**ORIE 6300 Mathematical Programming I** October 21, 2014

# Lecture 16

*Lecturer: David P. Williamson* *Scribe: Shih-Hao, Tseng*

## 1 The Cutting Stock Problem

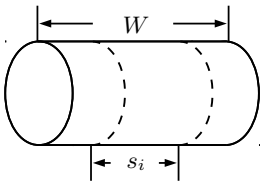


Figure 1: Raw

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 $i = 1, \ldots, m$. 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).

How can we use LP to solve this problem? We start out by formulating this as an *integer* program. Here is one possible formulation (which is not very good, as we will see). Let $w_j$ denote the length of the $j$th final to be produced (i.e. it is $s_i$ for some $i$). We set up decision variables $y_i$ and $x_{ij}$ in which we use $y_i \in \{0, 1\}$, $i = 1, 2, ..., N$ 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 is obtained by cutting that length from the $i$-th complete raw. Thus we may formulate the integer programming problem as:

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

It turns out this is a bad integer programming formulation, but to understand why, we need to know a little about how integer programs are solved using linear programs as a subroutine. First, we solve the LP relaxation of the IP: Relax $x_{ij} \in \{0, 1\} \rightarrow 0 \leq x_{ij} \leq 1, y_i \in \{0, 1\} \rightarrow 0 \leq y_i \leq 1$. The value of the LP relaxation is at most the value of the IP, since the optimal IP solution is feasible for the LP relaxation. If the optimal LP solution is integral (i.e. it has $x_{ij} \in \{0, 1\}, y_i \in \{0, 1\}$), then the optimal LP solution is the optimal IP solution. If the LP solution is not integer, then we can use a branch-and-bound algorithm. In branch-and-bound, we take some fraction variable (say $y_i$), and solve two subproblems, one in which $y_i = 0$ and the other in which $y_i = 1$. Clearly, the cheaper of the two subproblems should be the solution to our overall problem. We solve the subproblem as we started out solving the general problem; i.e., solving a linear programming relaxation, and hoping to obtain an integral solution, and "branching" on a fractional variable if needed.

For this process to work well, it desirable to have at least two things:

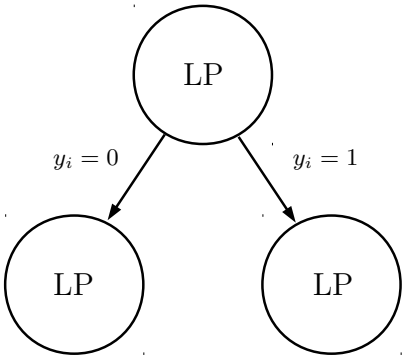


Figure 2: Branch-and-bound

1. The value of the LP relaxation is close to the IP optimal solution, so that we don't need to enumerate many subproblems before we find an integer solution.

2. The two possible branches partition the space of solutions, so we aren't considering the same possibilities under both branches (this will slow things down otherwise).

Neither is true of the above IP formulation of the problem. First, a trivial solution to the LP relaxation is $x_{ij} = y_i = 1/N \ \forall i, j$, which has LP value of 1. So no matter what the true IP solution is, our LP solution can be quite far away from it. Second, there are lots of different ways of representing the same integer solution.

## 1.1 Another Formulation

We switch our attention to one raw of length $W$. We use a column vector $A_j$ to represent a feasible cutting *pattern* or *configuration* $p_j$. 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 at most $W$ ($\sum_{i=1}^{m} a_{ij} \leq W$). Let $x_j$ = number of rows cut in pattern/configuration $j$. For example, $W = 10$, $s_1 = 5$, $s_2 = 3$, $s_3 = 2$ can cut raws into finals in the following ways:

$$\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} x_2 + \ldots \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} x_7 \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{1}$$

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 $j$.

We relax the IP by LP relaxation. Replace the integer constraint by $x \geq 0$. Is that a good relaxation of IP?

1. How many $x_j > 0$? At most $m$ for any basic feasible solution.

2. If we round up every non-zero LP variable to the nearest integer, this integer solution is feasible and has value at most $m$ more than LP solution.

## 1.2 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:

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

where $a_i$ is the number of finals of size $i$ to cut for pattern $a$, $s_i$ is the size of the $i$th 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, and we discuss this next time.