Revised Simplex Algorithm

Kenton Lam

June 7, 2019

The revised simplex algorithm is fundamental to linear programming but can be complicated at first glance. I've formulated it here as plainly and logically as possible to help myself and (hopefully) others.

1 Problem Definition

In general, a linear programming problem is defined as follows.

$$\begin{aligned} \text{maximise } & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{such that } & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

Minimisation problems can be done by maximising $-\boldsymbol{c}^\top \boldsymbol{x}$.

Here, $\boldsymbol{x}$ are the unknown variables we want to optimise for, expressed as an n-vector. $\boldsymbol{c}^\top \boldsymbol{x}$ is our (linear) *objective function* and $\boldsymbol{c}$ is an n-vector of *objective coefficients*.

A is an $m \times n$ matrix of constraint coefficients, corresponding to m linear constraints. The m-vector $\boldsymbol{b}$ contains the right hand side of these constraints.

1.1 Glossary

Some terms which will be used are:

- *solution* – Any $\boldsymbol{x}$ vector, not to be confused with *the* solution to the optimisation problem.
- *feasible solution* – A solution which satisfies all the constraints in the problem (we also have *infeasible solutions*).
- *basis* – Non-zero components of $\boldsymbol{x}$ in a particular solution.
- *basic feasible solution (BFS)* – Starting solution for the simplex algorithm. This should be easy to calculate.

1.2 Inequality constraints

You may have noticed that the definition above uses equality. But our model needs to handle inequality constraints, otherwise it wouldn't be very useful! Suppose we have a constraint of the form

$$\boldsymbol{a}_0 \cdot \boldsymbol{x} \leq b_0.$$

There's a clever trick we can use. We add a dummy variable, called a slack variable, so the inequality is expressed as an equality. Here, we use x_0.

$$\boldsymbol{a}_0 \cdot \boldsymbol{x} + x_0' = b_0$$

With the constraint that $x_0' \geq 0$, this ensures the inequality is maintained. We add x_0' to the $\boldsymbol{x}$ vector and we add a 0 to $\boldsymbol{c}$ because x_0' is not in the objective function. To A, we add a column which is zero everywhere except in row corresponding to this inequality. In the basic feasible solution, we can set $x_0 = 0$ and $x_0' = b_0$.

Greater than inequalities are more complicated. Applying the same logic as less-then constraints, we can try

$$x_1 \geq 40 \implies x_1 - x_1' = 40.$$

However, this has the problem of no BFS. We cannot blindly set $x_1 = 40$ because this may invalid $\leq$ constraints and we certainly can't set x_1' to a negative value. The solution is to add another variable, called an artificial variable, and penalise it in the objective function. That is,

$$x_1 - x_1' + x_1^* = 40.$$

We add x_1' and x_1^* to $\boldsymbol{x}$ and because we are maximising, we add a coefficient $M \ll 0$ to $\boldsymbol{c}$. This will ensure x_1^* is zero in the optimal solution. Then, our BFS contains $x_1 = x_1' = 0$ and $x_1^* = 40$.

2 The Algorithm

Essentially, simplex works by assuming that in any solution, a fixed number of the $\boldsymbol{x}$ variables are 0. Exactly which variables are 0 can and will change. The set of variables which are *not* 0 are called basis variables. By moving specific variables into and out of the basis, the simplex method computes the optimal solution to the problem.

We can rewrite the optimisation problem in terms of basis and non-basis variables. We split A into B and N, $\boldsymbol{x}$ into $\boldsymbol{x}^B$ and $\boldsymbol{x}^N$, and similarly for c. 'B' refers to variables in the basis and 'N' those which aren't.

$$\text{maximise } \begin{bmatrix}\boldsymbol{c}^B\\ \boldsymbol{c}^N\end{bmatrix}^\top \begin{bmatrix}\boldsymbol{x}^B\\ \boldsymbol{x}^N\end{bmatrix}$$
$$\text{such that } \begin{bmatrix}B \mid N\end{bmatrix} \begin{bmatrix}\boldsymbol{x}^B\\ \boldsymbol{x}^N\end{bmatrix} = \boldsymbol{b}$$

Again, remember that the basis variables *will change* as the algorithm iterates. It only makes sense to say a variable is "in the basis" if you specify which solution you are talking about.

Recall that our BFS is 0 for all variables in the original optimisation problem and slack/artificial variables set as described above. We begin with the BFS as our solution.

1. By construction, the solution satisfies the constraint $A\boldsymbol{x} = \boldsymbol{b}$. By assumption, the non-basis variables are 0 so
$$\boldsymbol{x}^N = 0 \implies B\boldsymbol{x}^B = \boldsymbol{b} \implies \boldsymbol{x}^B = B^{-1}\boldsymbol{b}.$$
Note that the objective value of this solution is given by $z^B = \boldsymbol{c}^B \cdot \boldsymbol{x}^B$.

2. Compute the *dual variables* as
$$\boldsymbol{y} = (B^{-1})^\top \boldsymbol{c}^B \quad \text{or} \quad \boldsymbol{y}^\top = (\boldsymbol{c}^B)^\top B^{-1}.$$

3. We need to choose which variable which enter the basis. For each j not in the basis, let $\boldsymbol{\rho}_j$ denote its corresponding column in A. Compute the *reduced cost*
$$c'_j = c_j - \boldsymbol{y} \cdot \boldsymbol{\rho}_j.$$
   (a) If for all j, $c'_j \leq 0$ then the current solution is optimal. Stop.
   (b) Otherwise, choose j^* such that c'_{j^*} is maximised. This will be the entering variable.

4. Now, we determine the leaving variable. Compute the *direction vector* of the entering variable as
$$\boldsymbol{\alpha} = B^{-1}\boldsymbol{\rho}_{j^*}.$$
   (a) If no component of $\boldsymbol{\alpha}$ is positive, the problem is unbounded. Stop.
   (b) Otherwise, choose r from the current basis such that $\alpha_r > 0$ and x_r^B/α_r is minimised. This is the leaving variable.

5. Recompute B, $\boldsymbol{x}^B$ and $\boldsymbol{c}^B$ after removing the leaving variable and adding the entering variable. Go to step 1.

3 Sensitivity Analysis

Once we have an optimal solution $\boldsymbol{x}$, its dual variables $\boldsymbol{y}$ can give us insights into the solution. Specifically, we can calculate the effect of small changes to the constraint RHS $\boldsymbol{b}$ and the objective coefficients $\boldsymbol{c}$.

Let $\boldsymbol{e}^j$ be a vector with 1 in the j-th position and 0 elsewhere.

- We perturb $\boldsymbol{b}$ by $\boldsymbol{b} + \Delta\boldsymbol{e}^j$ by some amount Δ. This changes the j-th constraint's RHS. The new objective value is given by

$$\begin{aligned} z^\Delta = \boldsymbol{y}^\top(\boldsymbol{b} + \Delta\boldsymbol{e}^j) &= \boldsymbol{y}^\top\boldsymbol{b} + \boldsymbol{y}^\top\Delta\boldsymbol{e}^j \\ &= z^B + \Delta y_j \quad (\boldsymbol{y}^\top\boldsymbol{b} = z^B \text{ by duality}) \end{aligned}$$

  This is valid subject to

$$\boldsymbol{x}^{B\Delta} = B^{-1}(\boldsymbol{b} + \Delta\boldsymbol{e}^j) = \boldsymbol{y} + \Delta(B^{-1})^\top\boldsymbol{e}^j \geq \boldsymbol{0}.$$

  In other words, for this Δ, each variable in the basis must remain non-negative.

- Consider a perturbation of $\boldsymbol{c} + \Delta\boldsymbol{e}^j$ to $\boldsymbol{c}$, changing the j-th variable's objective coefficient. The optimal solution remains optimal if the following conditions hold.

  - If j is a non-basis variable, the reduced cost,

$$c'_j = (c_j + \Delta) - \boldsymbol{y}^\top\boldsymbol{\rho}_j$$

    needs to remain non-positive. This tells us how much each objective coefficient needs to change before its variable enters the optimum basis.

  - If j is a basis variable, we need

$$\boldsymbol{y}^\Delta = B^{-1}(\boldsymbol{c}^B + \Delta\boldsymbol{e}^j) = \boldsymbol{y} + \Delta(B^{-1})^\top\boldsymbol{e}^j \geq \boldsymbol{0}.$$

    That is, the new dual variables need to be non-negative. With this, we can compute an interval for Δ where this solution remains optimal.

Note that these are only valid for changing one constraint or objective coefficient at a time.

3.1 Gurobi

Gurobi computes these for us, under the following attributes.

- For the j-th constraint 'con', 'con.Pi' is y_j, 'con.Slack' is the value of its slack variable, and 'con.SARHSLow' and 'con.SARHSUp' are the bounds on Δ.
- For the j-th variable 'var', 'var.RC' is the reduced cost c'_j, and the attributes 'var.SAObjLow' and 'var.SAObjUp' are bounds on Δ.