DS 320 Algorithms for Data Science **Lecture #19**
Spring 2026 Prof. Kira Goldner and Jeffrey Considine

Linear Programming II: Algorithms, Problems, and Approximations

NP-Completeness Recap

a. Decision problems return yes or no answers.

b. P is the class of decision problems solvable in polynomial time.

c. NP is the class of decision problems where yes answers can be confirmed in polynomial time with a *certificate*.

d. NP-hard problems are decision problems to which any NP problem is polynomial reducible.

e. NP-complete problems are both NP-hard and in NP.

f. The first NP-complete problem was SAT.

g. Most later NP-complete problems are proven with reductions from SAT or previous NP-complete problems.

Integer Linear Programs are NP-Complete

Via polynomial time reduction from 3CNF to integer linear programming (ILP),

a. We will map 3CNF assignments and formulas to ILP assignments and formulas.

(a) 3CNF variables x_i maps to ILP variable z_i.
(b) False variable assignments map to zero ILP variable assignments.
(c) True variable assignments map to one ILP variable assignments.

b. We will then show that there is a similar correspondance between the 3CNF formulas values and the ILP formula values.

(a) If the 3CNF formula value is zero, then the ILP formula value is also zero.
(b) If the 3CNF formula value is one, then the ILP formula value is at least one.

c. The mapping works as follows.

(a) Each original 3SAT variable x_i maps a new ILP variable z_i. Add two constraints $0 \leq z_i$ and $z_i \leq 1$.
(b) Each negated variables $\bar{x_i}$ maps to an ILP expression $(1 - z_i)$.
(c) The jth clause becomes a linear inequality where the sum of the mapped literals must be at least one. For example, $(x_1 \vee x_2 \vee \bar{x_3})$ maps to $z_1 + z_2 + (1 - z_3) \geq 1$.

d. The objective function does not matter; any feasible solution to the ILP corresponds to a satisfying assignment to the 3CNF formula.

What Does (Non-Integer) Linear Programming Buy Us?

a. We know efficient algorithms exist (and have a nice theory behind them).

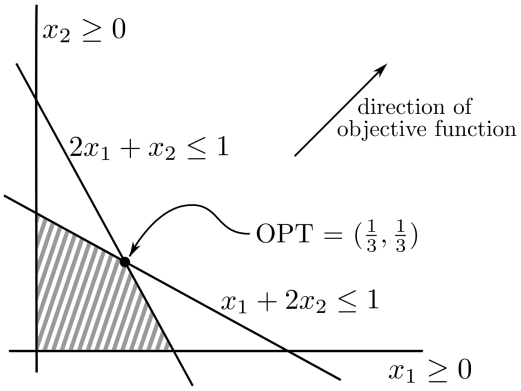
b. We can relate problems to one another through relaxations, duality.

c. It gives us techniques for approximation.

Linear Programming Algorithms

The standard algorithm for solving LPs is the Simplex Algorithm, developed in the 1940s. Its *worst-case* running time is not polynomial: you can come up with bad examples for it. But in practice, the algorithm runs pretty fast. Only much later in 1980 was it shown that linear programs could always be solved in polynomial time by the Ellipsoid Algorithm (but it tends to be slow in practice). Later on, a faster polynomial-time algorithm called Karmarkar's Algorithm was developed, which is competitive with Simplex. In practice, what you should actually do is use a commercial LP package, for instance LINDO, CPLEX, Gurobi, and Solver (in Excel). We'll just give some intuition and the high-level idea of how they work by viewing linear programming as a geometrical problem. Then we'll talk about an elegant algorithm for low-dimensional problems.

Geometry: Think of an n-dimensional space with one coordinate per variable. A solution is a point in this space. An inequality, like $x_1 + x_2 \leq 6$ is saying that we need the solution to be on a specified side of a certain hyperplane. The feasible region is the convex region in space defined by these constraints. Then we want to find the feasible point that is farthest in the "objective" direction.



$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_1 \geq 0 \\ & x_2 \geq 0 \\ & 2x_1 + x_2 \leq 1 \\ & x_1 + 2x_2 \leq 1. \end{aligned}$$

The Simplex Algorithm: The idea is to start at some "corner" of the feasible region. Then we repeatedly do the following step: look at all neighboring corners of our current position and go to the best one (the one for which the objective function is greatest) if it is better than our current position. Stop when we get to a corner where no neighbor has a higher objective value than we currently have. The key facts here are that

1. since the objective is linear, the optimal solution will be at a corner (or maybe multiple corners), and

2. there are no local maxima: if you're not optimal, then some neighbor of you must have a strictly larger objective value than you have. That's because the feasible region is convex.

So, the Simplex method is guaranteed to halt at the best solution. The problem is that it is possible for there to be an exponential number of corners, and it is possible for Simplex to take an exponential number of steps to find the optimal corner. But, in practice this usually works well.

The Ellipsoid Algorithm: The Ellipsoid Algorithm was invented by Khachiyan in 1980 in Russia. This algorithm solves just the "feasibility problem," but you can then do binary search with the objective function to solve the optimization problem. The idea is to start with a big ellipse (called an ellipsoid in higher dimensions) that we can be sure contains the feasible region. Then, try the center of the ellipse to see if it violates any constraints. If not, you're done. If it does, then look at some constraint violated. So we know the solution (if any) is contained in the remaining at-most-half-ellipse. Now, find a new smaller ellipse that contains that half of our initial ellipse. We then repeat with the new smaller ellipse. One can show that in each step, you can always create a new smaller ellipse whose volume is smaller, by at least a $(1 - 1/n)$-factor, than the original ellipse. So, every n steps, the volume has dropped by about a factor of $1/e$. One can then show that if you ever get too small a volume, as a function of the number of bits used in the coefficients of the constraints, then that means there is no solution after all.

One nice thing about the Ellipsoid Algorithm is you just need to tell if the current solution violates any constraints or not, and if so, to produce one. You don't need to explicitly write them all down. There are some problems that you can write as a linear program with an exponential number of constraints if you had to write them down explicitly, but where there is an fast algorithm to determine if a proposed solution violates any constraints and if so to produce one. For these kinds of problems, the Ellipsoid Algorithm is a good one.

Writing Problems We Know as Linear Programs

Independent Set

Recall from last lecture that we formulated the Independent set problem as a linear programming relaxation.

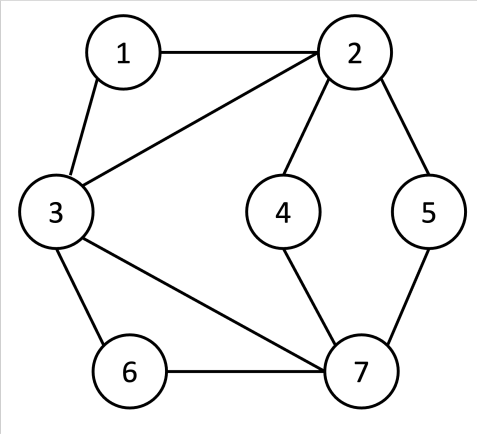
Given a graph $G = (V, E)$, each vertex i has weight w_i, find a maximum weighted *independent set*. S is an independent set if it does not contain both i and j for $(i, j) \in E$.

$$\begin{aligned} \max & \sum_{i \in V} w_i x_i & \\ \text{s.t.} \quad & x_i + x_j \leq 1 & (i,j) \in E \\ & 0 \leq x_i \leq 1 & i \in V. \end{aligned}$$

The Vertex Cover Problem

Given a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is a *vertex cover* if every edge $e = (i, j) \in E$ has at least one endpoint i or j in S. Our goal is to find a *minimum* vertex cover.

For the decision version of the problem, we ask: Given a graph G and a number k, does G contain a vertex cover of size at most k?



In this graph, the *minimum* vertex cover is the set of nodes {2, 3, 7} for a size of 3.

This is the same graph from last time when we discussed Independent Set. Do we notice any relationship?

Claim 1. For any graph $G = (V, E)$, S is an independent set if and only if $V \setminus S$ is a vertex cover.

Corollary 1. *Finding a maximum independent set is equivalent to finding a minimum vertex cover. Then Independent Set $\leq_P$ Vertex Cover and Vertex Cover $\leq_P$ Independent Set.*

Corollary 2. *Vertex Cover is NP-complete.*

Vertex Cover as an Integer Program

a. *Decision variables:* What are we try to solve for? A set of vertices S that is our vertex cover. So our variables are x_i for each item i, where we want $x_i = 1$ if i is in our vertex cover.

b. *Constraints:* We can never put more than 1 of a vertex into our cover, so

$$x_i \leq 1 \quad \forall i$$

and similarly, we can never take a negative quantity of an vertex, so

$$x_i \geq 0 \quad \forall i$$

Finally, we need to take at least one endpoint per edge:

$$x_i + x_j \geq 1 \quad (i,j) \in E$$

c. *Objective function:* We want to minimize the size/weight of our vertex cover:

$$\max \sum_i v_i x_i$$

Note that this is again a linear function.

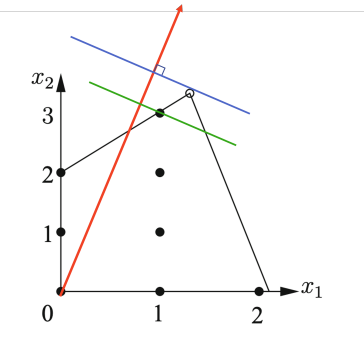
$$\begin{aligned} \min & \sum_{i \in V} w_i x_i & \\ \text{s.t.} \quad & x_i + x_j \geq 1 & (i,j) \in E \\ & x_i \in \{0,1\} & i \in V. \end{aligned}$$

Vertex Cover as a Linear Program

$$\begin{aligned} \min & \sum_{i \in V} w_i x_i & \\ \text{s.t.} \quad & x_i + x_j \geq 1 & (i,j) \in E \\ & x_i \in [0,1] & i \in V. \end{aligned}$$

Claim 2. Let S^* denote the optimal vertex cover of minimum weight, and let x^* denote the optimal solution to the Linear Program. Then $\sum_{i \in V} w_i x_i^* \leq w(S^*) = \text{OPT}$.

Proof. The vertex cover problem is equivalent to the integer program, whereas the linear program is a *relaxation*. Then there are simply more solutions allowed to the linear program, so the minimum can only be smaller. □



$$\begin{aligned} \max \quad & 4x_1 + x_2 \\ \text{subject to} \quad & -x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 17 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Figure 1: Left: The red arrow represents the objective function, with the green line tangent to the set of feasible integer solutions, indicating the optimal integral point, and the blue line tangent to the relaxed convex feasible set, indicating the best fractional point in the relaxation, with a larger objective function. Right: The linear program for the figure on the left.

Approximation

Using Linear Programming for a Vertex Cover Approximation Algorithm

$$\begin{aligned} \min & \sum_{i \in V} w_i x_i \\ \text{s.t.} \quad & x_i + x_j \geq 1 && (i,j) \in E \\ & x_i \in [0,1] && i \in V. \end{aligned}$$

Claim 3. Let S^* denote the optimal vertex cover of minimum weight, and let x^* denote the optimal solution to the Linear Program. Then $\sum_{i \in V} w_i x_i^* \leq w(S^*) = \text{OPT}$.

Proof. The vertex cover problem is equivalent to the integer program, whereas the linear program is a *relaxation*. Then there are simply more solutions allowed to the linear program, so the minimum can only be smaller. □

Claim 4. The set $S = \{i : x_i \geq 0.5\}$ is a vertex cover, and $w(S) \leq 2 \sum_{i \in V} w_i x_i^*$.

Proof. First, S is a vertex cover: for any edge $e = (i,j)$, at least one of i or j must be in S, because of our constraint $x_i + x_j \geq 1$, which forces at least one of these variables to be $\geq \frac{1}{2}$ and thus in S.

With respect to weight:

$$\sum_{i \in V} w_i x_i^* \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i = \frac{1}{2} w(S).$$

□

Then our algorithm of running an LP and rounding it to give the vertex cover S is a 2-approximation to the optimal vertex cover S^*, as $w(S) \leq 2\, w(S^*)$ by Claims 3 and 4.