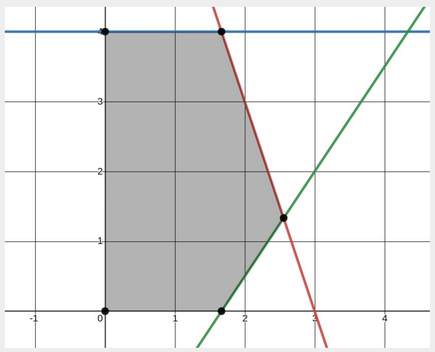
Integer Linear Programming is NP-Complete, but Linear Programming is in P

Petrichor Park, Batmend Batsaikhan, Mary Blanchard, Cecelia Erlichman, Andreas Miller, Hugh Shanno

What is (Integer) Linear Programming?

Linear Programming (LP) is a class of **optimization problems**. Given a system of linear equations over a set of variables, and some "objective function," return values for all the variables that **minimize or maximize the objective function**.[1]

Integer Linear Programming (ILP) is the same as LP, but with the additional constraint that the **variables are integers**. Interestingly enough, this constraint makes it NP-complete. Intuitively, this is because one has to check every point satisfying the constraints, not just the vertices.



This graph of an LP problem shows these constraints:

- $x \leq 4$
- $3x + y \leq 9$
- $1.5x - y \geq 2.5$

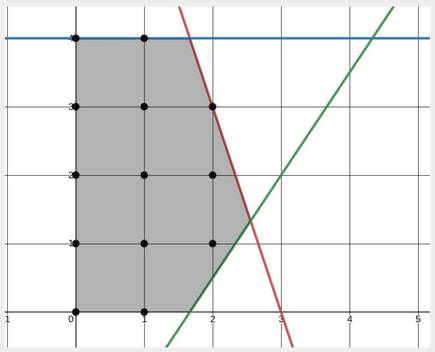
In addition, x and y are assumed to be positive.

The points in black are the possible answers. (Without an objective function, it's unsolvable.)

Integer Linear Programming is NP-Complete…

Although ILP is conceptually simple, it can be used to **easily encode a lot of problems**. A canonical example is the **Knapsack Problem**; create a constraint representing the limited space in the pack, and set the objective function to be the total price carried.

Our project as a whole is focused on using ILP to solve the CS Match, where we try to maximize for happiness of students and have constraints representing (for example) the class sizes or special requirements for graduation.



This is the same constraints as above, but as an ILP problem. Note how the answers must lie on integer coordinates.

Because of the extra restriction, you can no longer just check the vertices of the area; you have to check every point inside. (Fast algorithms that can immediately disregard certain points exist, but there are still many more points to check.)

… but Linear Programming is in P.

The standard algorithm for solving LP problems is called the "simplex algorithm." It usually runs in linear time, but has worst-case exponential time. An important breakthrough was the discovery of the **ellipsoid method** [2] by Soviet mathematician Leonid Khachiyan, which he proved to run in **polynomial time**.

First, it is **possible to check if a certain range of variables is feasible in polynomial time**, by a higher-dimensional geometric argument. Consider the problem to be a space of n dimensions, where n is the number of variables. Start with an n-ellipsoid in this space that is large enough it is guaranteed to include all solutions. Next, it is possible to (in polynomial time) either return a point in this ellipsoid that satisfies the constraints, or shrink the size of the ellipse. Continually doing this and shrinking the ellipse is proved to terminate in polynomial time, either by finding a feasible region or by saying that none exist.

Secondly, given this procedure, it is possible to **zero in on the best** set of variables for the LP problem. This is done by binary searching the space. Divide the space in two and check for feasibility on both sides. If only one side is feasible, repeat on that side, but if both are feasible, because the objective function must be linear it is easy to check which side will contain the best objective value.

Putting all this together results in an **algorithm that terminates in polynomial time** with an optimum solve of the problem, so **linear programming is in P.** Although this algorithm is not very efficient in practice, it was an important theoretical step, and now other algorithms have been found that run in faster polynomial times.

References

[1] Wayne L. Winston. 2004. *Operations Research: Applications and Algorithms* (4th. ed.). Brooks/Cole – Thomson Learning, Belmont, CA.

[2] Sanjeev Arora. The Ellipsoid Algorithm for Linear Programming. Retrieved October 27, 2023. https://www.cs.princeton.edu/courses/archive/fall05/cos521/ellipsoid.pdf

Acknowledgements

I would like to thank Layla Oesper for mentoring our comps group, and Dave Musicant for giving us lots of information about how the Match works internally and allowing us access to its code.