Chapter 21

Linear Programming II

By Sariel Har-Peled, December 7, 2009[1]

21.1 The Simplex Algorithm in Detail

The **Simplex** algorithm is presented on the right. We assume that we are given **SimplexInner**, a black box that solves a LP if the trivial solution of assigning zero to all the nonbasic variables is feasible. We remind the reader that $L' = \textbf{Feasible}(L)$ returns a new LP for which we have an easy feasible solution. This is done by introducing a new variable x_0 into the LP, where the original LP $\widehat{L}$ is feasible if and only if the new LP L has a feasible solution with $x_0 = 0$. As such, we set the target function in L to be minimizing x_0.

```
Simplex( L̂ a LP )
    Transform L̂ into slack form.
    Let L be the resulting slack form.
    Compute L' ← Feasible(L) (as described above)
    x ← LPStartSolution(L')
    x' ← SimplexInner(L', x)        (*)
    if objective function value of x' is > 0 then
        return "No solution"
    x'' ← SimplexInner(L, x')
    return x''
```

Figure 21.1: The **Simplex** algorithm.

We now apply **SimplexInner** to L' and the easy solution computed for L' by **LPStartSolution**(L'). If $x_0 > 0$ in the optimal solution for L' then there is no feasible solution for L, and we exit. Otherwise, we found a feasible solution to L, and we use it as the starting point for **SimplexInner** when it is applied to L.

Thus, in the following, we have to describe **SimplexInner** - a procedure to solve an LP in slack form, when we start from a feasible solution defined by the nonbasic variables assigned value zero.

One technicality that is ignored above, is that the starting solution we have for L', generated by **LPStartSolution**(L) is not legal as far as the slack form is concerned, because the non-basic variable x_0 is assigned a non-zero value. However, this can be easily resolve by immediately pivot on x_0 when we execute (*) in Figure 21.1. Namely, we first try to decrease x_0 as much as possible.

B - Set of indices of basic variables N - Set of indices of nonbasic variables $n = \lvert N \rvert$ - number of original variables b, c - two vectors of constants $m = \lvert B \rvert$ - number of basic variables (i.e., number of inequalities) $A = \left\{ a_{ij} \right\}$ - The matrix of coefficients $N \cup B = \{1, \ldots, n+m\}$ v - objective function constant.

(i)

$$\begin{array}{lll} \text{Max} & z = v + \sum_{j \in N} c_j x_j, & \\ \text{s.t.} & x_i = b_i - \sum_{j \in N} a_{ij} x_j & for \quad i \in B, \\ & x_i \geq 0, \quad \forall i = 1, \ldots, n+m. & \end{array}$$

(ii)

Figure 21.2: A linear program in slack form is specified by a tuple (N, B, A, b, c, v).

21.2 The SimplexInner Algorithm

We next describe the **SimplexInner** algorithm.

We remind the reader that the LP is given to us in slack form, see Figure 21.2. Furthermore, we assume that the trivial solution $x = \tau$, which is assigning all nonbasic variables zero, is feasible. In particualr, we immediately get the objective value for this solution from the notation which is v.

Assume, that we have a nonbasic variable x_e that appears in the objective function, and furthermore its coefficient c_e is positive in (the objective function), which is $z = v + \sum_{j \in N} c_j x_j$. Formally, we pick e to be one of the indices of

$$\left\{ j \;\middle|\; c_j > 0, \quad j \in N \right\}.$$

The variable x_e is the ***entering variable*** variable (since it is going to join the set of basic variables).

Clearly, if we increase the value of x_e (from the current value of 0 in τ) then one of the basic variables is going to vanish (i.e., become zero). Let x_l be this basic variable. We increase the value of x_e (the ***entering*** variable) till x_l (the ***leaving*** variable) becomes zero.

Setting all nonbasic variables to zero, and letting x_e grow, implies that $x_i = b_i - a_{ie} x_e$, for all $i \in B$.

All those variables must be non-negative, and thus we require that $\forall i \in B$ it holds $x_i = b_i - a_{ie} x_e \geq 0$. Namely, $x_e \leq (b_i / a_{ie})$ or alternatively, $\dfrac{1}{x_e} \geq \dfrac{a_{ie}}{b_i}$. Namely, $\dfrac{1}{x_e} \geq \max_{i \in B} \dfrac{a_{ie}}{b_i}$ and, the largest value of x_e which is still feasible is

$$U = \left(\max_{i \in B} \frac{a_{ie}}{b_i} \right)^{-1}.$$

We pick l (the index of the leaving variable) from the set all basic variables that vanish to zero when $x_e = U$. Namely, l is from the set

$$\left\{ j \;\middle|\; \frac{a_{je}}{b_j} = U \text{ where } j \in B \right\}.$$

Now, we know x_e and x_l. We rewrite the equation for x_l in the LP so that it has x_e on the left size. Formally, we do

$$x_l = b_l - \sum_{j \in N} a_{lj} x_j \qquad \Rightarrow \qquad x_e = \frac{b_l}{a_{le}} - \sum_{j \in N \cup \{l\}} \frac{a_{lj}}{a_{le}} x_j, \qquad \text{where } a_{ll} = 1.$$

We need to remove all the appearances on the right side of the LP of x_e. This can be done by substituting x_e into the other equalities, using the above equality. Alternatively, we do beforehand Gaussian elimination, to remove any appearance of x_e on the right side of the equalities in the LP (and also from the objective function) replaced by appearances of x_l on the left side, which we then transfer to the right side.

In the end of this process, we have a new *equivalent* LP where the basic variables are $B' = (B \setminus \{l\}) \cup \{e\}$ and the non-basic variables are $N' = (N \setminus \{e\}) \cup \{l\}$.

In end of this ***pivoting*** stage the LP objective function value had increased, and as such, we made progress. Note, that the linear system is completely defined by which variables are basic, and which are non-basic. Furthermore, pivoting never returns to a combination (of basic/non-basic variable) that was already visited. Indeed, we improve the value of the objective function in each pivoting stage. Thus, we can do at most

$$\binom{n+m}{n} \leq \left(\frac{n+m}{n} \cdot e\right)^n$$

pivoting steps. And this is close to tight in the worst case (there are examples where 2^n pivoting steps are needed.

Each pivoting step takes polynomial time in n and m. Thus, the overall running time of **Simplex** is exponential in the worst case. However, in practice, **Simplex** is extremely fast.

21.2.1 Degeneracies

If you inspect carefully the **Simplex** algorithm, you would notice that it might get stuck if one of the b_is is zero. This corresponds to a case where $> m$ hyperplanes passes through the same point. This might cause the effect that you might not be able to make any progress at all in pivoting.

There are several solutions, the simplest one is to add tiny random noise to each coefficient. You can even do this symbolically. Intuitively, the degeneracy, being a local phenomena on the polytope disappears with high probability.

The larger danger, is that you would get into cycling; namely, a sequence of pivoting operations that do not improve the objective function, and the bases you get are cyclic (i.e., infinite loop).

There is a simple scheme based on using the symbolic perturbation, that avoids cycling, by carefully choosing what is the leaving variable. We omit all further details here.

There is an alternative approach, called ***Bland's rule***, which always choose the lowest index variable for entering and leaving out of the possible candidates. We will not prove the correctness of this approach here.

21.2.2 Correctness of linear programming

Theorem 21.2.1 (Fundamental theorem of Linear Programming.) *For an arbitrary linear program, the following statements are true:*

1. *If there is no optimal solution, the problem is either infeasible or unbounded.*
2. *If a feasible solution exists, then a basic feasible solution exists.*
3. *If an optimal solution exists, then a basic optimal solution exists.*

Proof: Proof is constructive by running the simplex algorithm. ■

21.2.3 On the ellipsoid method and interior point methods

The **Simplex** algorithm has exponential running time in the worst case.

The ellipsoid method is *weakly* polynomial (namely, it is polynomial in the number of bits of the input). Khachian in 1979 came up with it. It turned out to be completely useless in practice.

In 1984, Karmakar came up with a different method, called the *interior-point method* which is also weakly polynomial. However, it turned out to be quite useful in practice, resulting in an arm race between the interior-point method and the simplex method.

The question of whether there is a *strongly* polynomial time algorithm for linear programming, is one of the major open questions in computer science.

21.3 Duality and Linear Programming

Every linear program L has a ***dual linear program*** L'. Solving the dual problem is essentially equivalent to solving the ***primal linear program*** (i.e., the original) LP.

21.3.1 Duality by Example

Consider the linear program L depicted on the right (Figure 21.3). Note, that any feasible solution, gives us a lower bound on the maximal value of the target function, denoted by η. In particular, the solution $x_1 = 1, x_2 = x_3 = 0$ is feasible, and implies $z = 4$ and thus $\eta \geq 4$.

Similarly, $x_1 = x_2 = 0$, $x_3 = 3$ is feasible and implies that $\eta \geq z = 9$.

$$
\begin{array}{ll}
\max & z = 4x_1 + x_2 + 3x_3 \\
\text{s.t.} & x_1 + 4x_2 \leq 1 \\
& 3x_1 - x_2 + x_3 \leq 3 \\
& x_1, x_2, x_3 \geq 0
\end{array}
$$

Figure 21.3: The linear program L.

We might be wondering how close is this solution to the optimal solution? In particular, if this solution is very close to the optimal solution, we might be willing to stop and be satisfied with it.

Let us add the first inequality (multiplied by 2) to the second inequality (multiplied by 3). Namely, we add the inequality $2(x_1 + 4x_2) \leq 2(1)$ to the inequality $+3(3x_1 - x_2 + x_3) \leq 3(3)$. The resulting inequality is

$$11x_1 + 5x_2 + 3x_3 \leq 11. \tag{21.1}$$

Note, that this inequality must hold for any feasible solution of L. Now, the objective function is $z = 4x_1 + x_2 + 3x_3$ and x_1, x_2 and x_3 are all non-negative, and the inequality of Eq. (21.1) has larger coefficients that all the coefficients of the target function, for the corresponding variables. It thus follows, that for any feasible solution, we have $z \leq 11x_1 + 5x_2 + 3x_3 \leq 11$.

As such, the optimal value of the LP L is somewhere between 9 and 11.

We can extend this argument. Let us multiply the first inequality by y_1 and second inequality by y_2 and add them up. We get:

$$\begin{array}{ll} \max & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & \qquad \text{for } i = 1, \ldots, m, \\ & x_j \geq 0, \\ & \qquad \text{for } j = 1, \ldots, n. \end{array}$$

(a) ***primal program***

$$\begin{array}{ll} \min & \sum_{i=1}^{m} b_i y_i \\ \text{s.t.} & \sum_{i=1}^{m} a_{ij} y_i \geq c_j, \\ & \qquad \text{for } j = 1, \ldots, n, \\ & y_i \geq 0, \\ & \qquad \text{for } i = 1, \ldots, m. \end{array}$$

(b) ***dual program***

$$\begin{array}{ll} \max & \sum_{i=1}^{m} (-b_i) y_i \\ \text{s.t.} & \sum_{i=1}^{m} (-a_{ij}) y_i \leq -c_j, \\ & \qquad \text{for } j = 1, \ldots, n, \\ & y_i \geq 0, \\ & \qquad \text{for } i = 1, \ldots, m. \end{array}$$

(c) dual program in standard form

Figure 21.5: Dual linear programs.

$$\begin{array}{ccccccccc} y_1(x_1 & + & 4x_2 & & &) \leq & y_1(1) \\ + y_2(3x_1 & - & x_2 & + & x_3 &) \leq & y_2(3) \\ \hline (y_1 + 3y_2)x_1 & + & (4y_1 - y_2)x_2 & + & y_2 x_3 & \leq & y_1 + 3y_2. \end{array} \tag{21.2}$$

Compare this to the target function $z = 4x_1 + x_2 + 3x_3$. If this expression is bigger than the target function in each variable, namely

$$\begin{array}{lcl} 4 & \leq & y_1 + 3y_2 \\ 1 & \leq & 4y_1 - y_2 \\ 3 & \leq & y_2, \end{array}$$

then, $z = 4x_1 + x_2 + 3x_3 \leq (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2 x_3 \leq y_1 + 3y_2$, the last step follows by Eq. (21.2).

Thus, if we want the best upper bound on η (the maximal value of z) then we want to solve the LP $\widehat{L}$ depicted in Figure 21.4. This is the dual program to L and its optimal solution is an upper bound to the optimal solution for L.

$$\begin{array}{ll} \min & y_1 + 3y_2 \\ \text{s.t.} & y_1 + 3y_2 \geq 4 \\ & 4y_1 - y_2 \geq 1 \\ & y_2 \geq 3 \\ & y_1, y_2 \geq 0. \end{array}$$

Figure 21.4: The dual LP $\widehat{L}$. The primal LP is depicted in Figure 21.3.

21.3.2 The Dual Problem

Given a linear programming problem (i.e., ***primal problem***, seen in Figure 21.5 (a), its associated ***dual linear program***s in Figure 21.5 (b). The standard form of the dual LP is depicted in Figure 21.5 (c). Interestingly, you can just compute the dual LP to the given dual LP. What you get back is the original LP. This is demonstrated in Figure 21.6.

We just proved the following result.

Lemma 21.3.1 *Let L be an LP, and let L' be its dual. Let L'' be the dual to L'. Then L and L'' are the same LP.*

$$
\begin{aligned}
\max\ & \sum_{i=1}^{m} (-b_i) y_i \\
\text{s.t.}\ & \sum_{i=1}^{m} (-a_{ij}) y_i \leq -c_j, \\
& \qquad \text{for } j = 1, \ldots, n, \\
& y_i \geq 0, \\
& \qquad \text{for } i = 1, \ldots, m.
\end{aligned}
$$

(a) dual program

$$
\begin{aligned}
\min\ & \sum_{j=1}^{n} -c_j x_j \\
\text{s.t.}\ & \sum_{j=1}^{n} (-a_{ij}) x_j \geq -b_i, \\
& \qquad \text{for } i = 1, \ldots, m, \\
& x_j \geq 0, \\
& \qquad \text{for } j = 1, \ldots, n.
\end{aligned}
$$

(b) the dual program to the dual program

$$
\begin{aligned}
\max\ & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.}\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\
& \qquad \text{for } i = 1, \ldots, m, \\
& x_j \geq 0, \\
& \qquad \text{for } j = 1, \ldots, n.
\end{aligned}
$$

(c) ... which is the original LP.

Figure 21.6: The dual to the dual linear program. Computing the dual of (a) can be done mechanically by following Figure 21.5 (a) and (b). Note, that (c) is just a rewriting of (b).

21.3.3 The Weak Duality Theorem

Theorem 21.3.2 *If $(x_1, x_2, \ldots, x_n)$ is feasible for the primal LP and $(y_1, y_2, \ldots, y_m)$ is feasible for the dual LP, then*

$$\sum_j c_j x_j \leq \sum_i b_i y_i .$$

Namely, all the feasible solutions of the dual bound all the feasible solutions of the primal.

Proof: By substitution from the dual form, and since the two solutions are feasible, we know that

$$\sum_j c_j x_j \leq \sum_j \left(\sum_{i=1}^{m} y_i a_{ij} \right) x_j \leq \sum_i \left(\sum_j a_{ij} x_j \right) y_i \leq \sum_i b_i y_i \ .$$

■

Interestingly, if we apply the weak duality theorem on the dual program (namely, Figure 21.6 (a) and (b)), we get the inequality $\sum_{i=1}^{m} (-b_i) y_i \leq \sum_{j=1}^{n} -c_j x_j$, which is the original inequality in the weak duality theorem. Thus, the weak duality theorem does not imply the strong duality theorem which will be discussed next.