(3, 4)$: I would have $x + y - d_1 \leq x_1 + y_1$, rather than $x + y - d_1 \leq 7$,

Variables vs Constants in an LP

- LPs must be linear functions of the *variables*
- In other words, must be linear in the things we are solving for!
- What were the variables in the diet problem? In the router problem?
- In general: the parameters of the specific instance are *constants* as far as the LP is concerned (x_1 and y_1 are "constants" in the above)
- You may multiply these constants, do precomputations on them—whatever you want so long as you get a final correct LP for the given *instance*

Variables vs Constants in an LP

Reminder of what we're doing

- A linear program is a recipe
- Let's say you have roommates and you actually want to figure out the best place to put the router. What will you do?
- Find the *actual values* of x_1, y_1, etc.
- Set up the system of equations above for your actual roommates
- Use an LP solver to find the best x and y
- Takeaway: when you are using the LP solver for a *specific instance*, the only variables here are x and y.

What can you solve with LP?

- Clasically: optimization problems (resource allocation, network flow like problems)
- Magic wand if your problem is continuous and has linear constraints and objective
- Also odd things like shortest path, even things like sorting

Back to router example

- Let’s say we used Euclidean distance with the router. Can we use an LP then?
- $$d((x, y), (x_1, y_1)) = \sqrt{(x - x_1)^2 + (y - y_1)^2)}$$
- Don’t need the square root to minimize...
- But still doesn’t seem possible

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

Let's create variables for 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.

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?

Yep, $\sum_j x_{i,j} \leq t_i$

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, so they can grade a total of $x_{i,j}/h_j$ assignments.

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 $r_{i-1,j}$ assignments remaining for course j. The TAs can grade $x_{i,j}/h_j$ assignments, and $w_{i,j}$ new assignments are turned in. Therefore, $r_{i,j} \geq r_{i-1,j} + w_{i,j} - x_{i,j}/h_j$.

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 $\sum_i r_{i,j}$
- The total number of submissions is $\sum_i \sum_j w_{i,j}$
- Need $r_{i,j} \geq 0$!
- Objective function: minimize $\left(\sum_j \sum_i r_{i,j}\right) / \left(\sum_i \sum_j w_{i,j}\right)$

Example 3: Final LP

Remember that h_j is a constant!

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

Constraints:

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

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

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

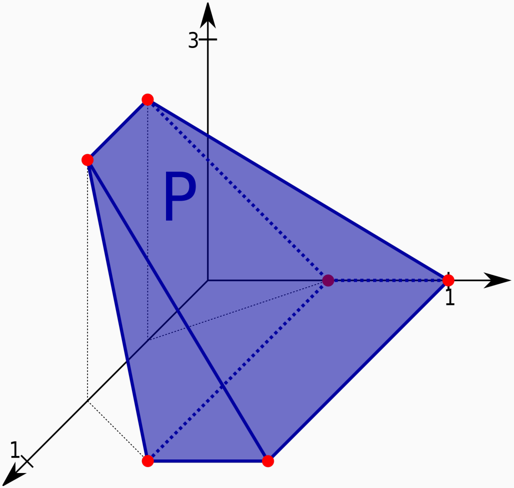
- What are the variables? What are the constants?
- Is this an LP? What is its size? How many dimensions?
- How can we go from a feasible LP solution to a real-world schedule?

Structure of Linear Programs

Canonical Form

- Without loss of generality, can always put all constants on the right
- All constraints are $=$ without loss of generality
  - Use *auxiliary variables* to achieve $\leq$ or $\geq$
  - $3x - 3 \geq 0$ becomes: $3x - a_0 = 3$ for some $a_0 \geq 0$
  - $x - 3 + y - 4 \leq d_1$ becomes: $x + y - d_1 + a_1 = 7$ for some $a_1 \geq 0$
- Necessary for some LP solvers. I believe we won't need this for our solver.

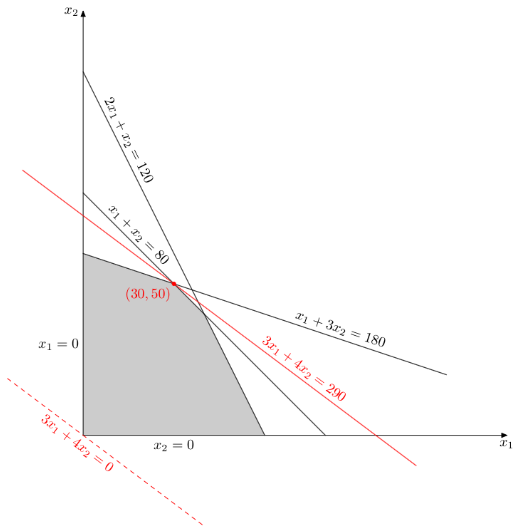
Extreme Points



- Where can a solution lie?
- Can't ever be *inside* the polytope
- In fact, don't need to look along a line either
- Without loss of generality, all solutions are at an *extreme point*
- Defn: does not lie on a line between two other points in 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

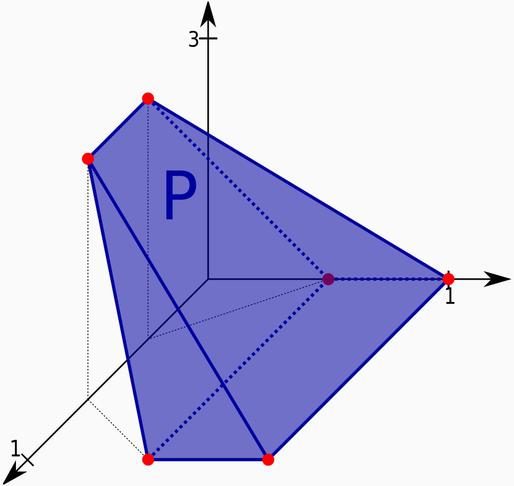
Some theory

- $O(n)$ time for constant dimensions
- Polynomial time algorithm in general!
    - “Ellipsoid method” (Khachiyan 1979)
    - “Interior point methods” (Karmarkar 1984)
    - Best known currently: Cohen, Lee, Song, Zhang 2019
    - “Strongly” polynomial still open

Simplex Algorithm

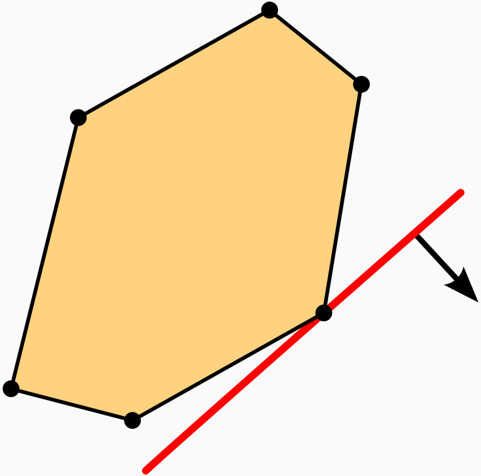
- Invented by Dantzig in 1947
- Simple, most common in practice
- Works extremely well on real-world data
- Exponential time in the worst case
- We will just see a tiny piece of this algorithm

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
- This 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 3

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 means connected by a line
- More formally: "adjacent" extreme points can be determined by loosening one constraint and tightening another
- Called a "pivot"

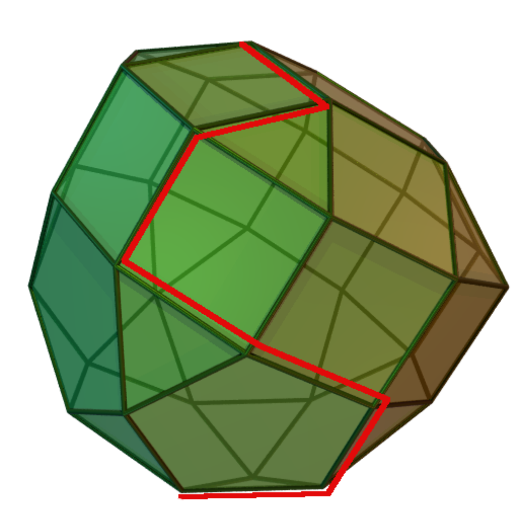
Simplex: searching through extreme points

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

Does this work?

- By our lemma, if it finishes, the value it returns is correct.
- When might it not finish?
- 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.

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?

- 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!)
- Even if all constants are in $\{1, 2, 3, 4\}$
- Good news: bad cases are very very carefully crafted, extremely rare in practice

Conclusion/Summary

What Takeaways do I want?

- What is an LP?
- How to take a problem and phrase it as a linear program?
- How does the simplex algorithm work?