The Simplex Algorithm

We keep mentioning the simplex algorithm as a method to solve linear programming problems. In this section, we describe this algorithm assuming we are using the standard form. The Cormen text presents the simplex algorithm assuming slack form, and the student is referred to the text for that treatment.

Simplex in the Standard Form

The main intuition behind the simplex algorithm is to note that a vertex occurs at the point n hyperplanes meet in a common point. Therefore, at a given point in the algorithm, we seek a subset of constraints such that, if a unique point satisfies them with equality and that point is feasible, then it must be a vertex. Furthermore, we note that two vertices are neighbors if they share $n-1$ of their constraints in common.

Given this observation, we outline the simplex algorithm as follows. On each iteration of the algorithm we first check to see if the current vertex is optimal. If not, we then determine which neighboring vertex to consider. Both of these tasks are easy if the starting vertex is at the origin.

Why the origin? Let's consider the form of the objective function. If we are working with an objective function of the form "maximize $\mathbf{c}^T\mathbf{x}$," then we will find that the function is optimal if and only if all $c_i \leq 0$. Thus all we have to do is scan the coefficients of the objective function. Furthermore, if there exists some $c_i > 0$, then the origin is not at the optimal location and we can raise the objective value simply by raising the value of the corresponding variable x_i. So the algorithm proceeds by picking an x_i such that $c_i > 0$ and increase its value until we "hit some other constraint." This happens at the point we reach equality in a constraint utilizing that variable.

The tricky part of the algorithm comes once we have identified the neighbor. At that point, we need to transform the entire problem to a new coordinate space where the new vertex is at the origin. This is done by observing that, for some constraint $\mathbf{a}_i \cdot x \leq b_i$, we need to find a new y_i such that $y_i = b_i - \mathbf{a}_i \cdot x$. We then take this and substitute into the other constraints defining the hyperplanes that meet at this point.

Complexity

The simplex algorithm is, perhaps, the most widely used algorithm to solve linear programming problems. Is it the fastest? Unfortunately, no. To analyze the time complexity of the simplex algorithm, assume we have n decision variables $x_1, \ldots, x_n$ and m constraints. As a hillclimbing algorithm, we need to consider the complexity associated with analyzing (and transforming) a single vertex, and we need to consider the worst case number of vertices to visit.

We start by considering the cost of a single iteration. A vertex in the simplex corresponds to a unique solution to n of the constraints. (We have degeneracy when there are more than n constraints satisfied.) Each of the neighboring vertices will share $n-1$ of these constraint, thus we have $n \cdot m$ neighbors to consider for moving. Technically, when considering a possible neighbor, we need to verify that neighbor is a vertex (or generate the vertex directly) and then determine if the point is feasible. Determinine/calculate the vertex involves solving a system of n linear equations in n unknowns using something like Gaussian elimination. Gaussian elimination has complexity $O(n^3)$. Thus an iteration has complexity $O(mn^3)$.

Note that we can improve on this bound by observing that the pivot process, whereby we place our target vertex at the origin, involves a rewrite step. Once the objective function and constraints have been rewritten, evaluating the point is simple. Indeed, the complexity of the rewrite process is $O(n(m+n))$. Then to pick the best neighbor for that transformation, we note that we are working with an objective function of the form "maximize $k + \mathbf{c} \cdot \mathbf{y}$" (for a transformed coordinate space over $\mathbf{y}$). As long as we pick a variable where c_i is positive, we will continue to improve the objective value. Since there are only n variables in the objective function, and we are increasing one variable at a time, we have our final complexity of an iteration being $O(mn)$. Thus the total complexity is $O(n(m+n)) + O(mn) = O(mn + n^2)$.

The "hard part" of the algorithm comes in determining the number of vertices to be visited. Recall that we have a total of $n+m$ variables and constraints to consider. In the worst case, we have to consider all possible sets of variable-constraint combinations where we have solutions of constraints. Thus there are, in the worst case, $\binom{n+m}{n}$ possible vertices. Thus, in the worst case, the simplex algorithm runs in exponential time. Fortunately, these cases occur rarely in practice, and the typical behavior of the simplex algorithm is very fast.