Mathematical Programming **Lecture 16**
OR 630 Fall 2006 **October 19, 2006**
Scribe: Collin Chan

In this lecture, we consider the 1-dimensional cutting-stock problem. Column generation and dynamic programming are discussed while we solve the problem.

1 The One-Dimensional Cutting Stock Problem

1.1 Some recap from last class / one way of formulating the problem

Consider the problem of minimizing the number of rolls of paper (each with length W) needed to produce b_k rolls with length s_k for $k = 1, 2, ..., m$ where $0 \leq s_k \leq W$ for each k. A clear upper-bound for this problem is $\sum_{k=1}^{m} b_k = N$ (i.e., using one complete roll of paper to produce each one of the rolls needed).

Using $y_i \in \{0, 1\}$, $i = 1, 2, ..., N$ to denote whether or not the i-th roll is used in a solution and $x_{ij} \in \{0, 1\}$, $j = 1, 2, ..., N$ to denote whether or not the j-th desired roll (with length $w_j \in \{s_k : k = 1, ..., m\}$) is obtained from the i-th complete roll, we may formulate the integer programming problem as:

$$\begin{array}{ll} \min & \sum_{i=1}^{N} y_i \\ \text{s.t.} & (1)\text{: } \sum_{j=1}^{N} x_{ij} w_j \leq W \text{ for } i = 1, \ldots, N \text{ (maximum available per roll)} \\ & (2)\text{: } \sum_{i=1}^{N} x_{ij} = 1 \text{ for } j = 1, \ldots, N \text{ (all demands must be satisfied)} \\ & (3)\text{: } 0 \leq x_{ij} \leq y_i \ \forall\, i, j \text{ (must cut from a complete roll which is used)} \\ & (4)\text{: } x_{ij}, y_i \text{ integers.} \end{array}$$

If we drop the "integer-valued" requirement in (4), we get an LP relaxation of the original problem which we can solve to get a lower bound for the IP. Unfortunately, the lower bound obtained is almost useless. The existence of the feasible solution $x_{ij} = y_i = 1/N \ \forall i, j$ implies the optimal solution we get will always be smaller than or equal to 1.

We can improve this somewhat by changing (1) to (1'): $\sum_{j=1}^{N} x_{ij} w_j \leq W y_i$ for all i (where the RHS represents the fraction of each complete roll we actually use). However, even this formulation is not great. One can show that the optimal value of the LP relaxation is merely $\sum_{k=1}^{m} b_k s_k / W$ (a fairly natural lower bound which one could compute without solving an LP!). Heuristically, one can make an improvement by starting with a good solution (which possibly does not make use of all N rolls).

Of course, the quality of the bound produced by the LP relaxation is not the only factor in how effective a formulation is within a branch-and-bound code. Another factor is the extent to which solutions are represented multiple times, since then bad fractional solutions have to be ruled out multiple times within the search. The above formulation is particularly bad in this respect.

1.2 An alternate formulation

We switch our attention to one complete roll with length W. We use a column vector A_j to represent a feasible cutting pattern p_j. The i-th component of A_j (a_{ij}) corresponds to the number of pieces of length s_i cut in one roll of configuration p_j. For p_j to be a feasible cutting pattern, the elements of A_j must all be non-negative integers and the sum of them must be $\leq W$ ($\sum_{i=1}^{m} a_{ij} \leq W$).
Taking $W = 10$, $s_1 = 5$, $s_2 = 3$ and $s_3 = 2$ as an example, it is easy to verify that the set of all feasible cutting patterns (expressed in rows) is
$\{(2,0,0),(1,1,1),(0,3,0),(1,0,2),(0,0,5),(0,1,3),(0,2,2),(1,0,2)\} = \mathbf{A}$ union with all possible subsets of each of the elements of $\mathbf{A}$. (Aside: Note that the set $\mathbf{A}$ consists only of cutting patterns that could not fit any more rolls with one of the required lengths. We call these cutting patterns "maximal". Any subset of an element of $\mathbf{A}$ will however have enough space to fit one or more rolls with one of the required lengths.) Let n be the total number of distinct feasible cutting patterns (that is, the number of vectors A_j satisfying the constraints) and let x_j denote the number of rolls cut according to the j-th pattern. We may now formulate the 1-dimensional cutting stock problem as the following IP:

$$\begin{array}{ll} \min & \sum_{j=1}^{n} x_j \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \text{ for } i = 1, \ldots, m \text{ (demand constraint)} \\ & x_j \geq 0 \text{ integer-valued} \end{array}$$

Solving the LP-relaxation will give us another lower bound for the IP. Recall that since there are only m constraints, there are at most m non-zero variables. Given the LP-solution, we may round up decision variables with fractional values to the next integer and conclude that **IP-optimal $\leq$ LP-optimal + m**. (Note that an IP which has $\geq$ replaced by $=$ in the constraints above but otherwise remains the same is also valid. But the $\geq$ format meshes well with the situation where we only use "maximal" cutting patterns.)

1.3 Column Generation

With the possibility that n is very large, it can be very difficult to solve the LP directly. This leads us to the idea of column generation. We restrict ourselves to a few patterns and generate new ones as needed. We hope that the number of iterations depends on m but not n here.
To run revised simplex on this LP using column generation, we need to be able to:
(1) find an initial basis, and
(2) algorithmically determine if all non-basic columns have reduced cost ≥ 0; and if not, find a column with negative reduced cost.

It is not difficult to find an initial basic feasible solution for this problem. Consider the set of column vectors $\{A_j\}_{1 \leq j \leq m}$ where $a_{ij} = \lfloor W/s_j \rfloor$ if $i = j$ and 0 otherwise. Using this as a basis, we get the basic feasible solution $x_j = b_j$ for $j = 1, ..., m$ and 0 otherwise.

We now consider how to generate new patterns (i.e., how to find a non-basic column whose reduced cost is negative.) Given any basic feasible solution, we can obtain the corresponding dual solution easily by solving $\overline{y} A_B = c_B$. The reduced cost associated with an arbitrary cutting pattern p_j is equal to $c_j - \overline{y} A_j = 1 - \overline{y} A_j$. Consider the auxiliary optimization problem:

$$\begin{array}{ll} \max & \sum_{i=1}^{m} \overline{y}_i a_i \\ \text{s.t.} & \sum_{i=1}^{m} s_i a_i \leq W \\ & a_i \geq 0, \ \text{integer-valued } \forall i \end{array}$$

This problem is referred to as the integer knapsack problem. With the discussion above, one can see that any feasible solution for the auxiliary problem corresponds to a feasible cutting pattern in the cutting-stock problem.

Lemma 1 *There exists a negative reduced cost column (in the cutting-stock problem) if and only if the optimal value to this auxiliary IP is > 1. The optimal solution gives a column whose associated reduced cost is < 0.*

Proof: As mentioned, any feasible solution to the integer knapsack problem corresponds to a feasible cutting pattern in the cutting-stock problem. If the optimal solution for the knapsack problem is > 1 , its reduced cost would be equal to $1 - \sum_{i=1}^{m} \overline{y}_i a_i < 0$. Hence we may enter this column into the basis (in the cutting stock problem). If the integer knapsack problem has an optimal solution ≤ 1, all the reduced costs are non-negative and we may conclude that we have an optimal solution for the cutting-stock problem.

1.4 Dynamic Programming

The last thing we need to consider is how to solve the integer knapsack problem. One approach is via dynamic programming. Because the units used in length measurements can be scaled, we may assume that s_i, $i = 1, ..., m$ and W are integers without loss of generality.

Let $F_i(v)$ denote the maximum value obtained from the integer knapsack problem where W is replaced by v and m is replaced by i above. The ultimate goal here is to find out what $F_m(W)$ is. We first try to write an expression for $F_i(v)$. Given a roll of paper with length v, we start by cutting rolls with length s_i from it. The number of rolls with length s_i one could get is any integer a_i between 0 and $\lfloor v/s_i \rfloor$. Fixing a_i, we then try to cut rolls with lengths s_k, $k = 1, ..., i-1$ optimally from the $v - a_i s_i$ which remains. Therefore, we can express $F_i(v)$ using the following equation:

$$F_i(v) \quad = \quad \max_{a_i = 0, \ldots, \lfloor v/s_i \rfloor} (0, \overline{y}_i a_i + F_{i-1}(v - s_i a_i))$$

This gives a recursive relationship that links F_i with F_{i-1}. Because it is very easy to compute $F_1(v)$ ($F_1(v) = \overline{y}_i \lfloor v/s_i \rfloor$ if $\overline{y}_i \geq 0$ and 0 otherwise), we have a way of computing $F_m(W)$. In each iteration i, compute F_i for all v. These values depend only on the input data and the values of F_{i-1} already computed. By storing the maximization of each entry, we may backwards construct the optimal solution we need (the cutting pattern with the most negative reduced cost).