ORIE 6300 Mathematical Programming I October 30, 2008

Lecture 17

Lecturer: David P. Williamson *Scribe: Ross Anderson*

1 The Cutting Stock Problem

Recall from last lecture the cutting stock problem. Paper is produced in long rolls, or *raws*, of some fixed length W. The customers demand b_j rolls of length $s_j < w$, called *finals*, for $j = 1, \dots, m$. The producer cuts the finals from the raws, trying to use as few raws as possible while producing all demanded finals.

1.1 A Formulation

Suppose we define a *pattern* (or configuration) $p \in \mathbb{R}^m : p_i \in \mathbb{N}_+$ to be a cutting of a raw producing p_i finals of type i, where $i = 1, \dots, m$ enumerates the sizes of the finals demanded. A pattern is *feasible* if $\sum_{i=1}^m p_i w_i \le W$ and *maximal* if $\sum_{i=1}^m p_i w_i > W - \min_i w_i$. Clearly, there are only finitely many feasible patterns, enumerated by $j = 1, \dots, n$. We define the matrix $A \in \mathbb{R}^{m \times n}$ to have the jth column as the jth feasible pattern. Letting $x_j \in \mathbb{N}$ for $j = 1, \dots, m$ be the decision variable indicating the number of times pattern j is cut from a raw, and $b \in \mathbb{R}^m$ where b_i is the total demand for finals of size i, this leads to the alternate formulation:

$$\begin{array}{ll} \min & \sum_{i=1}^{n} x_i \\ \text{subject to} & Ax \ge b \\ & x \in \mathbb{N}^m. \end{array}$$

Note that while the constraint could be an equality constraint, by making it a greater than or equal to constraint we ensure that the integer program always has a solution with a basis of maximal patterns. A basis in this problem will consist of m patterns.

Clearly if we relax the integrality constraint on x, we will get a smaller optimal value. However, the relaxation is a very good approximation. There is no known instance of the cutting stock problem where the optimal value and the linear programming relaxation differ by more than just above 1, and it is suspected (but remains to be shown) that the optimal value and the relaxation can never differ by more than some fixed constant.

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 find an initial basis of columns, and efficiently find a column with negative reduced cost.

We can generate an initial basis of feasible patterns $A = [A_1, \dots, A_m]$, 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 $i \ne j$. 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 integer program:

$$\begin{array}{ll} \max & \displaystyle\sum_{i=1}^{n} y_i a_i \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} s_i a_i \leq W \\ & \qquad a \in \mathbb{N}^n. \end{array}$$

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. We have reduced to problem of finding a negative reduced cost column to enter the basis to solving this new integer programming problem. This technique is called *column generation.* The new integer program is in fact the Knapsack Problem, which can be solved via dynamic programming.

1.2 The Knapsack Problem

In the Knapsack Problem, we have items $1 = 1, \ldots, m$ with size s_i and value y_i, a knapsack of size W, and want to maximize the value of the goods that fit in the knapsack by taking a_i of each type of good. We can efficiently solve the knapsack problem with a dynamic programming algorithm.

We assume s_i and W are integers and each $s_i > 0$.[1] We then define $F_i(v)$ to be the optimum value of the knapsack if the knapsack size is v and we only can take items from $\{1, \ldots, i\}$. Ultimately, we want $F_m(w)$. First, we compute directly $F_1(v)$ for $v = 0, \ldots, W$ directly, as

$$F_1(v) = \begin{cases} \left\lfloor \frac{v}{s_1} \right\rfloor y_1 & y_1 > 0, \\ 0 & \text{otherwise.} \end{cases}$$

For each $i = 1, \ldots, m-1$, for each $v = 0, \ldots, W$, we can compute the remaining terms by the recurrence

$$F_{i+1}(v) = \max_{a_{i+1}=0,\ldots,\left\lfloor \frac{v}{s_{i+1}} \right\rfloor} y_{i+1}a_{i+1} + \underbrace{F_i(\overbrace{v - a_{i+1}s_{i+1}}^{\substack{\text{Space left over}\\ \text{after taking } a_{i+1}\\ \text{items of type } i+1}})}_{\substack{\text{Optimal value of items } 1,\ldots,i\\ \text{fitting in the remaining space}}}$$

As we have mW entries and we must maximize over at most w terms in computing each entry (if for example, every $s_i = 1$), the algorithm will run in $\mathcal{O}(mW^2)$. By modifying the algorithm to store items that were selected, we can obtain the optimal a as well, or alternatively, we can reconstruct a directly from the dynamic programming table F.

[1]More generally, we need only assume s_i and W are rationals where $s_i > 0$, as we can multiply our constraints by a common denominator to make everything integer valued. We cannot in general extend this algorithm for irrational s_i or w.

2 Dantzig-Wolfe Decompositions

Consider a linear program of the form

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{m} c_i^T x_i & \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} A_{0i} x_i = b_0 & \\ & A_{ii} x_i = b_i & \forall i = 1, \ldots, m \\ & x_i \geq 0 & \forall i = 1, \ldots, m \end{array}$$

where $A_{ij} \in \mathbb{R}^{m_i \times n_j}$, $c_j, x_j \in \mathbb{R}^{n_j}$, and $b_i \in \mathbb{R}^{m_i}$. The first type of constraint is called a linking constraint. Such an LP is quite reasonable. For example, consider a bakery chain minimizing its costs. Each bakery in the chain has its own requirements for production, and there are global constraints on shared resources, perhaps flour. We want to minimize the costs of the entire chain.

In such linear programs, we assume that the local optimization problems

$$\begin{array}{ll} \min & \hat{c}^T x_j \\ \text{subject to} & A_{jj} x_j = b_j \\ & x_j \geq 0 \end{array}$$

can be solved easily for any cost function $\hat{c}$, and use this to solve the global optimization problem.