1 Introduction

A linear programming problem may be defined as an optimization problem of a linear function subject to linear constraints, whether they are equalities or inequalities. The standard form of a linear programming problem is given by the following:

**Definition 1** (The Standard Problem). Find an vector \( \mathbf{x} \in \mathbb{R}^n \), such that:

\[
c^t \mathbf{x} = c_1 x_1 + \cdots + c_n x_n,
\]

(1.1)

is minimized subject to the following main constraints:

\[
\begin{align*}
a_{11} x_1 + a_{12} x_2 &+ \cdots + a_{1n} x_n = b_1 \\
a_{21} x_1 + a_{22} x_2 &+ \cdots + a_{2n} x_n = b_2 \\
&\vdots \\
a_{m1} x_1 + a_{m2} x_2 &+ \cdots + a_{mn} x_n = b_m
\end{align*}
\]

(1.2)

( or \( A \mathbf{x} = \mathbf{b} \)),

where \( b_i \geq 0, 1 \leq i \leq m \); and the non-negativity constraints:

\[
x_i \geq 0, \quad i = 1, \ldots, n \quad (\text{or } \mathbf{x} \geq \mathbf{0}).
\]

(1.3)

Note that sometimes people maximize instead of minimizing it in the definition of a standard form; this is not an issue as one can always reverse the signs of \( c_i \) to transform the problem from one to the other. The requirement \( b_i \geq 0 \), however, is fairly standard in the field.

In the literature, people also use extensively the standard maximum form and the standard minimum form as follows:

**Definition 2** (The Standard Maximum Problem). Find an vector \( \mathbf{x} \in \mathbb{R}^n \), such that:

\[
c^t \mathbf{x} = c_1 x_1 + \cdots + c_n x_n,
\]

(1.4)

is maximized subject to the following main constraints:

\[
\begin{align*}
a_{11} x_1 + a_{12} x_2 &+ \cdots + a_{1n} x_n \leq b_1 \\
a_{21} x_1 + a_{22} x_2 &+ \cdots + a_{2n} x_n \leq b_2 \\
&\vdots \\
a_{m1} x_1 + a_{m2} x_2 &+ \cdots + a_{mn} x_n \leq b_m
\end{align*}
\]

(1.5)

( or \( A \mathbf{x} \leq \mathbf{b} \)),

and the non-negativity constraints:

\[
x_i \geq 0, \quad i = 1, \ldots, n \quad (\text{or } \mathbf{x} \geq \mathbf{0}).
\]

(1.6)
Definition 3 (The Standard Minimum Problem). Find an vector \( y \in \mathbb{R}^m \), such that:
\[
b' y = b_1 y_1 + \cdots + b_m y_m,
\]
is minimized subject to the following main constraints:
\[
y_1 a_{11} + y_2 a_{21} + \cdots + y_m a_{m1} x_n \geq c_1 \\
y_1 a_{12} + y_2 a_{22} + \cdots + y_m a_{m2} x_n \geq c_2 \\
\vdots \\
y_1 a_{1n} + y_2 a_{2n} + \cdots + y_m a_{mn} x_n \geq c_n
\]
(or \( y^t A \geq c^t \)),
\[
(1.8)
\]
and the non-negativity condition:
\[
y_i \geq 0, \quad i = 1, \ldots, m \quad (\text{or} \quad y \geq 0).
\]
(1.9)

Later we will see that any linear programming problem can be put into the standard form, or into either the standard maximum form or the standard minimum form. We’ll try to build algorithms to solve the linear programming problem based on the standard form, but many applications are more conveniently described in the standard maximum/minimum forms.

At the moment, let us have some visual understanding of the standard maximum problem in the case \( n = 2 \). In the \((x_1, x_2)\)-plane, the constraints altogether (both the main ones and the non-negativity ones) define a polygon that is composed of all points that satisfy these constraints. Such points are called feasible solutions and this polygon region is called the constraint set. For example, suppose we have the following three main constraints:
\[
x_1 + 2x_2 \leq 4 \\
4x_1 + 2x_2 \leq 12 \\
-x_1 + x_2 \leq 1
\]
Then the polygon region is depicted in Figure \[1\]. Now we consider maximizing the following linear function:
\[
x_1 + x_2.
\]
Geometrically the maximum problem becomes finding a straight line with slope \(-1\) as far away from the origin to the right as possible, such that its intersection with the shaded area is not empty. Clearly, in this case the solution is illustrated by the dashed line passing the vertex A, which is also the unique maximizer.

In another scenario, however, if the objective function is set to \( x_1 + 2x_2 \), the dashed line we would have drawn will be parallel to the line \( AB \) and clearly all the points on this edge of the shaded region solve the maximization problem. In reality we may encounter different scenarios including: (1) there is no feasible point at all – in this case the linear programming problem is said to be infeasible, (2) the standard maximum (resp., minimum) problem is feasible but the objective function can assume arbitrarily large positive (resp., negative) values at feasible points – this linear programming problem is said to be unbounded, and (3) otherwise, the problem is said to be bounded and its solutions are called the optimal feasible solutions.

Clearly, if the constrained set is non-empty and bounded, then the linear programming problem itself is feasible and bounded. In practice, however, the algorithm should be able to tell us whether the problem is feasible or not, and bounded or not if it is indeed feasible.

The linear programming problem is widely used in business and economics, for example it can be used to model logistic arrangement, resource allocation, and scheduling, etc.
Example 1 (The Transportation Problem). Suppose we have $M$ warehouses $W_1, \ldots, W_M$ and $N$ supermarkets $S_1, \ldots, S_N$. For a certain commodity, each warehouse $W_i$ possesses an amount of $s_i$ units of this commodity and the market $S_j$ must receive at least $d_j$ units for uninterrupted operation. The cost associated with shipping one unit of this commodity from $W_i$ to $S_j$ is $c_{ij}$. Then the problem is to meet the market requirements with the minimal transportation costs.

If we denote the number of the commodity to be shipped from $W_i$ to $S_j$ by $x_{ij}$, then for these numbers to have physical meaning at all the non-negative constraint clearly holds for $x_{ij}$, $1 \leq i \leq M, 1 \leq j \leq N$. Now we have two sets of constraints – from the warehouse side:

$$\sum_{j=1}^{N} x_{ij} \leq s_i, \quad \forall 1 \leq i \leq M,$$

and on the supermarket side:

$$\sum_{i=1}^{M} x_{ij} \geq d_j, \quad \forall 1 \leq j \leq N.$$

Finally, the total cost is given by:

$$\sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} x_{ij},$$

the objective function to be minimized. Now the problem is almost formulated in the standard minimum form with $m = MN$ variables and $n = N + M$ main constraints, except that the constraints on the warehouse side have the wrong direction. But no worry, they can be easily made to the
equivalent form:
\[
\sum_{j=1}^{N} (-x_{ij}) \geq -s_i, \quad \forall 1 \leq i \leq M.
\]

**Example 2** (The Diet Problem). We have \(m\) different foods \(F_1, \ldots, F_m\) that supply \(n\) nutrients \(N_1, \ldots, N_n\), which are essential for good health. Let \(c_i\) be the minimum daily amount of \(N_i\) and let \(p_j\) be the price per unit \(F_j\). Finally, suppose \(F_j\) contains \(a_{ji}\) units of \(N_i\). The problem is then to find the combination of foods so that the daily requirement is met with the least total price.

Let us buy \(y_j\) units of \(F_j\) daily, then they are of course non-negative. The daily requirement for the nutrients is formulated as:
\[
\sum_{j=1}^{m} y_j a_{ji} \geq c_i, \quad \forall 1 \leq i \leq n,
\]
and the total price for such a take-in per day is:
\[
\sum_{j=1}^{m} p_j y_j,
\]
the objective function to be minimized. Without any modification, we have a problem formulated in the standard minimum form.

**Example 3** (The Weight-Loss Plan). A more interesting problem is the weight-loss problem, especially for someone like me who cannot stop eating until feeling full. Again, we assume there are \(m\) different foods \(F_1, \ldots, F_m\) that supply \(n\) nutrients \(N_1, \ldots, N_n\) as before, and retain the daily nutrient requirement and the numbers \(c_i\) and \(a_{ji}\) in this model. Furthermore, we assume each unit of \(F_j\) creates \(C_j\) calories once consumed. Finally instead of the price, however, we consider the volume these foods will take in our stomach. Suppose each unit \(F_j\) fills \(v_j\) units of space in the stomach; and we start to feel full when a total of \(r\) units of space are filled. We also assume that the maximum volume is \(s\). Then the problem is how to plan our meals so that the daily nutrient requirement is met, and we feel full with the minimum amount of calories taken in.

Let \(y_j\) be the units of \(F_j\) consumed daily, then the nutrient constraints remain the same as in the previous example. However, we have in addition the following two constraints:
\[
\sum_{j=1}^{m} v_j y_j \geq r,
\]
so that we can feel full and stop eating and
\[
\sum_{j=1}^{m} v_j y_j \leq s,
\]
so that the stomach is not stuffed. The objective function is then the total calories consumed:
\[
\sum_{j=1}^{m} C_j y_j,
\]
which is to be minimized. Clearly, once we multiply both sides of the not-over-stuff constraint by \(-1\), a standard minimum problem with \(m\) variables and \(n + 2\) constraints is obtained.
Example 4 (The Optimal Assignment Problem). Suppose there are \( N \) workers available for \( M \) tasks. The worker \( i \) produces \( a_{ij} \) units of value working on the task \( j \) for one day; here \( 1 \leq i \leq N \) and \( 1 \leq j \leq M \). Only one person is allowed at a task at one time. The target is to assign the workers properly so that maximum value is produced.

Let \( x_{ij} \) be the portion of the day the worker \( i \) spends on the task \( j \). Then from the worker’s perspective we have:

\[
\sum_{j=1}^{M} x_{ij} \leq 1, \quad \forall 1 \leq i \leq N
\]

since one cannot spent more than 100% of the time; and on the task’s side

\[
\sum_{i=1}^{N} x_{ij} \leq 1, \quad \forall 1 \leq j \leq M
\]

that is only one worker is allowed on a task at one time. Clearly all \( x_{ij} \) are non-negative, and the target is to maximize:

\[
\sum_{i=1}^{N} \sum_{j=1}^{M} a_{ij} x_{ij},
\]

hence the problem is a standard maximum problem with \( n = MN \) and \( m = M + N \).

Example 5 (\( L_\infty \) Curve Fitting). Suppose there is a set of \( M \) data points \((x_i, y_i)\), and we want to find a straight line \( y(x) = ax + b \) such that:

\[
\max_{1 \leq i \leq M} |a x_i + b - y_i|
\]

is minimized.

This last example is much less straightforward from the previous ones in terms of a linear programming problem. In fact, if we minimize the \( L_2 \)-norm of the fitting error vector the solution is given by the least squares problem. To see that the \( L_\infty \) curve fitting is indeed a linear programming problem, we reformulate the minimization part as: Find \( a, b, \) and \( \delta \) such that:

\[
|a x_i + b - y_i| \leq \delta, \quad \forall 1 \leq i \leq M,
\]

and

\[
\delta
\]

is minimized. To understand why these two problems are equivalent to each other is very important – it is a basic technique to transform a problem to a linear programming one if possible. Let’s give a brief proof here. The two problems are formulated as:

\[
\min_{a,b} \max_{1 \leq i \leq M} |a x_i + b - y_i|, \quad (1.10)
\]

and

\[
\min_{a,b,\delta} \delta, \quad \mathcal{S} \overset{\text{def}}{=} \{(a,b,\delta) \in \mathbb{R}^3 : |a x_i + b - y_i| \leq \delta, \forall 1 \leq i \leq M\}. \quad (1.11)
\]
First we look at (1.10) \(\Rightarrow\) (1.11), and suppose \((a^*, b^*)\) solves the former. Defining \(\delta^* = \max_{1 \leq i \leq M} |a^*x_i + b^* - y_i|,\) then it is clear that \((a^*, b^*, \delta^*) \in S.\) If there exists \((a', b', \delta') \in S\) such that \(\delta' < \delta^*,\) by the definition of \(S\) for this pair \((a', b'):\)

\[
\max_{1 \leq i \leq M} |a'x_i + b' - y_i| < \delta^*,
\]

contradiction! Hence \((a^*, b^*, \delta^*)\) solves (1.11).

Proving the other direction proceeds as follows: Let \((a^*, b^*, \delta^*)\) be a solution to (1.11), we immediately see that:

\[
\max_{1 \leq i \leq M} |a^*x_i + b^* - y_i| = \delta^*.
\]

Hence we have:

\[
\min \max_{a, b \in \mathbb{R}} a x_i + b - y_i \leq \max_{1 \leq i \leq M} |a^*x_i + b^* - y_i| = \delta^*.
\]

Now let \((a', b')\) be arbitrary and define \(\delta' = \max_{1 \leq i \leq M} |a'x_i + b' - y_i|\). Then \((a', b', \delta') \in S\) and by assumption, \(\delta' \geq \delta^*;\) hence

\[
\min \max_{a, b \in \mathbb{R}} a x_i + b - y_i \geq \delta^*.
\]

Combining the two, we proved (1.10).

As a final remark here, a strict mathematical proof also requires considering the solvability of the two problems and show their equivalence. The technique for this part is very similar to what we have just now.

Continuing with our example, now we have a minimization problem as follows – Minimize:

\[
0 \times a + 0 \times b + 1 \times \delta
\]

subject to the constraints:

\[
-x_i a - b + y_i + \delta \geq 0, \quad 1 \leq i \leq M; \\
x_i a + b - y_i + \delta \geq 0, \quad 1 \leq i \leq M.
\]

This is almost a standard minimum problem for \(m = 3\) \((a, b, \delta)\) and \(n = 2M,\) except that we only have the non-negativity on \(\delta\) but not on \(a\) and \(b.\) To fix the last issue, we write:

\[
a = a_1 - a_2, \quad a_1, a_2 \geq 0; \quad b = b_1 - b_2, \quad b_1, b_2 \geq 0.
\]

Eventually a standard minimum problem with \(m = 5\) variables \((a_1, a_2, b_1, b_2, \delta)\) and \(n = 2M\) constraints is obtained.

At the end of this section, we illustrate how to convert any linear programming problem to the standard form (Definition 1) or one of the other two standard forms (Definition 2 and Definition 3). Here the linear programming problem is defined as maximizing or minimizing a linear function subject to linear constraints, including both equality ones and inequality ones.

The recipes converting a general linear programming problem to a standard maximum/minimum form include:

- An equality constraint is usually removed by solving this equation for some variable. For example if \(\sum a_{ij}x_j = b_i\) and without loss of generality \(a_{i1} \neq 0,\) we compute \(x_1 = a_{i1}^{-1}(b_i - \sum_{j \neq 1} a_{ij}x_j)\) and plug it into any other formulas.
• A “no greater than” constraint $\sum_i a_{ij} x_j \leq b_i$ can be converted to a “no less than” constraint by multiplying both sides by $-1$: $\sum_i (-a_{ij}) x_j \geq -b_i$; and vice versa.

• If some variable $x$ is not restricted to be non-negative, we write $x = x_1 - x_2$ and impose the non-negativity on $x_1$ and $x_2$.

The strategies converting a general problem to the standard form given by Definition 1 are the following:

• Negate the signs of the coefficient in the objective function if the original problem is maximization.

• Multiply both sides of the inequality constraints by $-1$ if the right hand sides are negative.

• For any “no greater than” constraint $\sum_i a_{ij} x_j \leq b_i$, introduce the slack variable $\epsilon_i \geq 0$ such that $\sum_i a_{ij} x_j + \epsilon_i = b_i$.

• For any “no less than” constraint $\sum_i a_{ij} x_j \geq b_i$, introduce the surplus variable $\epsilon_i \geq 0$ such that $\sum_i a_{ij} x_j - \epsilon_i = b_i$.

• For any unconstrained variable $x$, write it as $x = x_1 - x_2$ for $x_1, x_2 \geq 0$.

Sometimes we call $\epsilon_i$ of the third and the fourth cases uniformly as the “slack variable”. Another way to deal with the unconstrained variable is to use an equality constraint to remove this variable entirely from the problem – this procedure may not always be possible.

2 Basic Analysis of the Standard Problem

Let us consider the standard problem:

$$\begin{align*}
\text{minimize} & \quad c^T x \\
\text{such that} & \quad Ax = b \geq 0 \\
& \quad x \geq 0.
\end{align*}$$

Here $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $b \geq 0$. First we take a look at the equality constraints $Ax = b$. If the row vectors of $A$ are linearly dependent, say:

$$\alpha_{i_1} \tilde{a}_{i_1} + \cdots + \alpha_{i_k} \tilde{a}_{i_k} = 0,$$

where $\tilde{a}_j \in \mathbb{R}^{1 \times n}$ is the $j$-th row of $A$, $\alpha_{i_1}, \ldots, \alpha_{i_k} \neq 0$, and $1 \leq i_1 < i_2 < \cdots < i_k \leq m$; then there are two scenarios. First, if $\alpha_{i_1} b_{i_1} + \cdots + \alpha_{i_k} b_{i_k} \neq 0$ the constraints contradict with themselves and the problem is infeasible. Second, if $\alpha_{i_1} b_{i_1} + \cdots + \alpha_{i_k} b_{i_k} = 0$ we can always remove at least one constraint without changing the solution of the problem at all. For this reason, in the following analysis we always assume that: $A$ is full rank and $\text{rank} A = m$. This assumption implies that $m \leq n$, i.e., there are more variables than the constraints.
At a first glance, this assumption is not realistic because in many of our previous examples, we actually have more constraints than variables. However, those constraints are inequality ones; and if we introduce a slack variable for each of them, there will be more variables than constraints.

Ignoring the non-negativity requirement for a moment, \( Ax = b \) alone defines an affine space that contains all feasible solutions. Recalling our previous example of two variables (Figure 1), if the problem is converted to a standard form there will be five variables and three constraints – the three new variables are the slack ones corresponding to the constraints. The solution to the problem occurs at the vertex \( A \), where exactly two slack variables are zero. It is not surprising that the number of zeroes equal the difference between the number of constraints and the number of variables (2 = 5 − 3); this is actually a universal result for linear programming problem and the foundation for the simplex method.

To tackle the idea, let us introduce some terminologies.

- Let \( x \) satisfy the constraint \( Ax = b \), then it is a basic solution if and only if at least \( n - m \) of its components are zero; the rest \( m \) components of \( x \) (they may also be zero) are called the basic variables. If \( B \) is an \( m \times m \) sub-matrix of \( A \) that interacts with the non-zero components of \( x \) in the product \( Ax \), we say \( x \) is a basic solution w.r.t. \( B \).
- If one or more of the basic variables are zero, \( x \) is said to be a degenerate basic solution.
- If a basic solution \( x \) is also feasible (i.e., \( x \geq 0 \)), it is called a basic feasible solution; and if it is also a degenerate basic solution, it is called a degenerate basic feasible solution.

The fundamental theorem of linear programming is stated as below:

**Theorem 2.1 (Fundamental Theorem of Linear Programming).** Consider the standard linear programming problem \((2.1)\). Suppose \( A \) is \( m \times n \) with rank \( m \), then:

(i) if there is a feasible solution, there is a basic feasible solution.

(ii) if there is an optimal feasible solution, there is an optimal basic feasible solution.

Before proving this theorem, let us go back to our 2D example before. When we try to maximize \( x_1 + x_2 \), we see that the vertex \( A \) is the unique optimal feasible solution and it is also a basic solution because two slack variables corresponds to the two constraint lines intersecting at \( A \) are zero. When we instead maximize \( x_1 + 2x_2 \), all the points on the line segment \( AB \) are optimal feasible solutions, which include both basic ones (\( A \) and \( B \)) and non-basic ones.

**Proof.** We denote the column vectors of \( A \) by \( a_1, a_2, \ldots, a_n \). Let us first consider (i). We define the subset of natural numbers:

\[ K = \{ k \in \mathbb{N} : \exists x, \text{ s. t. } Ax = b \text{ and exactly } k \text{ components of } x \text{ are non-zero.} \} \]

If there is a feasible solution, \( K \neq \emptyset \) and we can find an \( x^* \) and \( k^* \) such that \( k^* \) is the minimum of \( K \). If \( k^* \leq m \) then \( x^* \) is a basic feasible solution; now we suppose \( k^* > m \). Without loss of generality, let us suppose \( x_1^*, \ldots, x_{k^*}^* \) are non-zero, hence we have:

\[ x_1^* a_1 + \cdots + x_{k^*}^* a_{k^*} = b. \]
Because $k^* > m$, these $k^*$ column vectors of $A$ are linearly dependent and we can find not-all-zero real numbers $\alpha_1, \ldots, \alpha_{k^*}$, where at least one $\alpha_i$ is positive, such that:

$$\alpha_1 a_1 + \cdots + \alpha_{k^*} a_{k^*} = 0.$$ 

Hence for any $\varepsilon > 0$, we have:

$$(x_1^* - \varepsilon \alpha_1) a_1 + \cdots + (x_{k^*}^* - \varepsilon \alpha_{k^*}) a_{k^*} = b.$$

Defining $\alpha = [\alpha_1 \alpha_2 \cdots \alpha_{k^*} 0 \cdots 0]^t$, the former equality states that $x^\varepsilon = x^* - \varepsilon \alpha$ solves $Ax = b$; and $x^\varepsilon$ has at most $k^*$ non-zero components. For small enough $\varepsilon$, $x^\varepsilon$ has exactly the same number of non-zero components as $x^*$; but if we increase the value $\varepsilon$ gradually, one term of $x^*$ will become smaller and smaller until it reaches zero while all other components remain non-negative. Formally speaking, let us set $\varepsilon$ to:

$$\varepsilon_0 = \min_{i: 1 \leq i \leq k^*, \alpha_i > 0} \frac{x_i^*}{\alpha_i},$$

then $x^{\varepsilon_0}$ is non-negative hence feasible, and it has at most $k^* - 1$ non-zero components, contradiction!

Next we consider (ii). The proof herein is almost identical to the case of (i), except that we need to verify $c^t x^\varepsilon = c^t x^*$ for all $\varepsilon$. Note here $x^*$ is an optimal feasible solution with the smallest possible number of non-zero components $k^*$. We already saw that for all $|\varepsilon|$ small (including negative ones), $x^\varepsilon$ is feasible hence we must have:

$$c^t x^* \leq c^t x^\varepsilon = c^t x^* - \varepsilon c^t \alpha$$

for all these $\varepsilon$. Hence $c^t \alpha$ and in particular $x^{\varepsilon_0}$ is also an optimal feasible solution with less non-zero components than $x^*$, which is a contradiction. \qed

A direct consequence of Theorem 2.1 is that we can confidently search for the optimal feasible solution in the basic solutions. That is, if an optimal feasible solution exists, we will be able to find all its non-zero components by solving

$$B \tilde{x} = b,$$

where $B$ is an $m \times m$ sub-matrix of $A$; and there are only $\frac{n!}{m!(n-m)!}$ such sub-matrices! This brute force method is clearly not practical unless in very special situations (such as $m = n$). And in the next lecture we’ll talk about one of the most popular method to solve the problem (2.1) efficiently, namely the simplex method.

**Acknowledgement**

Many examples of this and next a few lecture notes are given by Thomas S. Furguson in his excellent notes “Linear Programming, A Concise Introduction” (available at https://www.math.ucla.edu/~tom/LP.pdf).