Game Theory

Lecture 08

The Simplex Method for Solving Linear Programs

Introduction

- If you only remember one thing about linear programming, make it this:

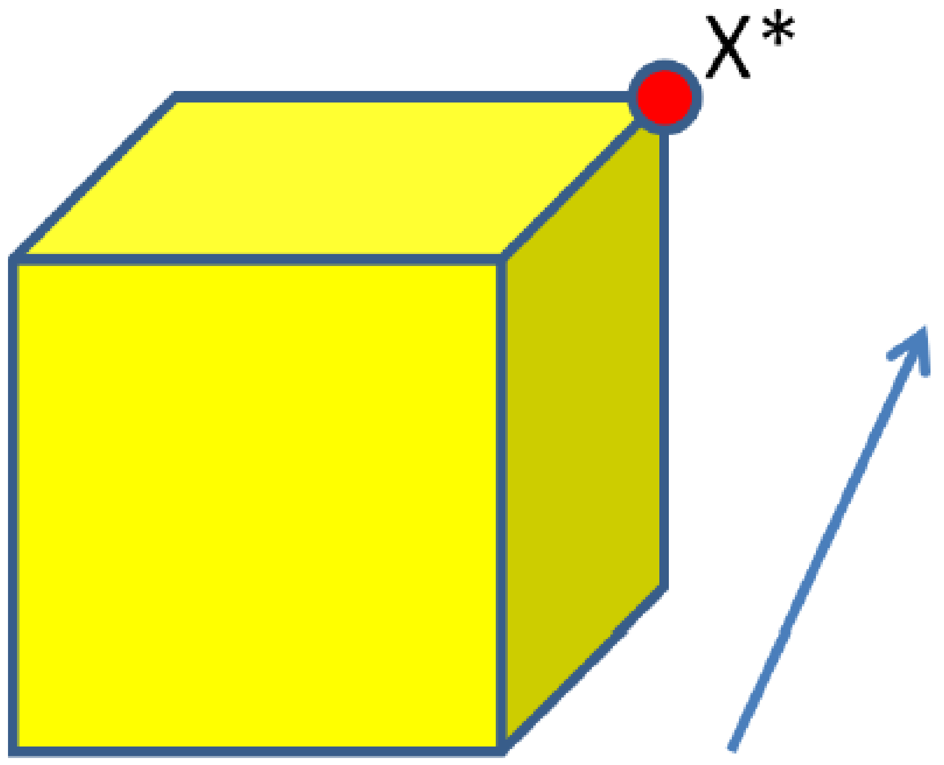
Linear programs can be solved efficiently, in both theory and practice.

  - By “in theory," we mean that linear programs can be solved in polynomial time in the worst-case. By “in practice," we mean that commercial solvers routinely solve linear programs with input size in the millions. (Warning: the algorithms used in these two cases are not necessarily the same.)

- In 1947 George Dantzig developed both the general formalism of linear programming and also the first general algorithm for solving linear programs, the ***simplex method***.
- Amazingly, the simplex method remains the dominant paradigm today for solving linear programs.

Geometry

- In Lecture #7 we developed geometric intuition about what it means to solve a linear program, and one of our findings was that there is always an optimal solution at a vertex (i.e., corner") of the feasible region.



- This observation implies a **finite** (but bad) algorithm for linear programming. (This is not trivial, since there are an infinite number of feasible solutions.)
  - ➢ The reason is that every vertex satisfies at least *n* constraints with equality (where *n* is the number of decision variables).
  - ➢ The finite algorithm is then:
    - Enumerate all (finitely many) subsets of n linearly independent constraints, check if the unique point of R^n that satisfies all of them is a feasible solution to the linear program, and remember the best feasible solution found in this way.

geometric idea of simplex

- Input: Given $(f, \texttt{Opt}, C)$, and given some start "vertex" $x \in K(C) \subseteq \mathbb{R}^n$.
  (Never mind, for now, that we have no idea how to find $x \in K(C)$ -or even whether C is Feasible!- let alone a "vertex" x.)

  While (x has some "neighbor vertex", $x' \in K(C)$,
  such that $f(x') > f(x)$)

  - Pick such a neighbor x'. Let $x := x'$.
  - (If neighbor at "infinity", Output: "Unbounded".)

  Output: $x^* := x$, and $f(x^*)$ is optimal value.

Question: Why should this work? Why don't we get "stuck" in some "local optimum"?

Key reason: The region $K(C)$ is convex, meaning if $x, y \in K(C)$ then every point z on the "line segment" between x and y is also in $K(C)$. (Recall: K is convex iff $x, y \in K \Rightarrow \lambda x + (1 - \lambda)y \in K$, for $\lambda \in [0, 1]$.) On a convex region, a "local optimum" of a linear objective is always the "global optimum".

Ok. The geometry sounds nice and simple. But realizing it algebraically is not a trivial matter!

LP's in "Primal Form"

Using the simplification rules from the last lecture, we can convert any LP into the following form:

Maximize $c_1\, x_1 + c_2\, x_2 + \ldots + c_n\, x_n + d$

Subject to:

$$\begin{array}{ll} a_{1,1}\, x_1 + a_{1,2}\, x_2 + \ldots + a_{1,n}\, x_n & \leq b_1 \\ a_{2,1}\, x_1 + a_{2,2}\, x_2 + \ldots + a_{2,n}\, x_n & \leq b_2 \\ \ldots & \ldots \\ \ldots & \ldots \\ a_{m,1}\, x_1 + a_{i,2}\, x_2 + \ldots + a_{m,n}\, x_n & \leq b_m \end{array}$$

$x_1, \ldots, x_n \geq 0$

slack variables

We can add a "slack" variable y_i to each inequality, to get equalities:

Maximize $c_1\, x_1 + c_2\, x_2 + \ldots + c_n\, x_n + d$
Subject to:

$$\begin{aligned}
&a_{1,1}\, x_1 + a_{1,2}\, x_2 + \ldots + a_{1,n}\, x_n + y_1 &&= b_1\\
&a_{2,1}\, x_1 + a_{2,2}\, x_2 + \ldots + a_{2,n}\, x_n + y_2 &&= b_2\\
&\ldots &&\ldots\\
&a_{m,1}\, x_1 + a_{i,2}\, x_2 + \ldots + a_{m,n}\, x_n + y_m &&= b_m
\end{aligned}$$

$$x_1, \ldots, x_n \geq 0; \quad y_1, \ldots, y_m \geq 0$$

The two LPs are "equivalent". (Explanation.)
The new LP has some particularly nice properties:

1. Every equality constraint C_i has at least one variable on the left with coefficient 1 and which doesn't appear in any other equality constraint.
2. Picking one such variable for each equality, we obtain a set of m variables B called a **Basis**. An obvious basis above is $B = \{y_1, \ldots, y_m\}$.
3. Objective $f(x)$ involves only non-Basis variables.

Let us call an LP in such a form a "dictionary".

Basic Feasible Solutions

Rewrite our dictionary (renaming "y_i", "x_{n+i}") as:

Maximize $c_1\, x_1 + c_2\, x_2 + \ldots + c_n\, x_n + d$
Subject to:

$$
\begin{aligned}
x_{n+1} &= b_1 - a_{1,1}\, x_1 - a_{1,2}\, x_2 - \ldots - a_{1,n}\, x_n \\
x_{n+2} &= b_2 - a_{2,1}\, x_1 - a_{2,2}\, x_2 - \ldots - a_{2,n}\, x_n \\
\ldots & \quad \ldots \\
x_{n+m} &= b_m - a_{m,1}\, x_1 - a_{i,2}\, x_2 - \ldots - a_{m,n}\, x_n
\end{aligned}
$$

$$x_1, \ldots, x_{n+m} \geq 0$$

Suppose, somehow, $b_i \geq 0$ for all $i = 1, \ldots, m$.
Then we have what's called a "*feasible dictionary*"
and a feasible solution for it, namely,
let $x_{n+i} = b_i$, for $i = 1, \ldots, m$, and
let $x_j = 0$, for $j = 1, \ldots, n$. The value is $f(0) = d$!

Call this a *basic feasible solution*(BFS),with basis B.

Geometry: A BFS corresponds to a "vertex".
(But different Bases B may yield the same BFS!)

Question: How do we move from one BFS with basis B to a "neighboring" BFS with basis B'?

Answer: Pivoting!

Pivoting

Suppose our current dictionary basis (the variables on the left) is $B = \{x_{i_1}, \ldots, x_{i_m}\}$, with x_{i_r} the variable on the left of constraint C_r.

The following pivoting procedure moves us from basis B to basis $B' := (B \setminus \{x_{i_r}\}) \cup \{x_j\}$.

Pivoting to add x_j and remove x_{i_r} from basis B:

1. Assuming C_r involves x_j, rewrite C_r as $x_j = \alpha$.
2. Substitute α for x_j in all other constraints C_l, obtaining C'_l.
3. The new constraints C', have a new basis: $B' := (B \setminus \{x_{i_r}\}) \cup \{x_j\}$.
4. Also substitute α for x_j in $f(x)$, so that $f(x)$ again only depends on variables not in the new basis B'.

This new basis B' is a "possible neighbor" of B.

However, not every such basis B' is eligible!

sanity checks for pivoting

To check *eligibility* of a pivot, we have to make sure:

1. The new constants b'_i remain ≥ 0, so we retain a "feasible dictionary", and thus B' yields a BFS.
2. The new BFS must improve, or at least must not decrease, the value $d' = f(0)$ of the new objective function. (Recall, all non-basic variables are set to 0 in a BFS, thus $f(BFS) = f(0)$.)
3. We should also check for the following situations:
   - Suppose all non-basic variables involved in $f(x)$ have negative coefficients. Then any increase from 0 in these variables will decrease the objective. We are thus (it turns out) at an optimal BFS x^*. Output: Optimal solution: x^* and $f(x^*) = f(0) = d'$.
   - Suppose there is a non-basic variable x_j in $f(x)$ with coefficient $c_j > 0$, and such that the coefficient of x_j in every constraint C_r is also ≥ 0. Then we can increase x_j, and the objective value, to "infinity" without violating any constraints. So, Output: "Feasible but Unbounded".

finding and choosing eligible pivots

- In principle, we could exhaustively check the sanity conditions for eligibility of all potential pairs of entering and leaving variables. There are at most $(n * m)$ candidates.

- But, there are <u>much more efficient</u> ways to choose pivots, by inspection of the coefficients in the dictionary.

- We can also efficiently choose pivots according to lots of additional criteria, or <u>pivoting rules</u>, such as, e.g., "most improvement in objective value", etc.

- There are many such "rules", and it isn't clear *a priori* what is "best".

The Simplex Algorithm

Dantzig's Simplex algorithm can be described as follows:

Input: a feasible dictionary;

Repeat

1. Check if we are at an optimal solution, and if so, Halt and output the solution.
2. Check if we have an "infinity" neighbor, and if so Halt and output "Unbounded".
3. Otherwise, choose an eligible pivot pair of variables, and Pivot!

Fact If this halts the output is correct: an output solution is an optimal solution of the LP.

Oops! We could cycle back to the same basis for ever, never strictly improving by pivoting.

There are several ways to address this problem (See slide 25)

Simplex Algorithm on a Toy Example

We will now discuss the best-known algorithm (really, a family of algorithms) for solving a program, the *simplex algorithm*.

We will demonstrate it on an example.

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ 4x_1 + 2x_2 \leq 16 \\ x_1 + 2x_2 \leq 8 \\ x_1 + x_2 \leq 5 \\ x_1 \geq 0; x_2 \geq 0\end{array}}$$

To run the simplex algorithm, we introduce a slack variable w_i for each constraint i, so that we can rewrite the linear program in equality form, as follows:

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ w_1 = 16 - 4x_1 - 2x_2 \\ w_2 = 8 - x_1 - 2x_2 \\ w_3 = 5 - x_1 - x_2 \\ w_1, w_2, w_3, x_1, x_2 \geq 0\end{array}}$$

If we set $x_1 = x_2 = 0$, we get a feasible solution to this linear program. (Of course, this is not the case for every linear program, and we will see what to do if this is not the case later on.)

Our goal is to improve this solution.

Simplex Algorithm on a Toy Example

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ w_1 = 16 - 4x_1 - 2x_2 \\ w_2 = 8 - x_1 - 2x_2 \\ w_3 = 5 - x_1 - x_2 \\ w_1, w_2, w_3, x_1, x_2 \geq 0\end{array}}$$

If we increase either x_1 or x_2, then the objective value will increase.

Let us start by increasing x_1.

At some point, one of the constraints will be violated—that is, one of the slack variables will become negative.

Specifically, if we increase x_1 to 4, then the first constraint $4x_1 + 2x_2 \leq 16$ will be just barely satisfied, that is, w_1 will be 0, so we cannot increase x_1 further. (The other constraints are still satisfied at this point.)

The objective value at this current solution of $x_1 = 4, x_2 = 0$ is 12, a good start but we are not yet at optimality.

Simplex Algorithm on a Toy Example

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ w_1 = 16 - 4x_1 - 2x_2 \\ w_2 = 8 - x_1 - 2x_2 \\ w_3 = 5 - x_1 - x_2 \\ w_1, w_2, w_3, x_1, x_2 \geq 0\end{array}}$$

The key trick of the simplex algorithm is that at this point, we rewrite the linear program, changing the roles of some of the original and slack variables.

After we do so, the current solution will once again correspond to the origin.

Specifically, we remove x_1, whose value is no longer 0, from the objective and the right-hand sides of the equalities;

we replace it with an expression involving w_1, whose value is now 0.

Specifically, from the first constraint, we know that $w_1 = 16 - 4x_1 - 2x_2$, or equivalently, $x_1 = 4 - 0.25w_1 - 0.5x_2$.

We replace the first constraint with this new equality.

We also rewrite the objective as

$$3x_1 + 2x_2 = 3(4 - 0.25w_1 - 0.5x_2) + 2x_2 = 12 - 0.75w_1 + 0.5x_2.$$

We rewrite the second constraint and the third constraint

Simplex Algorithm on a Toy Example

This results in the following linear program, which is equivalent to our original linear program:

$$\boxed{\begin{array}{l} \textbf{maximize } 12 - 0.75w_1 + 0.5x_2 \\ \textbf{subject to} \\ x_1 = 4 - 0.25w_1 - 0.5x_2 \\ w_2 = 4 + 0.25w_1 - 1.5x_2 \\ w_3 = 1 + 0.25w_1 - 0.5x_2 \\ w_1, w_2, w_3, x_1, x_2 \geq 0 \end{array}}$$

Our current solution consists of setting $w_1 = 0, x_2 = 0$.

Because both of these are 0, the other values are easy to read off: the current objective value is 12, x_1 is 4, w_2 is 4, and w_3 is 1.

We call a linear program written in this way a *dictionary*; the left-hand side variables are called the *basic* variables, and the right-hand side variables (which are set to 0 in the current solution) the *nonbasic* variables.

When we moved from the first dictionary to the second dictionary, we performed a *pivot*; in this pivot

x_1 was the *entering variable* (going from nonbasic to basic)

and w_1 was the *leaving variable* (going from basic to nonbasic).

Simplex Algorithm on a Toy Example

$$\begin{array}{l}\textbf{maximize } 12 - 0.75w_1 + 0.5x_2 \\ \textbf{subject to} \\ x_1 = 4 - 0.25w_1 - 0.5x_2 \\ w_2 = 4 + 0.25w_1 - 1.5x_2 \\ w_3 = 1 + 0.25w_1 - 0.5x_2 \\ w_1, w_2, w_3, x_1, x_2 \geq 0\end{array}$$

It is easy to see that we have not yet arrived at an optimal solution: the coefficient of x_2 in the objective is positive, meaning that by increasing x_2 we can increase the objective value.

In contrast, there is no sense in increasing w_1 because its coeffient in the objective is negative.

So, we will increase x_2 as much as we can without violating a constraint.

If we increase x_2 to 2, then w_3 will be equal to 0 (and the other two basic variables will still be positive.)

So, x_2 is our entering variable, and w_3 is our leaving variable.

We know that $w_3 = 1 + 0.25w_1 - 0.5x_2$, or equivalently,

$$x_2 = 2 + 0.5w_1 - 2w_3,$$

so we replace the last constraint with this expression.

We also use this expression to replace occurrences of x_2 in the objective and the right-hand sides of the constraints.

Simplex Algorithm on a Toy Example

We obtain:

$$\boxed{\begin{array}{l} \textbf{maximize } 13 - 0.5w_1 - w_3 \\ \textbf{subject to} \\ x_1 = 3 - 0.5w_1 + w_3 \\ w_2 = 1 - 0.5w_1 + 3w_3 \\ x_2 = 2 + 0.5w_1 - 2w_3 \\ w_1, w_2, w_3, x_1, x_2 \geq 0 \end{array}}$$

- Again, our current solution corresponds to setting the ***non-basic*** variables w_1 and w_3 to 0.
- We can easily read off that the current value of our solution is 13, and that $x_1 = 3$, $x_2 = 2$.
- The next step would be to increase either w_1 or w_3 to increase the objective.
  - ➢ However, the coefficient on both of these variables in the objective is negative, so there is no point to doing this: it will only decrease the objective.
  - ➢ So the simplex algorithm terminates.
  - ➢ In fact, this last ***dictionary*** gives a proof that our current solution is optimal: because w_1 and w_3 must be nonnegative, clearly the objective value can be at most 13, and our current solution achieves this.

A Richer Example

- To illustrate some additional phenomena involving the simplex algorithm, we now consider the following richer example:

$$\begin{aligned}
&\textbf{maximize } 4x_1 + 5x_2 + 4x_3 + 7x_4 + x_5 \\
&\textbf{subject to} \\
&x_1 + x_3 + x_4 \leq 1 \\
&x_1 + x_2 + x_4 \leq 1 \\
&x_2 + x_3 \leq 1 \\
&x_4 + x_5 \leq 1 \\
&x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}$$

- We now write this linear program in equality form:

$$\begin{aligned}
&\textbf{maximize } 4x_1 + 5x_2 + 4x_3 + 7x_4 + x_5 \\
&\textbf{subject to} \\
&w_1 = 1 - x_1 - x_3 - x_4 \\
&w_2 = 1 - x_1 - x_2 - x_4 \\
&w_3 = 1 - x_2 - x_3 \\
&w_4 = 1 - x_4 - x_5 \\
&w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}$$

- Again, this is a feasible dictionary, in the sense that setting all of the ***non-basic*** variables to 0 corresponds to a feasible solution.

- Now we need to choose an ***entering*** variable.
  - All of the x_j have a positive coefficient in the objective, so we can choose any one of them.
  - A natural heuristic is to choose the one with the greatest coefficient, as we want to improve the objective as much as possible.

A Richer Example

$$\begin{array}{l}\textbf{maximize } 4x_1 + 5x_2 + 4x_3 + 7x_4 + x_5 \\ \textbf{subject to} \\ w_1 = 1 - x_1 - x_3 - x_4 \\ w_2 = 1 - x_1 - x_2 - x_4 \\ w_3 = 1 - x_2 - x_3 \\ w_4 = 1 - x_4 - x_5 \\ w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0\end{array}$$

- So we choose x_4 as the ***entering*** variable.
  - Once we increase x_4 to 1, w_1,w_2, and w_4 all simultaneously become 0. That means that in this case, we can choose any one of them as the ***leaving*** variable.
  - Let us choose w_1 as the ***leaving*** variable. The resulting ***pivot*** produces the following new ***dictionary***:

$$\begin{array}{l}\textbf{maximize } 7 - 7w_1 - 3x_1 + 5x_2 - 3x_3 + x_5 \\ \textbf{subject to} \\ x_4 = 1 - w_1 - x_1 - x_3 \\ w_2 = w_1 - x_2 + x_3 \\ w_3 = 1 - x_2 - x_3 \\ w_4 = w_1 + x_1 + x_3 - x_5 \\ w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0\end{array}$$

- We now have a choice between x_2 and x_5 as the next ***entering*** variable;
- x_2 has a larger coefficient in the objective, let us choose x_2.

A Richer Example

$$\begin{aligned}
&\textbf{maximize } 7 - 7w_1 - 3x_1 + 5x_2 - 3x_3 + x_5\\
&\textbf{subject to}\\
&x_4 = 1 - w_1 - x_1 - x_3\\
&w_2 = w_1 - x_2 + x_3\\
&w_3 = 1 - x_2 - x_3\\
&w_4 = w_1 + x_1 + x_3 - x_5\\
&w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}$$

- Now, something strange happens:
  - we cannot increase x_2 at all without violating one of the constraints, because the current value of w_2 is already 0 and we have $-x_2$ on the right-hand side.
  - Still, in some sense, w_2 is the first basic variable that becomes 0, so we choose it as the ***leaving*** variable.
  - This results in the following ***dictionary***:

$$\begin{aligned}
&\textbf{maximize } 7 - 2w_1 - 5w_2 - 3x_1 + 2x_3 + x_5\\
&\textbf{subject to}\\
&x_4 = 1 - w_1 - x_1 - x_3\\
&x_2 = w_1 - w_2 + x_3\\
&w_3 = 1 - w_1 + w_2 - 2x_3\\
&w_4 = w_1 + x_1 + x_3 - x_5\\
&w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}$$

- We note that in this last ***pivot***, the objective remained at 7 (whereas “normally” the objective increases).
- Such a pivot is called a **degenerate pivot**.

A Richer Example

- Degenerate pivots can cause difficulties in general:
  - for example, it is possible that a sequence of ***degenerate pivots*** returns us to the same ***dictionary*** that we started with, resulting in an infinite loop. (This cannot happen with ***nondegenerate pivots***, because the strict increases in the objective make it impossible to return to the same dictionary.)
  - We will see how to deal with this in general later;
  - however, "usually" ***degenerate pivots*** do not cause any problems, and, as it turns out, it does not cause any trouble in our example:

$$\begin{array}{l}
\textbf{maximize } 7 - 2w_1 - 5w_2 - 3x_1 + 2x_3 + x_5 \\
\textbf{subject to} \\
x_4 = 1 - w_1 - x_1 - x_3 \\
x_2 = w_1 - w_2 + x_3 \\
w_3 = 1 - w_1 + w_2 - 2x_3 \\
w_4 = w_1 + x_1 + x_3 - x_5 \\
w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0
\end{array}$$

- Let x_3 be the next ***entering variable***, so that w_3 becomes the next ***leaving variable***.
- We obtain the following ***dictionary*** from this (***nondegenerate!***) pivot:

A Richer Example

$$\begin{array}{l}\textbf{maximize } 8 - 3w_1 - 4w_2 - w_3 - 3x_1 + x_5 \\ \textbf{subject to} \\ x_4 = 0.5 - 0.5w_1 - 0.5w_2 + 0.5w_3 - x_1 \\ x_2 = 0.5 + 0.5w_1 - 0.5w_2 - 0.5w_3 \\ x_3 = 0.5 - 0.5w_1 + 0.5w_2 - 0.5w_3 \\ w_4 = 0.5 + 0.5w_1 + 0.5w_2 - 0.5w_3 + x_1 - x_5 \\ w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0\end{array}$$

- Finally, we choose x_5, which is the only variable with a positive coefficient in the objective, as the ***entering variable***, so that w_4 becomes the ***leaving variable***.
- We end up with the following ***dictionary***:

$$\begin{array}{l}\textbf{maximize } 8.5 - 2.5w_1 - 3.5w_2 - 1.5w_3 - w_4 - 2x_1 \\ \textbf{subject to} \\ x_4 = 0.5 - 0.5w_1 - 0.5w_2 + 0.5w_3 - x_1 \\ x_2 = 0.5 + 0.5w_1 - 0.5w_2 - 0.5w_3 \\ x_3 = 0.5 - 0.5w_1 + 0.5w_2 - 0.5w_3 \\ x_5 = 0.5 + 0.5w_1 + 0.5w_2 - 0.5w_3 - w_4 + x_1 \\ w_1, w_2, w_3, w_4, x_1, x_2, x_3, x_4, x_5 \geq 0\end{array}$$

- Now, all of the coefficients in the objective are negative, so we know that it is impossible to obtain a better solution than 8.5, and we have arrived at the optimal solution which sets $x_2 = x_3 = x_4 = x_5 = 0.5$.

General Comments

- In general, we can choose any ***non-basic variable*** with positive coefficient in the objective as the ***entering variable***.
- As the ***leaving variable***, we must choose the ***basic variable*** that drops to zero first as the ***entering variable*** increases.
  - If there are multiple ***basic variables*** that drop to zero first, we can choose any one of them.
  - It may be the case that no ***basic variables*** will ever drop to zero; in this case, the linear program is ***unbounded***.
- Rules for making the above choices are called **pivoting rules**.

General Comments

- In the previous examples, we were lucky in the sense that whenever we changed a variable from ***non-basic*** to ***basic***, it never changed back; as a result, in some sense, we traversed the ***shortest possible path of pivots***.

- Unfortunately, in general, this is not the case.

  - In fact, for the common ***pivoting rules***, there are examples of linear programs where the simplex algorithm goes through a path of ***exponentially many dictionaries*** (Klee-Minty cubes are a common example of such linear programs).

  - It is not known if there is a pivoting rule that only requires ***poly-nomially many pivots*** on any example; in fact it is not known if there is always a path of only polynomially many pivots to the optimal solution.

  - In practice, however, the simplex algorithm tends to be extremely fast.

Avoiding Cycling

- Things may be even worse, though: as we mentioned above, if there are ***degenerate pivots***, then
  - There is the possibility that the simplex algorithm gets stuck in an infinite loop, that is, it **cycles**.
  - Cycling is extremely rare and hardly ever an issue even if we do not take precautions.
  - As it turns out, it is possible to avoid cycling in the simplex algorithm.
    - One way to avoid this is to use **Bland's rule** for pivoting.
      - When this rule has a choice among multiple variables as the entering (or leaving) variable, it always chooses the one with the lowest index.
    - Another way to avoid cycling is to use the **lexicographic method.**
      - The idea is to slightly perturb the constants so that degeneracy does not occur but the optimal solution is not really affected.

Unboundedness

- We have skipped over the issue of what happens if the linear program is unbounded.
- This is best illustrated with an example.
- Let us consider the following unbounded linear program:

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ w_1 = 1 - x_1 + x_2 \\ w_2 = 1 + x_1 - x_2 \\ w_1, w_2, x_1, x_2 \geq 0\end{array}}$$

- We first choose x_1 as the ***entering variable***, resulting in w_1 being the ***leaving variable***:

$$\boxed{\begin{array}{l}\textbf{maximize } 3 - 3w_1 + 5x_2 \\ \textbf{subject to} \\ x_1 = 1 - w_1 + x_2 \\ w_2 = 2 - w_1 \\ w_1, w_2, x_1, x_2 \geq 0\end{array}}$$

- Now we must choose x_2 as the ***entering variable***.
- However, as we increase x_2, none of the ***basic variables*** decrease. Hence, we can increase x_2 forever, indicating that the program is **unbounded**.

Finding a feasible dictionary

- So far, we have assumed that we have an ***initial feasible dictionary***.
  - While in many problems, setting all of the variables to zero corresponds to a feasible solution, this is certainly not always the case.
  - If we do not have an ***initial feasible dictionary***, we first need to find one, which can also be done using the simplex algorithm.
- Finding an ***initial feasible dictionary*** is generally referred to as "Phase 1."
- Going from there to the optimal solution (as we have done above) is called "Phase 2."
- Next, we discuss Phase 1.

checking feasibility via simplex

Consider the following new LP:

Maximize $-x_0$

Subject to:

$$
\begin{array}{ll}
a_{1,1}\,x_1 + a_{1,2}\,x_2 + \ldots + a_{1,n}\,x_n - x_0 & \le b_1 \\
a_{2,1}\,x_1 + a_{2,2}\,x_2 + \ldots + a_{2,n}\,x_n - x_0 & \le b_2 \\
\ldots & \ldots \\
a_{m,1}\,x_1 + a_{i,2}\,x_2 + \ldots + a_{m,n}\,x_n - x_0 & \le b_m
\end{array}
$$

$$x_0, x_1, \ldots, x_n \ge 0$$

- This LP is feasible: let $x_0 = -\min\{b_1, \ldots, b_m, 0\}$, $x_j = 0$, for $j = 1, \ldots, n$. We can also set up a feasible dictionary, and thus initial BFS, for it by introducing slack variables in an appropriate way.
- *Key point:* the original LP is feasible if and only if in an optimal solution to the new LP, $x_0^* = 0$.
- It also turns out, it is easy to derive a BFS for the original LP from an optimal BFS for this new LP.
- (In fact, finding an optimal solution given a feasible solution can also be reduced to checking whether a feasible solution exists.)

Example of Finding a feasible dictionary

- Consider the following linear program:

$$\boxed{\begin{array}{l}\textbf{maximize } 3x_1 + 2x_2 \\ \textbf{subject to} \\ 4x_1 + 2x_2 \leq 16 \\ x_1 + 2x_2 \leq 8 \\ x_1 + x_2 \leq 5 \\ -x_1 - x_2 \leq -2 \\ x_1 \geq 0; x_2 \geq 0\end{array}}$$

- Now, setting $x_1 = x_2 = 0$ is no longer a feasible solution.
  - ➢ To find a feasible solution, we temporarily forget about the objective.
  - ➢ Instead, we add an **auxiliary variable** x_0, and we require that each of the original constraints is violated by at most x_o.
  - ➢ This results in the following linear program:

$$\boxed{\begin{array}{l}\textbf{maximize } -x_0 \\ \textbf{subject to} \\ 4x_1 + 2x_2 - x_0 \leq 16 \\ x_1 + 2x_2 - x_0 \leq 8 \\ x_1 + x_2 - x_0 \leq 5 \\ -x_1 - x_2 - x_0 \leq -2 \\ x_0, x_1, x_2 \geq 0\end{array}}$$

- This LP has an optimal solution with objective value 0 if and only if there is a feasible solution to the original LP.

Example of Finding a feasible dictionary

$$\begin{aligned}
&\textbf{maximize } -x_0\\
&\textbf{subject to}\\
&4x_1 + 2x_2 - x_0 \le 16\\
&x_1 + 2x_2 - x_0 \le 8\\
&x_1 + x_2 - x_0 \le 5\\
&-x_1 - x_2 - x_0 \le -2\\
&x_0, x_1, x_2 \ge 0
\end{aligned}$$

- In equality form, we have:

$$\begin{aligned}
&\textbf{maximize } -x_0\\
&\textbf{subject to}\\
&w_1 = 16 - 4x_1 - 2x_2 + x_0\\
&w_2 = 8 - x_1 - 2x_2 + x_0\\
&w_3 = 5 - x_1 - x_2 + x_0\\
&w_4 = -2 + x_1 + x_2 + x_0\\
&w_1, w_2, w_3, w_4, x_0, x_1, x_2 \ge 0
\end{aligned}$$

- This is still not a ***feasible dictionary***, but we can transform it into a feasible dictionary by
  - ➢ choosing x_0 as the ***entering variable*** and the most negative basic variable, w_4, as the ***leaving variable***.
- This results in the following ***dictionary***:

Example of Finding a feasible dictionary

$$\begin{array}{l}
\textbf{maximize } -2 - w_4 + x_1 + x_2 \\
\textbf{subject to} \\
w_1 = 18 + w_4 - 5x_1 - 3x_2 \\
w_2 = 10 + w_4 - 2x_1 - 3x_2 \\
w_3 = 7 + w_4 - 2x_1 - 2x_2 \\
x_0 = 2 + w_4 - x_1 - x_2 \\
w_1, w_2, w_3, w_4, x_0, x_1, x_2 \geq 0
\end{array}$$

- Next, we choose (say) x_1 as the ***entering variable***, so that x_0 is the ***leaving variable***, resulting in:

$$\begin{array}{l}
\textbf{maximize } -x_0 \\
\textbf{subject to} \\
w_1 = 8 - 4w_4 + 2x_2 + 5x_0 \\
w_2 = 6 - w_4 - x_2 + 2x_0 \\
w_3 = 3 - w_4 + 2x_0 \\
x_1 = 2 + w_4 - x_2 - x_0 \\
w_1, w_2, w_3, w_4, x_0, x_1, x_2 \geq 0
\end{array}$$

- Now, all the coefficients in the objective are non-positive, so we have found the optimal solution to the auxiliary problem.

- We transform this into a feasible dictionary for the original problem by simply dropping x0 everywhere, and replacing the objective with the original one:

Example of Finding a feasible dictionary

maximize $3x_1 + 2x_2$
subject to

$$
\begin{aligned}
w_1 &= 8 - 4w_4 + 2x_2 \\
w_2 &= 6 - w_4 - x_2 \\
w_3 &= 3 - w_4 \\
w_4 &= x_1 + x_2 - 2 \\
& w_1, w_2, w_3, w_4, x_1, x_2 \geq 0
\end{aligned}
$$

- Because we now have a feasible dictionary (just set x_1=2,x_2=0), we can start Phase 2.