

Chapter 6

Linear Programming I: Basics and Applications

In the second part of the book, we study linear programming (LP) and semidefinite programming (SDP) and explore how these powerful optimization frameworks can be used to design and analyze efficient algorithms.

6.1 Introduction to Linear Programming

6.1.1 Optimization Problems

Optimization problems are everywhere: How to design the best schedule of classes? How to go from point A to point B in the quickest possible way? How to transfer information in a network at the highest rate possible? How to fit the largest number of packages in a delivery truck? These, and numerous examples alike, are all optimization problems.

For our purpose, we can model optimization problems formally as follows.

Definition 6.1. Let Ω be a **domain** and $f : \Omega \rightarrow \mathbb{R}$ be an **objective function** that assigns values to the elements of this domain. Moreover, let $C \subseteq \Omega$ denote a subset of the domain identified by the **constraints** of the problem. The goal is to solve the following:

$$\max_{x \in \Omega} f(x) \quad \text{subject to} \quad x \in C.$$

In words, [Definition 6.1](#) simply defines optimization problems as finding an element of the domain Ω that also belongs to the “feasible” subset C and maximizes the value of $f(x)$. Notice that it is also possible for an optimization problem to involve *minimizing* the objective function instead; however, since minimizing $f(x)$ and maximizing $-f(x)$ are equivalent, there is no need for a further definition.

Attempting to solve optimization problems at this level of generality is virtually hopeless. For instance,

- Consider the following family of optimization problems $\{\Omega_n, f_n : \Omega_n \rightarrow \mathbb{R}, C_n\}_{n \in \mathbb{N}}$: for any integer $n \in \mathbb{N}$, we take Ω_n to be the set of all n -state Turing machines, C_n to be the subset of n -states Turing machines that eventually halt on the input 0, and $f_n(x)$ for each Turing machine $x \in \Omega_n$ be the function that counts the number of 1’s output by x on input 0 (we set $f_n(x) = 0$ if x outputs any symbol other than 1).

Now, maximizing $f_n(x)$ subject to $x \in C_n$ is equivalent to the infamous *busy beaver* problem which is a well-known undecidable problem. Thus, it is impossible to design an algorithm for solving this optimization problem in any finite time.

- Another, perhaps “scarier” example, is the following optimization problem on only four variables:

$$\min_{x=(x_1, x_2, x_3, x_4) \in \mathbb{N}^4} |x_1^{x_4} + x_2^{x_4} - x_3^{x_4}| \quad \text{subject to} \quad x_1, x_2, x_3 \geq 1 \text{ and } x_4 \geq 3.$$

Checking whether the optimal solution to this optimization problem has value 0 or not is equivalent to proving *Fermat’s Last Theorem*!

Given these, it is clear that we need to impose certain restrictions on our optimization problems before we can hope to study them further.

6.1.2 Linear Optimization (a.k.a. Linear Programming)

We examine a canonical family of optimization problems: *Linear Programs (LPs)*¹. In a linear program, the domain Ω is $\mathbb{R}^n$, the objective function is a linear function $c^\top \cdot x$ (or equivalently $\langle c, x \rangle$), and the set C is identified by a series of linear constraints of the form $\langle a, x \rangle \geq b$.

Definition 6.2. In a linear program (LP), we have an n -dimensional vector of **variables** $x \in \mathbb{R}^n$, an n -dimensional **objective function** $c \in \mathbb{R}^n$, a set of m **constraints** given via an $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$ and an m -dimensional vector $b \in \mathbb{R}^m$. The goal is to solve the following problem:

$$\max_{x \in \mathbb{R}^n} c^\top \cdot x \quad \text{subject to} \quad A \cdot x \geq b.$$

See [Exercise 6.1](#) for other equivalent formulations, e.g., as a minimization program instead of a maximization one. A simple example of a linear program is the following:

$$\begin{aligned} \max_{(x_1, x_2) \in \mathbb{R}^2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2 \cdot x_2 \leq 6 \\ & 2 \cdot x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0. \end{aligned}$$

The optimal solution to this LP is $(x_1, x_2) = (2, 2)$ which achieves the value of 4 (see if you can *verify* this is indeed an optimal solution); see also [Figure 6.1](#).

There are many applications of linear programs:

- (a) We can *solve* numerous “day-to-day life” problems, ranging from simple scheduling to complicated scientific computing tasks, by modeling them via LPs and solving them using available LP solvers.
- (b) We can use LPs as a tool toward *designing* “generic” algorithms for various problems, for instance, in *combinatorial optimization*.
- (c) We can use LPs as an *analytical* tool for proving theorems (particularly in combinatorics) by exploiting strong properties of LPs such as duality theorems (that we will see soon).

¹The term ‘programming’ here has nothing to do with computer programming and instead is used as a term related to ‘scheduling’.

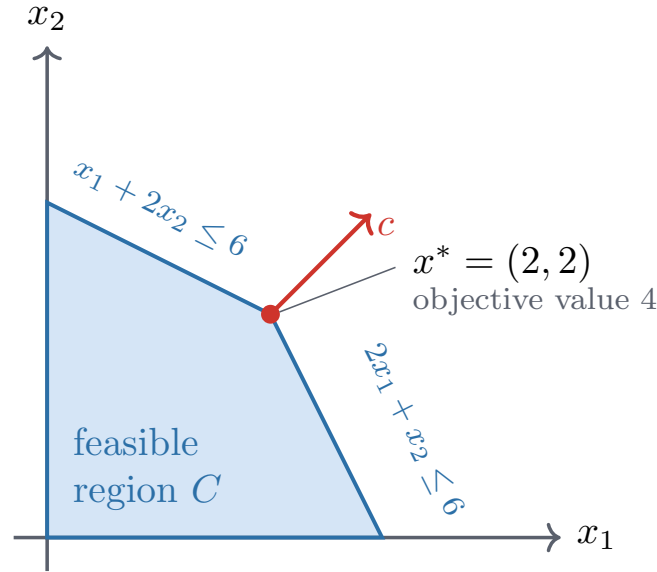


Figure 6.1: A geometric view of the example linear program (after Definition 6.2), drawn for two variables.

But there is also another highly important reason behind studying LPs: we can solve them *efficiently* both in theory and practice! Specifically for the former part, we have this theorem.

Theorem 6.3. Any linear program specified by $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ (Definition 6.2) can be solved in time polynomial in n, m and bit-length of entries in A, b, c .

More generally, an LP can be solved in time polynomial in n and bit-length of entries in A, b, c —irrespective of its number of constraints—as long as we have a polynomial time algorithm that given a potential solution x to the LP, determines whether x is feasible and if not, returns one constraint violated by x .

Covering the proof of Theorem 6.3 is beyond the scope of this book but we will see some ideas related to it in later chapters. Before moving, a brief remark on the history of linear programming is in order.

Remark. Linear programming has its roots in the late 1930s and 1940s. Kantorovich [Kan60] introduced it for production planning—work for which he later shared the 1975 Nobel Memorial Prize in Economics—while Dantzig formulated the general problem and gave its first practical algorithm, the celebrated *simplex method* [Dan51]. Simplex remains one of the most widely used LP algorithms and is often quite fast in practice even though it can take exponentially many steps in the worst case. Whether LPs can be solved in *polynomial* time was first settled, affirmatively, by Khachiyan [Kha79] via the *ellipsoid method*; soon after, Karmarkar [Kar84] gave the first efficient *interior-point method*, the approach underlying most modern fast LP solvers. Building on interior-point methods, the current state of the art solves a general LP in essentially the time of a single matrix multiplication [CLS21] (related ideas have very recently led to almost-linear-time algorithms for cornerstone special cases such as maximum flow and minimum-cost flow [CKL⁺22, vdBCP⁺23]).

6.2 Applications of Linear Programming

Let us make the above applications more concrete via several examples. We will see applications (a) and (b) in this chapter and postpone application (c) to the next one.

6.2.1 An Application of Type (a): Creating a “Balanced” Diet

The following is a textbook example of applications of LPs in solving “day-to-day life” problems, so we might as well repeat it here. Suppose we are interested in creating a balanced diet in terms of only its contents in vitamins A,B, C, its fiber, and its protein. We are further provided with the dietary guidelines on how much of each of these our meal should consist of, as well as a list of foods together with their vitamins/fiber/protein contents. On top of this, we are given the price for each of these foods. This is basically a table of the following form:

	Required amount	Food 1 (e.g., carrot)	Food 2 (e.g., pizza)	...	Food n
Vitamin A	r_A	a_1	a_2		a_n
Vitamin B	r_B	b_1	b_2		b_n
Vitamin C	r_C	c_1	c_2		c_n
Fiber	r_F	f_1	f_2		f_n
Protein	r_P	p_1	p_2		p_n
Price	–	$price_1$	$price_2$		$price_n$

Table 6.1: This table was supposed to be filled with actual numbers but since this book is not about dietary guidelines, we instead write them as the above parameters.

The goal is now to find a diet as a combination of these foods that satisfies all the required amounts of vitamins, fiber, and protein, and has the minimum price possible.

We can write this as an LP as follows:

- For $i \in [n]$, let x_i be a variable denoting how much of food i we will include;
- For $i \in [n]$, let a_i, b_i, c_i, f_i, p_i , denote vitamins A,B,C, fiber, and protein content of food i , respectively;
- For $i \in [n]$, let $price_i$ denote the price we have to pay for each unit of food i ;
- Let r_A, r_B, r_C, r_F, r_P denote the required dose of vitamins A,B,C, fiber, and protein.

The last three lines are *not* variables, rather, numbers that are (supposedly) in [Table 6.1](#).

We can now write the following LP for solving our problem.

$$\begin{aligned}
 & \min_{x=(x_1, \dots, x_n) \in \mathbb{R}^n} \sum_{i=1}^n price_i \cdot x_i \\
 & \text{subject to} \\
 & \sum_{i=1}^n a_i \cdot x_i \geq r_A, \quad \sum_{i=1}^n b_i \cdot x_i \geq r_B, \quad \sum_{i=1}^n c_i \cdot x_i \geq r_C, \\
 & \sum_{i=1}^n f_i \cdot x_i \geq r_F, \quad \sum_{i=1}^n p_i \cdot x_i \geq r_P \\
 & x_i \geq 0 \quad \forall i \in [n].
 \end{aligned}$$

(We again note that in this LP, the only variables are $x_1, \dots, x_n$; the remaining parameters are all numbers that we can fill up using [Table 6.1](#) assuming we had the actual numbers.)

We leave verifying that this LP actually solves the original problem to the reader as an easy exercise. At this point, we can simply give this LP to a standard LP solver and obtain the solution to our problem.

6.2.2 Another Application of Type (a): Linear Regression

In the **linear regression** problem, we have a set of m points from the n -dimensional space $\mathbb{R}^n$:

$$(a_{1,1}, a_{1,2}, \dots, a_{1,n}), (a_{2,1}, a_{2,2}, \dots, a_{2,n}) \cdots, (a_{m,1}, a_{m,2}, \dots, a_{m,n}).$$

We can denote these points by a matrix $A \in \mathbb{R}^{m \times n}$. The points are associated with real numbers in $\mathbb{R}$:

$$(b_1, b_2, \dots, b_m).$$

Similarly, we can denote these numbers by a vector $b \in \mathbb{R}^m$.

Think of each column of this matrix as a “feature” or an “attribute” of the points, and think of each real number b_i as the “label” of the point $a_i := (a_{i,1}, \dots, a_{i,n})$. Our goal is to find an “explanation” of these labels in terms of the features using a *linear* function. Formally, we would like to find a *hyperplane* $x \in \mathbb{R}^n$ such that given any point a_i , we can recover the label of a_i , namely, b_i , via $\langle a_i, x \rangle$.

Of course, such a hyperplane can only exist if the system of linear equations $Ax = b$ has a solution. However, this is not generally guaranteed (especially because we often have $m \gg n$). Thus, the goal is to find a hyperplane x with minimal “distance” from b . This leads to the family of regression problems wherein the goal is to solve the following optimization problem:

$$\min_{x \in \mathbb{R}^n} \|A \cdot x - b\|,$$

where we can pick the $\|\cdot\|$ differently depending on the application. Two popular choices of the norm are ℓ_2 -norm and ℓ_1 -norm. You might have seen ℓ_2 -regression elsewhere which admits a closed form solution

$$x = (A^\top A)^{-1} A^\top b;$$

this requires $A^\top A$ to be non-singular which is often the case (if not, we have to take the pseudo-inverse here instead). Regardless, this question is out of the scope of this chapter. We will instead focus on ℓ_1 -regression that can be solved via linear programming. Writing this problem as an LP is not entirely trivial so we are going to do it step by step.

Step 1. Recall that the ℓ_1 -norm of a vector $v \in \mathbb{R}^m$ is defined as:

$$\|v\|_1 := \sum_{i=1}^m |v_i|.$$

Thus, the ℓ_1 -regression problem is the following optimization problem:

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^m |\langle a_i, x \rangle - b_i|.$$

This is of course not yet an LP because the objective function is not a linear function. But we made the problem somewhat simpler without changing it at all (it is straightforward to check the problems are equivalent so far).

Step 2. We are going to define m new variables $z_1, \dots, z_m$, one per row of the matrix A . We write our optimization problem now as:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, z \in \mathbb{R}^m} \quad & \sum_{i=1}^m z_i \\ \text{subject to} \quad & z_i = \max(\langle a_i, x \rangle - b_i, b_i - \langle a_i, x \rangle) \quad \forall i \in [m]. \end{aligned}$$

This is still equivalent to the previous problem because for every $i \in [m]$,

$$|\langle a_i, x \rangle - b_i| = \max(\langle a_i, x \rangle - b_i, b_i - \langle a_i, x \rangle),$$

by definition; enforcing z_i 's being equal to this ensures that the problem remains the same. We are now one step closer since our objective function now is linear although we still have many non-linear constraints.

Step 3. The only (slightly) non-trivial step is this one where we are going to relax the constraints a bit without violating feasibility as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, z \in \mathbb{R}^m} \quad & \sum_{i=1}^m z_i \\ \text{subject to} \quad & z_i \geq \max(\langle a_i, x \rangle - b_i, b_i - \langle a_i, x \rangle) \quad \forall i \in [m]. \end{aligned}$$

It may not be entirely clear that this problem is the same as above, in particular because we actually *expanded* the feasible region. However, we still have the following claim.

Claim 6.4. *In any optimal solution of this problem, we have that for every $i \in [m]$,*

$$z_i = \max(\langle a_i, x \rangle - b_i, b_i - \langle a_i, x \rangle);$$

Proof. Suppose some z_i is strictly larger; then reduce it by a tiny $\varepsilon > 0$ which does not violate the constraint but reduces the objective function, contradicting the optimality. $\square$

This implies that optimal solutions of the problems in steps 2 and 3 coincide and thus we can still focus on solving this problem. We are still not done because our constraints are not yet linear.

Step 4. Finally, we can “linearize” the constraints simply as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, z \in \mathbb{R}^m} \quad & \sum_{i=1}^m z_i \\ \text{subject to} \quad & z_i \geq \langle a_i, x \rangle - b_i \quad \text{and} \quad z_i \geq b_i - \langle a_i, x \rangle \quad \forall i \in [m]. \end{aligned}$$

This is now truly a linear program and is clearly equivalent to the previous step: for all $i \in [m]$:

$$z_i \geq \max(\langle a_i, x \rangle - b_i, b_i - \langle a_i, x \rangle) \iff z_i \geq \langle a_i, x \rangle - b_i \quad \text{and} \quad z_i \geq b_i - \langle a_i, x \rangle$$

Wrap-up. Using the above four steps, we can write any ℓ_1 -regression problem with m points in n dimensions as an LP with $n + m$ variables and $2m$ constraints. Solving this LP then gives us the optimal solution to the ℓ_1 -regression problem as well.

6.2.3 An Application of Type (b): The Bipartite Matching Problem

We consider yet another textbook example of LPs, this time for solving algorithmic problems. In the **bipartite matching** problem, we are given a bipartite graph $G = (L \sqcup R, E)$ with bipartition L and R of vertices. A **matching** M in G is any collection of edges that do not share any vertices (hence, they ‘match’ vertices in L to vertices in R so that each participating vertex in the matching is matched to precisely one other vertex). In the bipartite matching problem, the goal is to find a maximum matching (one with the largest number of edges).

Let us define an *integral* 0/1 variable $y_e \in \{0, 1\}$ for each edge $e \in E$ (we emphasize that these variables are *not* free real variables in $\mathbb{R}^n$ and are constrained to be only 0 or 1). Think of $y_e = 1$ as meaning we pick the edge e in our maximum matching and $y_e = 0$ as we do not.

To ensure that the edges we pick form a matching, we need to make sure we do not pick more than one edge per vertex. We can write this as the following constraint:

$$\forall u \in L : \sum_{e \ni u} y_e \leq 1 \quad \text{and} \quad \forall v \in R : \sum_{e \ni v} y_e \leq 1.$$

Finally, our task of maximizing the number of edges in the matching becomes

$$\max \sum_{e \in E} y_e.$$

Putting these together, we get the following optimization problem:

$$\begin{aligned} \max_y \quad & \sum_{e \in E} y_e \\ \text{subject to} \quad & \forall u \in L : \sum_{e \ni u} y_e \leq 1 \\ & \forall v \in R : \sum_{e \ni v} y_e \leq 1 \\ & \forall e \in E : y_e \in \{0, 1\}. \end{aligned}$$

This optimization problem is *not* an LP (even though it may very much look like it). The problem is with the very last constraint of $y_e \in \{0, 1\}$ that forces the program to only use integer values for the variables. Such a program is called an **Integer Linear Program (ILP)**. Unfortunately, unlike LPs, ILPs are much harder to solve or reason about. For one thing, solving general ILPs is an NP-hard problem (you are asked to prove this in [Exercise 6.3](#)).

There is a general recipe in these situations: what if we *relax* the integrality constraint to obtain the following LP instead?

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \forall u \in L : \sum_{e \ni u} x_e \leq 1 \\ & \forall v \in R : \sum_{e \ni v} x_e \leq 1 \\ & \forall e \in E : 0 \leq x_e \leq 1. \end{aligned}$$

Such an LP is typically called the **LP relaxation** of the original ILP (or directly the original problem). But, now it is no longer clear that solving this LP can tell us much about the original problem. Of course, the optimal value of this LP is always an upper bound on the optimal value

of the original ILP simply because any solution to that ILP is a feasible solution here as well. But can it be that this relaxation made the problem so different that these values now differ drastically from each other?

The answer to this question *for general LP relaxations and ILPs* can be *Yes*; we will see examples of this soon. However, we are now going to prove that for the specific case of the bipartite matching problem, this is not the case.

Proposition 6.5. *Any feasible fractional solution $x \in \mathbb{R}^E$ to the LP for bipartite matching can be rounded to a feasible integral solution $y \in \{0,1\}^E$ without decreasing the objective value.*

Before getting to the proof, notice a direct corollary of this proposition. We can now solve the bipartite matching LP relaxation to find an optimal solution x . We know that the objective value of x is at least as large as the optimal solution of the ILP. By [Proposition 6.5](#), we can then round x to an integral solution y of the same value, thus implying that y is also optimal for the ILP. This then gives us a maximum matching of the original graph and solves the problem.

Proof of Proposition 6.5. Fix the feasible solution $x \in \mathbb{R}^E$. If x is already integral, we are done by taking $y = x$. Otherwise, perform the following step first.

Step one: cycle canceling. Find a cycle in the support of x , denoted by $\text{supp}(x) := \{e \in E : x_e > 0\}$, that contains a fractional value. If no such cycle is found go to the next step, otherwise let C be the cycle and let $e \in C$ be the edge with the smallest fractional value x_e in the cycle. Let $\delta = x_e$. Now update x as follows. Let $C = e_1, e_2, e_3, \dots, e_\ell$ denote the edges of the cycle ordered such that $e_1 = e$ is the picked edge with $x_e = \delta$. Update

$$x_{e_1} \leftarrow x_{e_1} - \delta, \quad x_{e_2} \leftarrow x_{e_2} + \delta, \quad x_{e_3} \leftarrow x_{e_3} - \delta, \dots,$$

namely, alternately decrease and increase x -values of edges by δ ; see [Figure 6.2](#) for an example.

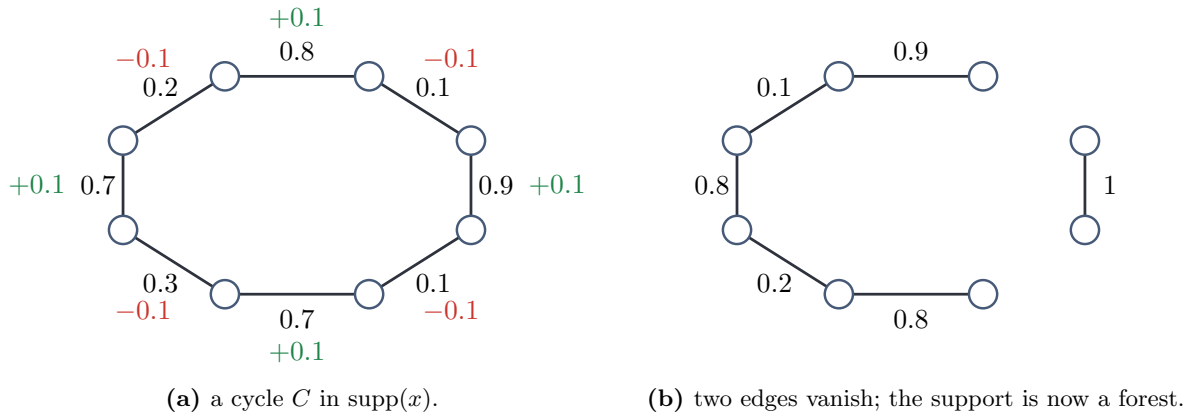


Figure 6.2: An illustration of cycle canceling with $\delta = 0.1$.

Since we are in a bipartite graph, the length of C must be even (recall that bipartite graphs are precisely those that have no odd cycles). This ensures that (i) the value of $\sum_{e \ni w} x_e$ remains the same for every vertex $w \in L \cup R$ in the graph since we increased one edge of w and decreased another one by the same value; and (ii) value of $\sum_{e \in E} x_e$ remains the same by the same reasoning. Moreover, support of x is now at least one edge smaller because the original edge e we started with now has $x_e = 0$.

Thus, through this transformation, we only *shrink* the support of x without decreasing the objective value. As the support is finite, this process has to end eventually and at that point, we can no longer have any cycles inside it.

Step two: rounding forests. Now that there are no cycles left in the support of x , we have that $\text{supp}(x)$ is a forest (namely, a graph without cycles). Any part of this support that is integral has to be disjoint from the rest because integral parts can only form a matching and thus vertices have degree exactly one (they form a collection of vertex-disjoint edges).

Let F be the fractional part of the support. Pick an arbitrary leaf-node in the forest F (namely, a vertex of degree one). Let v be this leaf-node, u be its parent, and $e = (u, v)$ be the edge between them. Let $x_e = 1$ and reduce the x -value of all other edges incident on u to become 0. Firstly, since v was a leaf-node and thus had no other edges, this transformation does not make x infeasible. Secondly, the total value of x incident on vertex u previously was at most 1 (to ensure feasibility of the original x) and by this change it has become exactly 1. Thus, we did not reduce the value of x at all. Finally, all edges incident on u have integral values and thus are removed from F . We can continue this process and at every step we obtain a feasible solution of the same value or higher and eventually transform the entire F into an integral solution.

Letting y be this integral solution, which has value at least that of x concludes the proof. $\square$

Remark. In general, rounding a fractional solution to an integral one does not necessarily need to be “lossless”. Finding the best rounding scheme (even the best LP relaxation of a problem) is one of the main topics studied in *approximation algorithms*, and we will revisit this later in the book. The gap between the optimal ILP value and its LP relaxation is referred to as the **integrality gap**.

6.2.4 Another Application of Type (b): The Bipartite Vertex Cover Problem

Let $G = (L \sqcup R, E)$ be a bipartite graph. A **vertex cover** in G is a set of vertices S that *covers* all the edges, i.e., any edge has at least one endpoint in S . Let us write this LP relaxation for bipartite vertex cover:

$$\begin{aligned} & \min_{y \in \mathbb{R}^V} \quad \sum_{v \in V} y_v \\ & \text{subject to} \quad y_u + y_v \geq 1 \quad \forall (u, v) \in E, \\ & \quad \quad \quad y_v \geq 0 \quad \quad \quad \forall v \in L \sqcup R. \end{aligned}$$

As before, if we assume the values in this LP are integral, then we get a program for minimum vertex cover, because $y_v = 1$ can be interpreted as picking the vertex v in the solution, and the $y_u + y_v \geq 1$ constraints ensure that from every edge we are picking at least one vertex. Of course, in general, there is no reason this LP picks its solution integrally and thus the LP more accurately corresponds to the *fractional* vertex cover problem. However, we show that the optimum value of this LP is equal to the minimum vertex cover of G .

Proposition 6.6. *Any feasible solution $y \in \mathbb{R}^V$ of the LP for bipartite vertex cover can be rounded to a vertex cover of the input graph with size at most $\sum_{v \in V} y_v$.*

Proof. We assume without loss of generality that for all $v \in V$, $y_v \leq 1$; otherwise, we can reduce y_v without violating any constraint while further reducing the objective value.

Pick $\theta \in (0, 1)$ uniformly at random and define the sets:

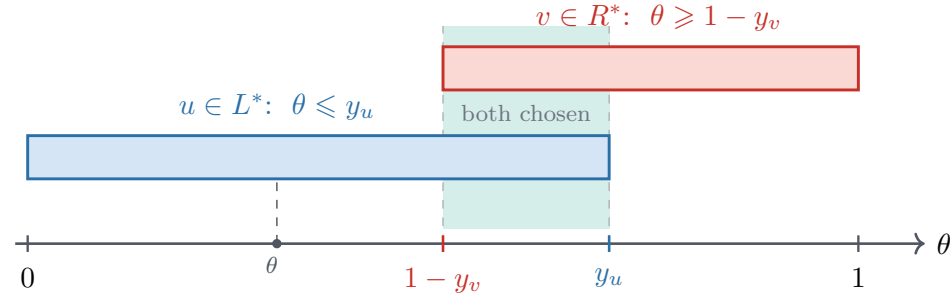
$$L^* = \{u \mid u \in L \text{ and } y_u \geq \theta\} \quad \text{and} \quad R^* = \{v \mid v \in R \text{ and } y_v \geq 1 - \theta\}.$$

We claim the set $L^* \cup R^*$ is a vertex cover of G because for an arbitrary edge $(u, v) \in E$ such that $u \in L, v \in R$, at least one of $u \in L^*$ or $v \in R^*$ is true. If $u \notin L^*$ and $v \notin R^*$, then we have $y_u < \theta$ and $y_v < 1 - \theta$, thus $y_u + y_v < 1$ which contradicts the feasibility of y for the LP. **Figure 6.3** illustrates this rounding on a single edge.

We now bound the size of the solution in expectation.

$$\begin{aligned}
 \mathbb{E}|L^* \cup R^*| &= \sum_{u \in L} \Pr(u \in L^*) + \sum_{v \in R} \Pr(v \in R^*) \\
 &= \sum_{u \in L} \Pr(y_u \geq \theta) + \sum_{v \in R} \Pr(y_v \geq 1 - \theta) && \text{(by the definition of } L^* \text{ and } R^*) \\
 &= \sum_{u \in L} y_u + \sum_{v \in R} y_v && (\theta \text{ is chosen uniformly at random}) \\
 &= \sum_{v \in V} y_v.
 \end{aligned}$$

This concludes the proof. □



Each range's length is the vertex's value, so $\Pr[u \in L^*] = y_u$ and $\Pr[v \in R^*] = y_v$; summing over all vertices gives $\mathbb{E}|L^* \cup R^*| = \sum_v y_v$.

Figure 6.3: An illustration of the randomized rounding of **Proposition 6.6**, shown for a single edge (u, v) with $u \in L$ and $v \in R$. Feasibility condition $y_u + y_v \geq 1$ ensures that any choice of θ leads to this edge being covered.

Remark. An important remark is in order here. While the above approach gives a randomized algorithm for rounding the value of the LP, a slightly more careful view of the same analysis implies that we can actually replace the random choice of $\theta \in (0, 1)$ with an *arbitrary* choice instead (for instance, simply take $\theta = 1/2$).

This is because in this algorithm, we always output a vertex cover. Our analysis shows that the expected size of this vertex cover is at most that of the minimum vertex cover. But then this implies that for *every* choice of θ , the size of the returned vertex cover should be equal to the minimum vertex cover size; otherwise, to “balance” the expectation, some choice of θ should result in a value which is even less than the minimum vertex cover size which is not possible^a.

^aFor a random variable Y , if $\mathbb{E}[Y] = \min_{y \in \text{supp}(Y)} y$, then we have $\Pr(Y = \min_{y \in \text{supp}(Y)} y) = 1$.

Exercises

Exercise 6.1. Definition 6.2 fixes a particular standard form for linear programs: the objective is a *maximization*, and every constraint has the form $\langle a, x \rangle \geq b$. Show that this entails no loss of generality by describing how to rewrite each of the following as an equivalent LP in the form of Definition 6.2:

- (a) a minimization objective, $\min \langle c, x \rangle$;
- (b) a constraint of the form $\langle a, x \rangle \leq b$;
- (c) an equality constraint $\langle a, x \rangle = b$.

Conclude that the diet LP, the ℓ_1 -regression LP, and the bipartite matching and vertex cover LP relaxations of this chapter are all instances of Definition 6.2.

Finally, some LP solvers instead demand the form in which *all variables are nonnegative* and all constraints are *equalities*. Show how to reduce to this form as well.

Exercise 6.2. Consider the regression problem of this chapter, but this time measured in the ℓ_∞ -norm. Given a matrix $A \in \mathbb{R}^{m \times n}$ with rows $a_1, \dots, a_m$ and a vector $b \in \mathbb{R}^m$, the ℓ_∞ -regression problem asks to solve

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_\infty = \min_{x \in \mathbb{R}^n} \max_{i \in [m]} |\langle a_i, x \rangle - b_i|.$$

Show that this problem can be written as a linear program, and determine how many variables and constraints your program has.

Exercise 6.3. Prove that solving integer linear programs in general is an NP-hard problem.

Exercise 6.4. The roundings in Proposition 6.5 and Proposition 6.6 both used that G is bipartite; here we see they can fail otherwise. Let $G = K_3$ be the triangle: three vertices $\{1, 2, 3\}$ with all three edges present.

- (a) Show that a maximum matching of K_3 has size 1, whereas the matching LP relaxation has optimal value $3/2$.
- (b) Show that a minimum vertex cover of K_3 has size 2, whereas the vertex cover LP relaxation has optimal value $3/2$.

Compute the *integrality gap* in each case. (In both parts, the fractional solution that assigns $1/2$ to every edge—respectively, to every vertex—is optimal for the LP.)