ORIE 6300 Mathematical Programming I October 12, 2016

Recitation 8

Lecturer: Calvin Wylie *Topic: Angela Zhou*

Column Generation

This is a problem from the paper industry. Paper is produced in W inch long rolls called *raws* in which W is very large. These raws are cut into smaller lengths called *finals* for sale according to demand. Suppose there is demand for b_i rolls of length $s_i \leq W$, where $i = 1, \dots, m$. The goal is to find a way to cut raws into finals such that we use the minimum possible number of raws. A clear upper-bound for this problem is $\sum_{i=1}^{m} b_i = N$ (i.e., using one complete roll of paper to produce each one of the rolls needed).

This will be an integer program since we can't decide to cut fractional numbers of raws. There are multiple ways to model this problem,but we want to formulate it as an integer linear program. We want the integer formulation to lead to a good LP relaxation, whose solution is not too "off" from the solution to the integer program.

One possible formulation uses the idea of a "pattern": given a raw of a certain length, you have certain lengths of demand that you need to satisfy, so you always cut your rod into discrete pieces of some of these lengths. There's only a finite number of patterns that you can cut the rod in.

We will represent a pattern as a column vector A_j, where the length is m, and each component of the column vector tells you how many pieces of size i do you cut from the raw. For the pattern to be feasible, elements are all positive, and the pattern cannot represent something that would require length greater than W to cut.

$$\sum_{i}^{m} a_{ij} s_i \leq W$$

for example: $W = 10$, $s_1 = 5, s_2 = 3, s_3 = 2$. One possible pattern is $A_1 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$, $A_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $A_2 = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}$, ... there are 7 feasible patterns.

Let x_j be the number of raws to cut with pattern j. Let A be the matrix of all patterns.

Our problem can be written as

$$\min \sum_{j} x_j$$
$$s.t. Ax \geq b$$
$$x \geq 0 \text{ integer}$$

So the integer program is

$$\begin{aligned} \text{Minimize } & \sum_j x_j \\ \text{subject to } & Ax \geq b \\ & x \geq 0 \qquad \text{are integers.} \end{aligned}$$

where a_{ij} = the number of finals of size s_i in the pattern j.

How good is the LP relaxation? If we removed the integer constraint and solved the LP how good would this solution be? If we solve the LP with the simplex method, we know we would get at most m of the $x_j > 0$ (basic variables), since the size of our basis is m. If we round every $x_j > 0$ up to the nearest integer, since A is nonnegative, this gives us a feasible solution to the integer program. The gap between the minimum LP solution and the true integer solution is at most m.

This is good because this m is independent of the scale of the demands - there are formulations where the gap between the LP solution and bounding can grow without bound if you scale up the demands. The optimality gap is only proportional to how many sizes there are, not the scale of your demand.

The issue with solving this problem is that once we grow in dimension, the number of possible patterns grows exponentially in the number of possible sizes. It won't be possible to fit the LP in memory.

Issue: # of the columns could be very large. We can exploit that for the simplex method to work, you only need to keep the current basis in memory. The size of the basis is only $m << n$.

0.1 Column Generation

As there are exponentially many feasible patterns, we do not want to consider them all simultaneously. To run the revised simplex method, we need only to:

1. find initial basic feasible solution.
2. decide if all non-basic j has $\bar{c}_j \geq 0$, or find A_j such that $\bar{c}_j < 0$.

We can generate an initial basis of feasible patterns $A = [A_1, \ldots, A_m]$, where A_j is a pattern that produces $\lfloor w/s_j \rfloor$ finals of type j and no other finals, so $A_{jj} = \lfloor w/s_j \rfloor$ and $A_{ij} = 0$ for all $i \neq j$. As the sub-matrix of these columns is diagonal with positive diagonal entries, it is full rank and forms a basis. A feasible x for these patterns is $x_j = b_j / \lfloor w/s_j \rfloor$, and $x_i = 0$ for $i \notin B$, as then $Ax = b$.

Given a basis, we can compute the reduced costs directly from the formulas $y = (A_B^T)^{-1} c_B$ and $\bar{c}_j = c_j - A_j^T y$. As there are exponentially many c_j and A_j, we need an efficient method to find a $\bar{c}_j$ which will be negative. However, $c_j = 1$ for every j, so the reduced cost for j is non-negative if and only if $A_j^T y \leq 1$. Thus all reduced costs are non-negative if and only if $A_j^T y \leq 1$ for all j. This motivates the following integer program:

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^{m} y_i a_i \\ \text{subject to } & \sum_{i=1}^{m} s_i a_i \leq W \\ & a_i \geq 0 \qquad\qquad \text{are integers.} \end{aligned}$$

where a_i is the number of finals of size i to cut for pattern a, s_i is the size of the ith final type, and the constraint ensures that the pattern is feasible. As any such feasible pattern a will be a column of A, so solving this integer program will give us the column with the least reduced cost. Thus if the optimal objective function value is less than 1, then there are no columns with negative reduced costs, otherwise the optimal a is a column with negative reduced cost. So we are optimal if $A_j^T y \leq 1, \forall j$. It's not feasible to check this for all j! What we can do is write another optimization which generates some A_j that proves that there is no column A_j that makes this less than or equal to 1; or if there is something greater than 1, provides it to us.

Consider the optimization problem, where a_i is like choosing a particular pattern: we want to constrain it to feasible patterns.

$$\begin{aligned} & \max_a \sum_{i=1}^{m} y_i a_i \\ \text{s.t. } & \sum_{i=1}^{m} s_i a_i \leq W \\ & a \geq 0 \text{ integer} \end{aligned}$$

If we can indeed maximize this and the optimal value is less than 1, this certifies optimality of our basis. If instead we maximize this and find that the objective value is greater than 1, that optimal a corresponds to a pattern that we should introduce into our basis!

If $y^T a^* \leq 1$ our current basis is optimal since $\bar{c}_j \leq 0$ for all non-basic j.

If $y^T a^T > 1$ then a should be introduced into basis.

We should verify that we can solve this subproblem - it's an instance of the knapsack problem. Let W be the weight constraint of the knapsack, y_i be the value of the items we are putting in. This can be solved via dynamic programming. However, it's NP-Hard so this is probably less doable for large n.

The dynamic programming solution is to take your first item, assume you put one item in the knapsack, what is the value of the other solutions? You work backwards from there: it's easy to consider the maximum value obtained by adding one item.

Analogously there exists constraint generation where there's too many constraints to consider, rather than variables. It's basically the dual of this approach! If you take the dual of an LP with too many variables you'll get a program with too many constraints. e.g. for the TSP LP relaxation, the best approach uses constraint generation. For the TSP you have an exponential number of 'hard' constraints. All these techniques rely on the existence of an efficient way of solving the subproblems.