Topics in Theoretical Computer Science May 4, 2015

Lecture 11

Lecturer: Ola Svensson *Scribes: Vincent Eggerling, Simon Rodriguez*

1 Introduction

In the last lecture we covered the *ellipsoid method* and its application for solving linear problems. We saw that to apply this method we mainly needed an efficient separation oracle ; in fact, we do not even have to be able to write the linear problem : it is possible to have an exponential number of constraints and still solve the problem. Moreover the ellipsoid method runs theoretically in polynomial time. But we also saw that the method is slow in practice and that some technicalities at initialization (choosing the first ellipsoid) and termination limit the use of it.
In this lecture, we are going to cover two other LP resolution methods : the *multiplicative weight update method* and the *simplex method*.

2 Multiplicative weight update method

In this section we present the general idea of the multiplicative weight update method and propose a corresponding algorithm and its analysis. We will have to restrict ourselves to a certain type of LPs, the *covering LPs*, defined as follows.

2.1 Covering LPs

Definition 1 *A linear program of the form :*

$$\begin{aligned} \textbf{\textit{minimize}} \quad & \sum_{j=1}^{n} c_j x_j \\ \textbf{\textit{subject to}} \quad & Ax \geq b \\ & 1 \geq x_j \geq 0 \quad \forall j \end{aligned}$$

*is a **covering linear program** if*

$$A \in \mathbb{R}_+^{m\times n} \text{ , } b \in \mathbb{R}_+^m \textit{ and } c \in \mathbb{R}_+^n$$

This definition simply ensures that all the coefficients of the constraints and of the objective function are non-negative.

Let us now introduce an example of covering LP that we will reuse later :

$$\begin{aligned} \textbf{minimize} \quad & x_1 + 2x_2 \\ \textbf{subject to} \quad & x_1 + 3x_2 \geq 2 \\ & 2x_1 + x_2 \geq 1 \\ & 1 \geq x_1, x_2 \geq 0 \end{aligned} \tag{1}$$

2.2 General idea

The idea of the method is to maintain a weight distribution over the set of constraints of a linear problem to solve, and to iteratively update those weights to get closer to an extreme point (if it exists).

We give to each constraint in the original LP a weight $w_i,\ i = 1, \ldots, m$. Those weights are initialized at 1. Then we sum up all the constraints according to their weights and obtain a new LP. Any optimal solution of the original LP is solution of this reduced problem, so the new problem will have at most the same cost as the previous one. We define an oracle for solving this reduced problem :

Definition 2 *An oracle that, given $p_1, \ldots, p_m \geq 0$ s.t $\sum_i p_i = 1$, outputs an optimal solution x^* to the following reduced linear problem :*

$$\begin{array}{ll} \textbf{\textit{minimize}} & \displaystyle\sum_{j=1}^{n} c_j x_j \\ \textbf{\textit{subject to}} & \displaystyle\left(\sum_{i=1}^{m} p_i A_i\right) \cdot x \geq \sum_{i=1}^{m} b_i p_i \\ & 1 \geq x \geq 0 \end{array}$$

As we want $\sum_i p_i = 1$ we will have to normalize the weights we originally defined by setting $p_i = \frac{w_i}{\sum_i w_i}$. This linear problem is in practice much easier to solve than the original one and we can design a simple algorithm for the oracle.

2.3 Implementation of the oracle

The oracle is given an objective function **minimize** $\sum_{i=1}^{n} c_i x_i$ and only one constraint that we can rewrite as $\sum_{i=1}^{n} d_i\, x_i \geq b$ (which is the weighted sum of all constraints). We also have $1 \geq x_i \geq 0\ \forall i$ and $d_i \geq 0\ \forall i$ since it is a covering problem.

The idea is to assign the maximum value (namely 1) to the variables that have the highest ratio *constraint coefficient/objective coefficient*. That way we will satisfy the constraint as fast as possible while maintaining a small objective function. Then assign zero to the other variables and possibly an intermediate value for the variable that is at the limit to make the constraint satisfied. Formally :

- Sort all variables x_i in non-increasing order according to d_i/c_i. This gives us a permutation $\sigma(j),\ 1 \leq j \leq n$.
- Let $k = min\{j : \sum_{i=1}^{j} d_{\sigma(i)} \geq b\}$ and $l = b - \sum_{i=1}^{k-1} d_{\sigma(i)}$.
- Set $x_{\sigma(i)} := 1$ for $i = 1, ..., k-1$.
- Set $x_{\sigma(k)} := l/d_{\sigma(k)}$.
- Set $x_{\sigma(i)} := 0$ for $i = k+1, ..., n$.

$$\sum_{i=1}^{n} d_i\, x_i = \sum_{i=1}^{n} d_{\sigma(i)}\, x_{\sigma(i)} = \sum_{i=1}^{k-1} d_{\sigma(i)}\, x_{\sigma(i)} + d_{\sigma(k)}\, x_{\sigma(k)} + \sum_{i=k+1}^{n} d_{\sigma(i)}\, x_{\sigma(i)} = \sum_{i=1}^{k-1} d_{\sigma(i)} + l = b$$

Therefore the constraint is barely satisfied and this ensures minimality for the objective function.

We now go back to the general multiplicative weight update method with an example.

2.4 Application to an example

If we apply the method we described to our example (1) (without normalization), we have two initials weights $w_1 = w_2 = 1$, and we sum all the constraints :

$$\begin{aligned} w_1(x_1 + 3x_2) + w_2(2x_1 + x_2) \geq w_1 \cdot 2 + w_2 \cdot 1 \Leftrightarrow \\ 3x_1 + 4x_2 \geq 3 \\ 1 \geq x_1, x_2 \geq 0 \end{aligned}$$

By using the oracle, an optimal solution to this reduced problem is $x_1 = 1, x_2 = 0$. But is this a feasible solution to our original problem ? By checking the constraints :

$$\begin{array}{ll} 2x_1 + x_2 = 2 \geq 1 & \text{OK} \\ x_1 + 3x_2 = 1 < 2 & \text{not OK} \end{array}$$

We will need to go back to the original problem and increase the weights of the unsatisfied constraints to give them more importance. Due to normalization this will decrease the weights of the satisfied ones. We will then recompute the weighted sum and iterate the process. The way the weights are modified and the precise output will be discussed in the next section, where we describe the algorithm for this method.

2.5 Algorithm

We first need to define the *width* ρ related to a given covering LP. The width represents the maximum value of the slack of a constraint (either by over-satisfying it or under-satisfying it) and we maximize over all the constraints.

Definition 3

$$\rho = \max_{1 \leq i \leq m} \{\max(b_i, A_i\mathbf{1} - b_i)\}$$

We can now write the multiplicative weight update algorithm for covering LPs, which will make use of this definition.

2.5.1 Initialization

Fix a step size $\eta \leq \frac{1}{2}$ and associate each constraint i with a weight $w_i^{(1)} = 1$, where the exponent denotes the current iteration.

2.5.2 Iterations

For $t = 1, 2, \ldots, T$

1. Let $x^{(t)}$ be the solution returned by the oracle with respect to values $\left(\frac{w_1^{(t)}}{\Phi^t}, \frac{w_2^{(t)}}{\Phi^t}, \ldots, \frac{w_m^{(t)}}{\Phi^t}\right)$ where $\Phi^t = \sum_{i=1}^m w_i^{(t)}$.
2. Let $m_i^{(t)} = \frac{A_i x^{(t)} - b_i}{\rho}$ be the quality of the solution with respect to the constraint i.
   Notice that $-1 \leq m_i^{(t)} \leq 1$. If $m_i^{(t)} \geq 0$, the constraint i is satisfied, else it is unsatisfied.
3. Update the weights : $w_i^{(t+1)} = w_i(t)(1 - \eta m_i^{(t)})$. This is the step that gave its name to the method: At the next iteration, constraints which were not satisfied ($m_i^{(t)} < 0$) will have a higher weight than satisfied constraints. The step size allows us to control the influence of the quality of the solution with respect to a given constraint in this process.

2.5.3 Output

We cannot just return one of the $x^{(t)}$, because at each iteration the solution will "advantage" some of the constraints (those with higher weights). Thus we output the average $\overline{x}$ in order to almost satisfy the maximum number of constraints.

$$\overline{x} = \sum_{t=1}^{T} \frac{x^{(t)}}{T}$$

2.6 Analysis

Theorem 4 *For any constraint i:*

$$0 \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}| + \frac{ln(m)}{\eta}$$

where $m = \Phi^{(1)} = \sum_i w_i^{(1)}$.

Corollary 5 *If $\eta = \frac{\epsilon}{4\rho}$ and $T = \lceil \frac{8\rho^2 ln(m)}{\epsilon^2} \rceil$, then for any constraint i, $A\overline{x} \geq b_i - \epsilon$.*

Which means that every constraint is almost satisfied and differs from true statisfaction only by ϵ. We see that if the oracle runs in polynomial time then the algorithm will be polynomial. We can also notice that the width is important for obtaining a polynomial execution time, as it appears in both η and T.

We start by giving a proof of this corollary:
Proof Using the expression of $m_i^{(t)}$, we can rewrite the inequality from the theorem as:

$$\begin{aligned}
0 \leq& \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{\rho} + \eta \sum_{t=1}^{T} |\frac{A_i x^{(t)} - b_i}{\rho}| + \frac{ln(m)}{\eta} \\
=& (1+\eta) \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{\rho} + 2\eta \sum_{<0} |\frac{A_i x^{(t)} - b_i}{\rho}| + \frac{ln(m)}{\eta}
\end{aligned}$$

since $\forall i, t$, $|m_i^{(t)}| \leq 1$, we have :

$$\leq (1+\eta) \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{\rho} + 2\eta T + \frac{ln(m)}{\eta}$$

multiplying by $\dfrac{\rho}{T}$:

$$0 \leq (1+\eta) \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} + 2\eta\rho + \frac{\rho}{T}\frac{ln(m)}{\eta}$$

choose η arbitrarily small, $\eta = \dfrac{\epsilon}{4\rho}$:

$$= (1+\eta) \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} + \frac{\epsilon}{2} + \frac{4\rho^2 ln(m)}{T\epsilon}$$

choose also $T = \lceil \dfrac{8\rho^2 ln(m)}{\epsilon^2} \rceil$:

$$\leq (1+\eta) \sum_{t=1}^{T} \frac{A_i x^{(t)} - b_i}{T} + \frac{\epsilon}{2} + \frac{\epsilon}{2}$$

$$
\begin{aligned}
&=(1+\eta)(A_i\left(\sum_{t=1}^{T}\frac{x^{(t)}}{T}\right)-b_i)+\epsilon\\
&=(1+\eta)(A_i\overline{x}-b_i)+\epsilon
\end{aligned}
$$

Therefore

$$
\begin{aligned}
0 &\leq A_i\overline{x}-b_i+\frac{\epsilon}{1+\eta}\\
b_i-\frac{\epsilon}{1+\eta} &\leq A_i\overline{x}\\
b_i-\epsilon &\leq A_i\overline{x}
\end{aligned}
$$

■

We will now proceed to the proof of the theorem. To do this, we will express a lower and an upper bound on $\Phi^{(T+1)}=\sum_{i=1}^{m}w_i^{(T+1)}$, i.e. the sum of the weights for the final output.

Proof
Upper bound : We are going to show that:

$$\Phi^{(T+1)} \leq m\exp(-\eta\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})$$

where $\mathbf{m}^{(t)}=(m_1^{(t)},\dots,m_m^{(t)})^T$ and $\mathbf{p}^{(t)}=(p_1^{(t)},\dots,p_m^{(t)})^T$.

We have $\forall t=1,\dots,T$:

$$
\begin{aligned}
\Phi^{(t+1)} &=\sum_{i=1}^{m}w_i^{(t+1)}\\
&=\sum_{i=1}^{m}(w_i^{(t)}(1-\eta m_i^{(t)}))\\
&=\Phi^{(t)}-\sum_{i=1}^{m}w_i^{(t)}\eta m_i^{(t)}\\
\text{since } p_i^{(t)}=\frac{w_i^{(t)}}{\Phi^{(t)}}\text{, then}&\\
&=\Phi^{(t)}-\sum_{i=1}^{m}\eta m_i^{(t)}p_i^{(t)}\Phi^{(t)}\\
&=\Phi^{(t)}(1-\eta\sum_{i=1}^{m}m_i^{(t)}p_i^{(t)})\\
&=\Phi^{(t)}(1-\eta\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})\\
\text{since } 1-x\leq\exp(-x)\quad\forall x&\\
&\leq\Phi^{(t)}\exp(-\eta\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})
\end{aligned}
$$

Now as $\Phi^{(1)}=m$, we can conclude by induction that :

$$\Phi^{(T+1)} \leq m\exp(-\eta\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})$$

Lower bound : We have

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = \prod_{t<T}(1-\eta m_i^{(t)}) \geq (1-\eta)^{\sum_{m_i>0} m_i^{(t)}}(1+\eta)^{-\sum_{m_i<0} m_i^{(t)}}$$

where we used that:

- $(1-\eta)^x \leq (1-\eta x)$ if $x \in [0,1]$
- $(1+\eta)^{-x} \leq (1-\eta x)$ if $x \in [-1,0]$

This is the lower bound we need. Now, by combining both bounds we can write:

$$\begin{aligned}(1-\eta)^{\sum_{m_i>0} m_i^{(t)}}(1+\eta)^{-\sum_{m_i<0} m_i^{(t)}} &\leq m\exp(-\eta\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})\\ ln(1-\eta)(\sum_{m_i>0} m_i^{(t)}) - ln(1+\eta)(\sum_{m_i<0} m_i^{(t)}) &\leq ln(m) - \eta\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)}\end{aligned}$$

But, we also have:

$$\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} = \sum_{i=1}^{m} m_i^{(t)}p_i^{(t)} = \sum_{i=1}^{m}\frac{A_i x^{(t)} - b_i}{\rho}p_i^{(t)} = \frac{1}{\rho}\underbrace{\left((\sum_{i=1}^{m} p_i^{(t)}A_i)x^{(t)} - (\sum_{i=1}^{m} b_i p_i^{(t)})\right)}_{\geq 0}$$

Using the constraint of the oracle reduced problem, we know that the term in parenthesis of this expression is non-negative. Therefore $\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} \geq 0$.
Multiplying by $-1/\eta$ and rearranging we get :

$$\begin{aligned}0 \leq& \frac{1}{\eta}\left(ln(m) - ln(1-\eta)(\sum_{m_i>0} m_i^{(t)}) + ln(1+\eta)(\sum_{m_i<0} m_i^{(t)})\right)\\ =& \frac{ln(m)}{\eta} - \frac{ln(1-\eta)}{\eta}(\sum_{m_i>0} m_i^{(t)}) + \frac{ln(1+\eta)}{\eta}(\sum_{m_i<0} m_i^{(t)})\\ =& \frac{ln(m)}{\eta} + \frac{1}{\eta}ln\left(\frac{1}{1-\eta}\right)(\sum_{m_i>0} m_i^{(t)}) + \frac{ln(1+\eta)}{\eta}(\sum_{m_i<0} m_i^{(t)})\end{aligned}$$

As $\eta \leq \frac{1}{2}$, we have $ln\left(\frac{1}{1-\eta}\right) \leq \eta + \eta^2$ and $ln(1+\eta) \geq \eta - \eta^2$. Taking into account the signs of the two sums, we have:

$$\begin{aligned}0 \leq& \frac{ln(m)}{\eta} + \frac{1}{\eta}(\eta+\eta^2)\sum_{m_i>0} m_i^{(t)} + \frac{1}{\eta}(\eta-\eta^2)\sum_{m_i<0} m_i^{(t)}\\ =& \frac{ln(m)}{\eta} + \sum_{m_i>0} m_i^{(t)} + \eta\sum_{m_i>0} m_i^{(t)} + \sum_{m_i<0} m_i^{(t)} - \eta\sum_{m_i<0} m_i^{(t)}\\ =& \frac{ln(m)}{\eta} + \sum_{i=1}^{m} m_i^{(t)} + \eta(\sum_{m_i>0} m_i^{(t)} - \sum_{m_i<0} m_i^{(t)})\\ =& \frac{ln(m)}{\eta} + \sum_{i=1}^{m} m_i^{(t)} + \eta\sum_{i=1}^{m}|m_i^{(t)}|\end{aligned}$$

We have the result. ■

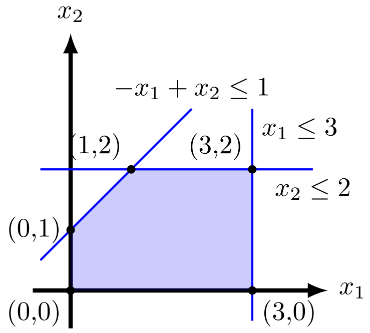
3 The Simplex Method

In this section we will talk about the simplex method which is used to solve LP problems. It was invented in 1947 by George Dantzig[1] and remains one of the most important algorithm of the 20th century. Despite the fact that its running time can be proved to be exponential in worst case, it runs very fast in practice and is widely used in industry even though we now know other algorithms that do run in polynomial time but that are slower for pratical purposes.

3.1 Example

We will introduce this method using a concrete example. Let us now focus on the following LP with its associated polytope :

$$\begin{array}{rl} \textbf{maximize} & x_1 + x_2 \\ \textbf{subject to} & -x_1 + x_2 \leq 1 \\ & x_1 \leq 3 \\ & x_2 \leq 2 \\ & x_1,\ x_2 \geq 0 \end{array}$$



The simplex method does not work with inequalities hence we have to change inequality signs with equalities. This can be achieved by introducing slack variables $s_1, .., s_m$ that *compensate* for the equalities. We also add a new line that will represent our current objective function which is expressed as $z =$ *objective function*. At the beginning the objective function is the one given in the original problem. Also we will remember that all variables must be non-negative including the slack variables. This gives us the following modified LP :

$$-x_1 + x_2 + s_1 = 1 \quad (1)$$
$$x_1 + s_2 = 3 \quad (2)$$
$$x_2 + s_3 = 2 \quad (3)$$
$$z = x_1 + x_2$$

In this initial configuration we set all $x_i := 0$ which implies the following assignation for the slack variables $s_1 := 1$, $s_2 := 3$, $s_3 = 2$ according to the constraints. Now the solving procedure can really start. We will maintain a *simplex tableau* which consists of all the constraints and the objective function, and we will always write the constraints such that the non-zero variables are on the left-hand side. These are called basic variables. This gives us:

$$s_1 = 1 + x_1 - x_2 \quad (1)$$
$$s_2 = 3 - x_1 \quad (2)$$
$$s_3 = 2 - x_2 \quad (3)$$
$$z = x_1 + x_2$$

$$x_1 := 0 \quad x_2 := 0 \quad s_1 := 1 \quad s_2 := 3 \quad s_3 := 2$$

[1]`http://www.ams.org/notices/200703/fea-cottle.pdf`

Now the goal is to maximize the objective function $x_1 + x_2$. We will iteratively increase as much as possible one variable appearing in the objective function that has a positive coefficient (since we want to maximize). In our case let us consider x_2.[2] Clearly since there is a positive coefficient in front of it (namely 1), an increase of x_2 will get us closer to the optimal solution (remember that currently $x_1 := 0$, $x_2 := 0 \Rightarrow z = 0$) but in the process we will necessarily have to decrease other variables to preserve the equalites. We now need to check by how much we can increase x_2 without breaking non-negativity for all variables. (1) tells us that $x_2 \leq 1$, (2) says nothing about x_2 and (3) implies $x_2 \leq 2$ therefore we increase x_2 by 1.

We will compensate this increase by modifying all the variables appearing on the left-hand sides of the constraints. Note that for the constraint that imposes the value for the variable ((1) in our case) the basic variable will become zero. Therefore we decrease s_1 and s_3 by 1.

We then rewrite the constraint that dictates the value for the chosen variable (constraint (1)) by swapping the left-hand side and the variable ($s_1 \leftrightarrow x_2$). This operation is called *pivoting.* We then substitute the new right-hand side of x_2 in the other constraints and in the objective function:

$$\begin{aligned} x_2 &= 1 + x_1 - s_1 && (1) \\ s_2 &= 3 - x_1 && (2) \\ s_3 &= 2 - \underbrace{(1 + x_1 - s_1)}_{=x_2} && (3) \\ z &= x_1 + \underbrace{(1 + x_1 - s_1)}_{=x_2} \end{aligned} \quad \Longrightarrow \quad \begin{aligned} x_2 &= 1 + x_1 - s_1 && (1) \\ s_2 &= 3 - x_1 && (2) \\ s_3 &= 1 - x_1 + s_1 && (3) \\ z &= 1 + 2x_1 - s_1 \end{aligned}$$

$$x_1 := 0 \quad x_2 := 1 \quad s_1 := 0 \quad s_2 := 3 \quad s_3 := 1$$

Now we see that only one variable has a positive coefficient in the objective function, namely x_1 and we will therefore try to increase it. Constraint (1) does not give any upperbound on x_1, (2) imposes $x_1 \leq 3$ and (3) enforces $x_1 \leq 1$ thus we increase x_1 by 1 which leads s_2 and s_3 to decrease by 1 and x_2 to increase by 1. We now pivote x_1 in constraint (3) and substitute its right-hand side in the rest :

$$\begin{aligned} x_2 &= 1 + \underbrace{(1 - s_3 + s_1)}_{=x_1} - s_1 && (1) \\ s_2 &= 3 - \underbrace{(1 - s_3 + s_1)}_{=x_1} && (2) \\ x_1 &= 1 - s_3 + s_1 && (3) \\ z &= 1 + 2\underbrace{(1 - s_3 + s_1)}_{=x_1} - s_1 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} x_2 &= 2 - s_3 && (1) \\ s_2 &= 2 + s_3 - s_1 && (2) \\ x_1 &= 1 - s_3 + s_1 && (3) \\ z &= 3 - 2s_3 + s_1 \end{aligned}$$

$$x_1 := 1 \quad x_2 := 2 \quad s_1 := 0 \quad s_2 := 2 \quad s_3 := 0$$

Again the only choice is to increase s_1 since it is the only variable that has a positive coefficient. (1) gives nothing about s_1, (2) dictates $s_1 \leq 2$ and (3) does not upperbound s_1. We therefore increase s_1 by 2, this implies to decrease s_2 by 2 and increase x_1 by 2, pivote s_1 and s_2 in (2) and substitute the new right-hand side of s_1.

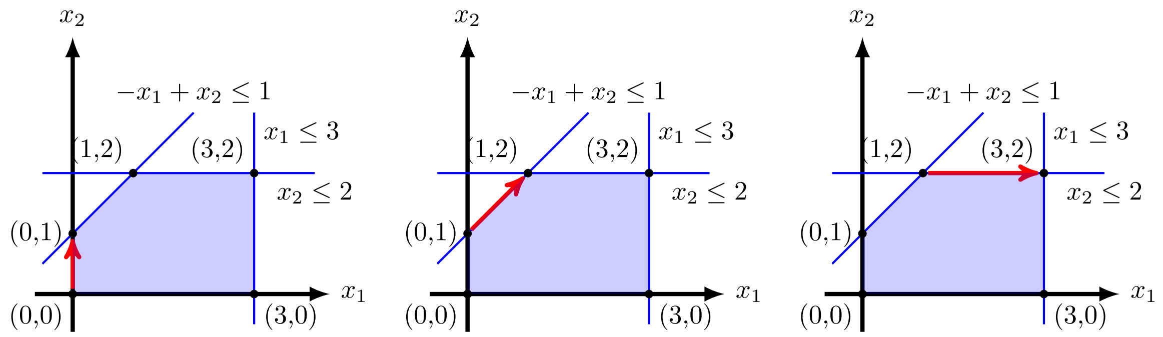
[2] Note that we could have chosen to increase x_1 since it has also a positive coefficient. Actually there is a whole *industry* behind the choice of the variable to increase in each step. A simple idea could be to choose the one with the largest coefficient but there can be other strategies.

$$\begin{aligned} x_2 &= 2 - s_3 && (1) \\ s_1 &= 2 + s_3 - s_2 && (2) \\ x_1 &= 1 - s_3 + \underbrace{(2 + s_3 - s_2)}_{=s_1} && (3) \\ z &= 3 - 2s_3 + \underbrace{(2 + s_3 - s_2)}_{=s_1} \end{aligned} \quad \Longrightarrow \quad \begin{aligned} x_2 &= 2 - s_3 && (1) \\ s_1 &= 2 + s_3 - s_2 && (2) \\ x_1 &= 3 - s_2 && (3) \\ z &= 5 - s_3 - s_2 \end{aligned}$$

$$x_1 := 3 \quad x_2 := 2 \quad s_1 := 2 \quad s_2 := 0 \quad s_3 := 0$$

Now we cannot increase any variables since they all have negative coefficients which would deteriorate the objective function. Therefore we stop the procedure and we can read the optimal solution $z = 5 - 0 - 0 = 5$ achieved with $x_1 = 3$, $x_2 = 2$. (We ignore the values of the slack variables).

If we look carefully at the values taken by x_1 and x_2 throughout the procedure we can see that at each step we actually follow an edge of the polytope that brings us closer to the optimal solution :



How can we be sure that this is the optimal solution ? Let us consider any feasible solution $\{\bar{x_1}, \bar{x_2}, \bar{s_1}, \bar{s_2}, \bar{s_3}\}$. We know that any of such feasible solution has to satisfy the equalities in the tableau and since all variables must be non-negative it is clear that the objective function is upperbounded by 5: $z = 5 - \bar{s_3} - \bar{s_2} \leq 5$. Therefore we cannot hope to achieve better than 5 in the objective function and the values we got for x_1 and x_2 lead exactly to 5.

3.2 Technicalities

We present here some problems that can arise when trying to apply the simplex method. We will not go into the details on how to get around these issues but it is good to be aware of them.

Unboundness : It might happen that the chosen variable to increase is not bounded by any constraint. This is the case when the polytope defined by the original constraints (the ones with inequalities) is not bounded and the optimal solution is infinite.

Degeneracy : During the execution of the simplex algorithm it might happen that we cannot increase any variable but still need to pivot two of them in order to proceed. The swapped variable will become a basic variable (on the left-hand side) even though its current value will be zero. This does not prevent the algorithm to find the final solution but we have to be careful not to cycle if we keep pivoting the same variables without changing their values. This can be avoid using different strategies such as a lexicographic ordering of the variables.

Infeasability : If the constraints define an empty polytope then there is no feasible solution. This can be detected easily by the simplex algorithm.

Initial vertex : In our exemple it was quite clear that $x_1 := 0$, $x_2 := 0$ was a feasible starting solution. In the general case the initial assignment is not trival and demands to solve another linear program to get the starting values. This is called the Phase I of the simplex method followed by the Phase II which is what we showed above.

3.3 Exercise

We present now another application of the simplex algorithm this time with a minimization problem. We will not go as deep in the details as we did above but still show the analysis at each step. Note that in this case we will choose the variables that have negative coefficients to increase since it is a minimization problem.

$$\begin{array}{rlcll}
\textbf{minimize} & -2x_1 + x_2 & & s_1 = 3 - x_2 & (1)\\
\textbf{subject to} & x_2 \leq 3 & \Longrightarrow & s_2 = 3 - x_1 + 3x_2 & (2)\\
& x_1 - 3x_2 \leq 3 & & z = -2x_1 + x_2 & \\
& x_1,\ x_2 \geq 0 & & x_1 := 0,\quad x_2 := 0,\quad s_1 := 3,\quad s_2 := 3 &
\end{array}$$

$$\nearrow x_1 \longrightarrow\ x_1 \leq ?\ (1),\ x_1 \leq 3\ (2) \longrightarrow x_1 := 3,\ s_2 := 0$$

$$\begin{array}{llcll}
s_1 = 3 - x_2 & (1) & & s_1 = 3 - x_2 & (1)\\
x_1 = 3 - s_2 + 3x_2 & (2) & \Longrightarrow & x_1 = 3 - s_2 + 3x_2 & (2)\\
z = -2(3 - s_2 + 3x_2) + x_2 & & & z = -6 + 2s_2 - 5x_2 & \\
x_1 := 3,\quad x_2 := 0,\quad s_1 := 3,\quad s_2 := 0 & & & &
\end{array}$$

$$\nearrow x_2 \longrightarrow x_2 \leq 3\ (1),\ x_2 \leq \infty\ (2) \longrightarrow x_2 := 3,\ s_1 := 0,\ x_1 := 12$$

$$\begin{array}{llcll}
x_2 = 3 - s_1 & (1) & & x_2 = 3 - s_1 & (1)\\
x_1 = 3 - s_2 + 3(3 - s_1) & (2) & \Longrightarrow & x_1 = 12 - s_2 + -3s_1 & (2)\\
z = -6 + 2s_2 - 5(3 - s_1) & & & z = -21 + 2s_2 + 5s_1 & \\
x_1 := 12,\quad x_2 := 3,\quad s_1 := 0,\quad s_2 := 0 & & & &
\end{array}$$

no negative weight $\Rightarrow OPT = -21$ at $x_1 = 12,\ x_2 = 3$

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[2] Gerard Cornuejols and Michael Trick. Quantitative methods for the management sciences, 1998. `http://mat.gsia.cmu.edu/classes/QUANT/NOTES/chap7.pdf`.

[3] Anupam Gupta. Solving lps/sdps using multiplicative weights. `http://http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture17.pdf`.

[4] Jiri Matousek and Bernd Gärtner. *Understanding and using linear programming*. Springer Science & Business Media, 2007.

[5] Scott Provan. The simplex method, 2013. `http://www.unc.edu/depts/stat-or/courses/provan/STOR614_web/lect03_simplex.pdf`.