Chapter 21

Linear Programming II

By Sariel Har-Peled, December 10, 2013[①] Version: 1.1

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($\widehat{L}$ a LP)
  Transform $\widehat{L}$ into slack form.
  Let L be the resulting slack form.
  $L' \leftarrow$ **Feasible**(L)
  $x \leftarrow$ **LPStartSolution**(L')
  $x' \leftarrow$ **SimplexInner**(L', x) (*)
  $z \leftarrow$ objective function value of x'
  if $z > 0$ **then**
    return "No solution"
  $x'' \leftarrow$ **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
$n = |N|$ - number of original variables
b, c - two vectors of constants
$m = |B|$ - number of basic variables (i.e., number of inequalities)
$A = \{a_{ij}\}$ - The matrix of coefficients
$N \cup B = \{1, \ldots, n+m\}$
v - objective function constant.

(i)

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

(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 particular, 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

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

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, $\dfrac{1}{x_e} \geq \max\limits_{i \in B} \dfrac{a_{ie}}{b_i}$ 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

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

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. This is described in detail in Section 21.6.

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

Definition 21.2.1. A solution to an LP is a ***basic solution*** if it the result of setting all the nonbasic variables to zero.

Note that the **Simplex** algorithm deals only with basic solutions. In particular we get the following.

Theorem 21.2.2 (Fundamental theorem of Linear Programming.). *For an arbitrary linear program, the following statements are true:*

(A) If there is no optimal solution, the problem is either infeasible or unbounded.

(B) If a feasible solution exists, then a basic feasible solution exists.

(C) 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 $\eta \geq 4$.

max	$z = 4x_1 + x_2 + 3x_3$
s.t.	$x_1 + 4x_2 \leq 1$
	$3x_1 - x_2 + x_3 \leq 3$
	$x_1, x_2, x_3 \geq 0$

Figure 21.3: The linear program L.

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

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 two inequalities:

$$\begin{aligned} 2(\ x_1 + 4x_2 \qquad) &\leq 2(1) \\ +3(3x_1 - x_2 + x_3) &\leq 3(3). \end{aligned}$$

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

$$z = 4x_1 + x_2 + 3x_3 \leq 11x_1 + 5x_2 + 3x_3 \leq 11,$$

since all the variables are non-negative. 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:

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

(a) ***primal program***

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

(b) ***dual program***

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

(c) dual program in standard form

Figure 21.5: Dual linear programs.

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

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

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

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.

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

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*** is 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.*

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

(a) dual program

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

(b) the dual program to the dual program

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

(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.

21.4 The strong duality theorem

The ***strong duality theorem*** states the following.

Theorem 21.4.1. *If the primal LP problem has an optimal solution $x^* = (x_1^*, \ldots, x_n^*)$ then the dual also has an optimal solution, $y^* = (y_1^*, \ldots, y_m^*)$, such that*

$$\sum_j c_j x_j^* = \sum_i b_i y_i^*.$$

Its proof is somewhat tedious and not very insightful, the basic idea to prove this theorem is to run the simplex algorithm simultaneously on both the primal and the dual LP making steps in sync. When the two stop, they must be equal of they are feasible. We omit the tedious proof.

21.5 Some duality examples

21.5.1 Shortest path

You are given a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, with source s and target t. We have weights $\omega(u, v)$ on each edge $(u \to v) \in \mathsf{E}$, and we are interested in the shortest path in this graph from s to t. To simplify the exposition assume that there are no incoming edges in s and no edges leave t. To this end, let d_x be a variable that is the distance between s and x, for any $x \in \mathsf{V}$. Clearly, we must have for any edge $(u \to v) \in \mathsf{E}$, that $d_u + \omega(u, v) \geq d_v$. We also know that $d_{\mathsf{s}} = 0$. Clearly, a trivial solution to this constraints is to set all the variables to zero. So, we are trying to find the assignment that maximizes d_{t}, such that all the constraints are filled. As such, the LP for computing the shortest path from s to t is the following LP.

$$\begin{aligned}
\max \quad & d_{\mathsf{t}} && \\
\text{s.t.} \quad & d_{\mathsf{s}} \leq 0 && \\
& d_u + \omega(u, v) \geq d_v && \forall (u \to v) \in \mathsf{E}, \\
& d_x \geq 0 && \forall x \in \mathsf{V}.
\end{aligned}$$

Equivalently, we get

$$\begin{aligned}
\max \quad & d_{\mathsf{t}} && \\
\text{s.t.} \quad & d_{\mathsf{s}} \leq 0 && \\
& d_v - d_u \leq \omega(u, v) && \forall (u \to v) \in \mathsf{E}, \\
& d_x \geq 0 && \forall x \in \mathsf{V}.
\end{aligned}$$

Let use compute the dual. To this end, let y_{uv} be the dual variable for the edge $(u \to v)$, and let y_{s} be the dual variable for the $d_{\mathsf{s}} \leq 0$ inequality. We get the following dual LP.

$$\begin{aligned}
\min \quad & \sum_{(u \to v) \in \mathsf{E}} y_{uv} \omega(u, v) && && \\
\text{s.t.} \quad & y_{\mathsf{s}} - \sum_{(\mathsf{s} \to u) \in \mathsf{E}} y_{\mathsf{s}u} \geq 0 && && (*) \\
& \sum_{(u \to x) \in \mathsf{E}} y_{ux} - \sum_{(x \to v) \in \mathsf{E}} y_{xv} \geq 0 && \forall x \in \mathsf{V} \setminus \{\mathsf{s}, \mathsf{t}\} && (**) \\
& \sum_{(u \to \mathsf{t}) \in \mathsf{E}} y_{u\mathsf{t}} \geq 1 && && (***) \\
& y_{uv} \geq 0 && \forall (u \to v) \in \mathsf{E}, && \\
& y_{\mathsf{s}} \geq 0. && &&
\end{aligned}$$

Look carefully at this LP. The trick is to think about the y_{uv} as a flow on the edge y_{uv}. (Also, we assume here that the weights are positive.) Then, this LP is the min cost flow of sending one unit of flow from the source s to t. Indeed, if the weights are positive, then (**) can be assumed to be hold with equality in the optimal solution, and this is conservation of flow. Equation (***) implies that one unit of flow arrives to the sink t. Finally, (*) implies that at least y_{s} units of flow leaves the source. The remaining of the LP implies that $y_{\mathsf{s}} \geq 1$. Of course, this min-cost flow version, is without capacities on the edges.

21.5.2 Set Cover and Packing

Consider an instance of Set Cover with $(\mathsf{S}, \mathcal{F})$, where $\mathsf{S} = \{u_1, \ldots, u_n\}$ and $\mathcal{F} = \{F_1, \ldots, F_m\}$, where $F_i \subseteq \mathsf{S}$. The natural LP to solve this problem is

$$\begin{aligned}
\min \quad & \sum_{F_j \in \mathcal{F}} x_j & \\
\text{s.t.} \quad & \sum_{\substack{F_j \in \mathcal{F}, \\ u_i \in F_j}} x_j \geq 1 & \forall u_i \in \mathsf{S}, \\
& x_j \geq 0 & \forall F_j \in \mathcal{F}.
\end{aligned}$$

The dual LP is

$$\begin{aligned}
\max \quad & \sum_{u_i \in \mathsf{S}} y_i & \\
\text{s.t.} \quad & \sum_{u_i \in F_j} y_i \leq 1 & \forall F_j \in \mathcal{F}, \\
& y_i \geq 0 & \forall u_i \in \mathsf{S}.
\end{aligned}$$

This is a ***packing*** LP. We are trying to pick as many vertices as possible, such that no set has more than one vertex we pick. If the sets in $\mathcal{F}$ are pairs (i.e., the set system is a graph), then the problem is known as ***edge cover***, and the dual problem is the familiar ***independent set*** problem. Of course, these are all the fractional versions – getting an integral solution for these problems is completely non-trivial, and in all these cases is impossible in polynomial time since the problems are NP-Complete.

As an exercise, write the LP for Set Cover for the case where every set has a price associated with it, and you are trying to minimize the total cost of the cover.

21.5.3 Network flow

(We do the following in excruciating details – hopefully its make the presentation clearer.)

Let assume we are given an instance of network flow G, with source s, and sink t. As usual, let us assume there are no incoming edges into the source, no outgoing edges from the sink, and the two are not connected by an edge. The LP for this network flow is the following.

$$\begin{aligned}
\max \quad & \sum_{(\mathsf{s} \to v) \in \mathsf{E}} x_{\mathsf{s} \to v} & \\
& x_{u \to v} \leq \mathsf{c}(u \to v) & \forall\, (u \to v) \in \mathsf{E} \\
& \sum_{(u \to v) \in \mathsf{E}} x_{u \to v} - \sum_{(v \to w) \in \mathsf{E}} x_{v \to w} \leq 0 & \forall v \in \mathsf{V} \setminus \{\mathsf{s}, \mathsf{t}\} \\
& - \sum_{(u \to v) \in \mathsf{E}} x_{u \to v} + \sum_{(v \to w) \in \mathsf{E}} x_{v \to w} \leq 0 & \forall v \in \mathsf{V} \setminus \{\mathsf{s}, \mathsf{t}\} \\
& 0 \leq x_{u \to v} & \forall\, (u \to v) \in \mathsf{E}.
\end{aligned}$$

To perform the duality transform, we define a dual variable for each inequality. We get the following

dual LP:

$$\begin{aligned}
\max \quad & \sum_{(\mathsf{s}\to v)\in \mathsf{E}} x_{\mathsf{s}\to v} & & & \\
& x_{u\to v} \le \mathsf{c}(u\to v) & * \quad & y_{u\to v} & \forall\,(u\to v)\in \mathsf{E} \\
& \sum_{(u\to v)\in \mathsf{E}} x_{u\to v} - \sum_{(v\to w)\in \mathsf{E}} x_{v\to w} \le 0 & * \quad & y_v & \forall v \in \mathsf{V}\setminus\{\mathsf{s},\mathsf{t}\} \\
& -\sum_{(u\to v)\in \mathsf{E}} x_{u\to v} + \sum_{(v\to w)\in \mathsf{E}} x_{v\to w} \le 0 & * \quad & y'_v & \forall v \in \mathsf{V}\setminus\{\mathsf{s},\mathsf{t}\} \\
& 0 \le x_{u\to v} & & & \forall\,(u\to v)\in \mathsf{E}.
\end{aligned}$$

Now, we generate the inequalities on the coefficients of the variables of the target functions. We need to carefully account for the edges, and we observe that there are three kinds of edges: source edges, regular edges, and sink edges. Doing the duality transformation carefully, we get the following:

$$\begin{aligned}
\min \quad & \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v)\, y_{u\to v} & & \\
& 1 \le y_{\mathsf{s}\to v} + y_v - y'_v & & \forall\,(\mathsf{s}\to v)\in \mathsf{E} \\
& 0 \le y_{u\to v} + y_v - y'_v - y_u + y'_u & & \forall\,(u\to v)\in \mathsf{E}(\mathsf{G}\setminus\{\mathsf{s},\mathsf{t}\}) \\
& 0 \le y_{v\to \mathsf{t}} - y_v + y'_v & & \forall\,(v\to \mathsf{t})\in \mathsf{E} \\
& y_{u\to v} \ge 0 & & \forall\,(u\to v)\in \mathsf{E} \\
& y_v \ge 0 & & \forall v \in \mathsf{V} \\
& y'_v \ge 0 & & \forall v \in \mathsf{V}
\end{aligned}$$

To understand what is going on, let us rewrite the LP, introducing the variable $d_v = y_v - y'_v$, for each $v \in V$[②]. We get the following modified LP:

$$\begin{aligned}
\min \quad & \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v)\, y_{u\to v} & & \\
& 1 \le y_{\mathsf{s}\to v} + d_v & & \forall\,(\mathsf{s}\to v)\in \mathsf{E} \\
& 0 \le y_{u\to v} + d_v - d_u & & \forall\,(u\to v)\in \mathsf{E}(\mathsf{G}\setminus\{\mathsf{s},\mathsf{t}\}) \\
& 0 \le y_{v\to \mathsf{t}} - d_v & & \forall\,(v\to \mathsf{t})\in \mathsf{E} \\
& y_{u\to v} \ge 0 & & \forall\,(u\to v)\in \mathsf{E}
\end{aligned}$$

Adding the two variables for t and s, and setting their values as follows $d_{\mathsf{t}} = 0$ and $d_{\mathsf{s}} = 1$, we get the following LP:

$$\begin{aligned}
\min \quad & \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v)\, y_{u\to v} & & \\
& 0 \le y_{\mathsf{s}\to v} + d_v - d_{\mathsf{s}} & & \forall\,(\mathsf{s}\to v)\in \mathsf{E} \\
& 0 \le y_{u\to v} + d_v - d_u & & \forall\,(u\to v)\in \mathsf{E}(\mathsf{G}\setminus\{\mathsf{s},\mathsf{t}\}) \\
& 0 \le y_{v\to \mathsf{t}} + d_{\mathsf{t}} - d_v & & \forall\,(v\to \mathsf{t})\in \mathsf{E} \\
& y_{u\to v} \ge 0 & & \forall\,(u\to v)\in \mathsf{E} \\
& d_{\mathsf{s}} = 1, \qquad d_{\mathsf{t}} = 0 & &
\end{aligned}$$

[②]We could have done this directly, treating the two inequalities as equality, and multiplying it by a single variable that can be both positive and negative – however, it is useful to see why this is correct at least once.

Which simplifies to the following LP:

$$\begin{aligned}
\min \sum_{(u\to v)\in \mathsf{E}} & \mathsf{c}(u\to v)\, y_{u\to v} \\
& d_u - d_v \le y_{u\to v} && \forall\, (u\to v)\in \mathsf{E} \\
& y_{u\to v} \ge 0 && \forall\, (u\to v)\in \mathsf{E} \\
& d_{\mathsf{s}} = 1, \qquad d_{\mathsf{t}} = 0.
\end{aligned}$$

The above LP can be interpreted as follows: We are assigning weights to the edges (i.e., $y_{(u\to v)}$). Given such an assignment, it is easy to verify that setting d_u (for all u) to be the shortest path distance under this weighting to the sink t, complies with all inequalities, the assignment $d_{\mathsf{s}} = 1$ implies that we require that the shortest path distance from the source to the sink has length exactly one.

We are next going to argue that the optimal solution to this LP is a min-cut. Lets us first start with the other direction, given a cut (S, T) with $\mathsf{s} \in S$ and $\mathsf{t} \in T$, observe that setting

$$\begin{aligned}
d_u &= 1 && \forall u \in S \\
d_u &= 0 && \forall u \in T \\
y_{u\to v} &= 1 && \forall\, (u\to v) \in (S, T) \\
y_{u\to v} &= 0 && \forall\, (u\to v) \in \mathsf{E} \setminus (S, T)
\end{aligned}$$

is a valid solution for the LP.

As for the other direction, consider the optimal solution for the LP, and let its target function value be

$$\alpha^* = \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v)\, y^*_{u\to v}$$

(we use (*) notation to the denote the values of the variables in the optimal LP solution). Consider generating a cut as follows, we pick a random value uniformly in $z \in [0, 1]$, and we set $S = \left\{ u \;\middle|\; d^*_u \ge z \right\}$ and $T = \left\{ u \;\middle|\; d^*_u < z \right\}$. This is a valid cut, as $\mathsf{s} \in S$ (as $d^*_{\mathsf{s}} = 1$) and $\mathsf{t} \in T$ (as $d^*_{\mathsf{t}} = 0$). Furthermore, an edge $(u \to v)$ is in the cut, only if $d^*_u > d^*_v$ (otherwise, it is not possible to cut this edge using this approach).

In particular, the probability of $u \in S$ and $v \in T$, is exactly $d^*_u - d^*_v$! Indeed, it is the probability that z falls inside the interval $[d^*_v, d^*_u]$. As such, $(u \to v)$ is in the cut with probability $d^*_u - d^*_v$ (again, only if $d^*_u > d^*_v$), which is bounded by $y^*_{(u\to v)}$ (by the inequality $d_u - d_v \le y_{u\to v}$ in the LP).

So, let $X_{u\to v}$ be an indicator variable which is one if the edge is in the generated cu. We just argued that $\mathbf{E}[X_{u\to v}] = \mathbf{Pr}[X_{u\to v} = 1] \le y^*_{(u\to v)}$. We thus have that the expected cost of this random cut is

$$\mathbf{E}\left[\sum_{(u\to v)\in \mathsf{E}} X_{u\to v}\, \mathsf{c}(u\to v) \right] = \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v)\, \mathbf{E}[X_{u\to v}\,] \le \sum_{(u\to v)\in \mathsf{E}} \mathsf{c}(u\to v) y^*_{u\to v} = \alpha^*.$$

That is, the expected cost of a random cut here is at most the value of the LP optimal solution. In particular, there must be a cut that has cost at most α^*, see Remark 21.5.2 below. However, we argued that α^* is no larger than the cost of any cut. We conclude that α^* is the cost of the min cut.

We are now ready for the kill, the optimal value of the original max-flow LP; that is, the max-flow (which is a finite number because all the capacities are bounded numbers), is equal by the strong duality theorem, to the optimal value of the dual LP (i.e., α^*). We just argued that α^* is the cost of the min cut in the given network. As such, we proved the following.

Lemma 21.5.1. *The Min-Cut Max-Flow Theorem follows from the strong duality Theorem for Linear Programming.*

Remark 21.5.2. In the above, we used the following "trivial" but powerful argument. Assume you have a random variable Z, and consider its expectation $\mu = \mathbf{E}[Z]$. The expectation μ is the weighted average value of the values the random variable Z might have, and in particular, there must be a value z that might be assigned to Z (with non-zero probability), such that $z \leq \mu$. Putting it differently, the weighted average of a set of numbers is bigger (formally, no smaller) than some number in this set.

This argument is one of the standard tools in the ***probabilistic method*** – a technique to prove the existence of entities by considering expectations and probabilities.

21.6 Solving LPs without ever getting into a loop - symbolic perturbations

21.6.1 The problem and the basic idea

Consider the following LP:

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

(Here $B = \{1, \ldots, n\}$ and $N = \{n+1, \ldots, n+m\}$.) The **Simplex** algorithm might get stuck in a loop of pivoting steps, if one of the constants b_i becomes zero during the algorithm execution. To avoid this, we are going to add tiny infinitesimals to all the equations. Specifically, let $\varepsilon > 0$ be an arbitrarily small constant, and let $\varepsilon_i = \varepsilon^i$. The quantities $\varepsilon_1, \ldots, \varepsilon_n$ are infinitesimals of different scales. We slightly perturb the above LP by adding them to each equation. We get the following modified LP:

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

Importantly, any feasible solution to the original LP translates into a valid solution of this LP (we made things better by adding these symbolic constants).

The rule of the game is now that we treat $\varepsilon_1, \ldots, \varepsilon_n$ as symbolic constants. Of course, when we do pivoting, we need to be able to compare two numbers and decide which one is bigger. Formally, given two numbers

$$\alpha = \alpha_0 + \alpha_1 \varepsilon_1 + \cdots + \alpha_n \varepsilon_n \qquad \text{and} \qquad \beta = \beta_0 + \beta_1 \varepsilon_1 + \cdots + \beta_n \varepsilon_n, \tag{21.3}$$

then $\alpha > \beta$ if and only if there is an index i such that $\alpha_0 = \beta_0, \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}$ and $\alpha_i > \beta_i$. That is, $\alpha > \beta$ if the vector $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ is ***lexicographically*** larger than $(\beta_0, \beta_1, \ldots, \beta_n)$.

Significantly, but not obviously at this stage, the simplex algorithm would never divide an ε_i by an ε_j, so we are good to go – we can perform all the needed arithmetic operations of the **Simplex** using

these symbolic constants, and we claim that now the constant term (which is a number of the form of Eq. (21.3)) is now never zero. This implies immediately that the **Simplex** algorithm always makes progress, and it does terminates. We still need to address the two issues:

(A) How are the symbolic perturbations are updated at each iteration?

(B) Why the constants can never be zero?

21.6.2 Pivoting as a Gauss elimination step

Consider the LP equations

$$x_i + \sum_{j \in N} a_{ij} x_j = b_i, \qquad \text{for } i \in B,$$

where $B = \{1, \ldots, n\}$ and $N = \{n+1, \ldots, n+m\}$. We can write these equations down in matrix form

x_1	x_2	$\ldots$	x_n	x_{n+1}	x_{n+2}	$\ldots$	x_j	$\ldots$	x_{n+m}	const
1	0	$\ldots$	0	$a_{1,n+1}$	$a_{1,n+2}$	$\ldots$	$a_{1,j}$	$\ldots$	$a_{1,n+m}$	b_1
0	1	$\ldots$	0	$a_{2,n+1}$	$a_{2,n+2}$	$\ldots$	$a_{2,j}$	$\ldots$	$a_{2,n+m}$	b_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$0 \ldots 0$	1	$0 \ldots 0$	$a_{k,n+1}$	$a_{k,n+2}$	$\ldots$	$a_{k,j}$	$\vdots$	$a_{k,n+m}$	b_k
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\ldots$	0	1	$a_{n,n+1}$	$a_{n,n+2}$	$\ldots$	$a_{n,j}$	$\ldots$	$a_{n,n+m}$	b_n

Assume that now we do a pivoting step with x_j entering the basic variables, and x_k leaving. To this end, let us multiply the kth row (i.e., the kth equation) by $1/a_{k,j}$, this result in the kth row having 1 instead of $a_{k,j}$. Let this resulting row be denoted by $\mathbf{r}$. Now, add $a_{i,j}\mathbf{r}$ to the ith row of the matrix, for all i. Clearly, in the resulting row/equation, the coefficient of x_j is going to be zero, in all rows except the kth one, where it is 1. Note, that on the matrix on the left side, all the columns are the same, except for the kth column, which might now have various numbers in this column. The final step is to exchange the kth column on the left, with the jth column on the right. And that is one pivoting step, when working on the LP using a matrix. It is very similar to one step of the Gauss elimination in matrices, if you are familiar with that.

21.6.2.1 Back to the perturbation scheme

We now add a new matrix to the above representations on the right side, that keeps track of the εs. This looks initially as follows.

x_1	x_2	$\ldots$	x_n	x_{n+1}	$\ldots$	x_j	$\ldots$	x_{n+m}	const	ε_1	ε_2	$\ldots$	ε_n
1	0	$\ldots$	0	$a_{1,n+1}$	$\ldots$	$a_{1,j}$	$\ldots$	$a_{1,n+m}$	b_1	1	0	$\ldots$	0
0	1	$\ldots$	0	$a_{2,n+1}$	$\ldots$	$a_{2,j}$	$\ldots$	$a_{2,n+m}$	b_2	0	1	$\ldots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$0 \ldots 0$	1	$0 \ldots 0$	$a_{k,n+1}$	$\ldots$	$a_{k,j}$	$\vdots$	$a_{k,n+m}$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$\ldots$	0	1	$a_{n,n+1}$	$\ldots$	$a_{n,j}$	$\ldots$	$a_{n,n+m}$	b_n	0	0	$\ldots$	1

Now, we run the algorithm as described above, using the εs to resolve which variables are entering and leaving. The critical observation is that throughout the algorithm execution we are adding rows, and multiplying them by non-zero constants. The matrix on the right has initially full rank, and throughout the execution of the algorithm its rank remains the same (because the linear operation we do on the rows can not change the rank of the matrix). In particular, it is impossible that a row on the right side of the matrix is all zero, or equal to another row, or equal to another row if multiplied by a constant. Namely, the symbolic constant encoded by the εs as we run the **Simplex** algorithm can never be zero. And furthermore, these constants are never equal for two different equations. We conclude that the **Simplex** algorithm now always make progress in each pivoting step.

21.6.2.2 The overall algorithm

We run the **Simplex** algorithm with the above described symbolic perturbation. The final stroke is that each basic variable x_i in the computed solution now equal to a number of the form $x_i = \alpha_0 + \sum_i \alpha_i \varepsilon_i$. We interpret this as $x_i = \alpha_0$, by setting all the εs to be zero.