An Integer Programming Solver for The Wicker Rod Optimization Problem

May 4, 2022

1 Requirements

To manufacture a rattan box, 4 pieces of length 35, 4 pieces of length 30 and 4 pieces of length 20 are needed. For that, rods of length 330 are provided. The goal of the producer is to find a batch size and the "recipe" that will allow the production of that batch while minimizing the absolute wicker waste.

2 Formalization of the problem

We start with a little bit of notation:

- We will call l_1, l_2, l_3 to the three lengths of the rods, sorted descending.
- Let l be the total length of the provided rods.
- Each "recipe" will amount giving to describing the ways of cutting the provided rods of length l; formally $p = (l^1 \ldots, l^k)$ where $l^1 + \cdots + l^k = l$.

In order to formalize the problem, we define the notion of *optimal partition* as a partition $(l^1, \ldots, l^r)$ where $l^1 \geq \cdots \geq l^{r-1} > l^r$ y $l^1, \ldots, l^{r-1} \in \{l_1, l_2, l_3\}$. The sorting condition is non-essential and it only simplifies the later development (and the clarify of presentation for the manufacturer), the important thing is that all pieces corresponds to lengths in l_1, l_2, l_3 and that there will be only one clipping piece, which won't be possible to use.

Let P be the set of all partitions, and let $\hat{P}$ the set of just the optimal ones. It will become clear that it's possible to compute easily the set $\hat{P}$; so we will

be able to write $\hat{P} = (p^1, \ldots, p^k)$, where k is the total quantity of optimal partitions for the parameters (l, l_1, l_2, l_3).

For each partition p, we consider for integer quantities:

1. $N_1(p), N_2(p)$ y $N_3(p)$ that amount to the multiplicities of l_1, l_2 and l_3 respectively in the partition p.
2. $C(p) = l^r$ si $p = (l^1, \ldots, l^r)$. That is, $C(p)$ is the clipping length of partition p.

We will finally introduce multiplicities m_1, m_2, m_3 that correspond to the quantities needed for manufacturing a production unit. In the requirement for the rattan box, we trivially have $m_1 = m_2 = m_3$, but this seems just like an accidental fact and also there are no great difficulties in taking $m_1, m_2, m_3 \in \mathbb{N}$ as wished.

All that said, we state the optimization problem:

Find $(n_1, .., n_k) \in \mathbb{N}^k$ that minimizes $\sum_{i=1}^{n} C(p_i) \times n_i$, sujebject to the constrains

1. $m_{j_1}(\sum_{i=1}^{k} N_{j_0}(p^i) \times n_i) = m_{j_0}(\sum_{i=1}^{k} N_{j_1}(p^i)) \times n_i) \quad \forall j_0, j_1 \in \{1, 2, 3\}$
2. $n_1 + \cdots + n_k < n_{bound}$

being n_{bound} the maximum admissible amount for the batch.

Notice that the number of rods returned is $n_0 = n_1 + \cdots + n_k$. But the batch size b must also accomplish $bm_0 \mid (\sum_{i=1}^{k} N_j(p^i) \times n_i)$; then we will set

$$m_1' = \text{lcm}(m_1, N_1)$$

- where N_j is a notation for $\sum_{i=1}^{k} N_j(p^i) \times n_i$- , as the quantity we need to manufacture for length l_1 and then set the batch size to $\frac{m_1'}{m_1}$. We have:

- $bm_1 = m_1' = \frac{m_1}{\text{gcd}(m_1, N_1)} N_1$ by property of lcm.
- $bm_2 = m_2 \frac{m_1'}{m_1} = m_2 \frac{N_1}{\text{gcd}(m_1, N_1)} = \frac{m_2 N_1}{\text{gcd}(m_1, N_1)} = \frac{m_1 N_2}{\text{gcd}(m_1, N_1)} = \frac{m_1}{\text{gcd}(m_1, N_1)} N_2$
- $bm_3 = m_3 \frac{m_1'}{m_1} = m_3 \frac{N_1}{\text{gcd}(m_1, N_1)} = \frac{m_3 N_1}{\text{gcd}(m_1, N_1)} = \frac{m_1 N_3}{\text{gcd}(m_1, N_1)} = \frac{m_1}{\text{gcd}(m_1, N_1)} N_3$

From this we can see that the vector $(\frac{m_1}{\text{gcd}(m_1, N_1)} n_1, \ldots, \frac{m_1}{\text{gcd}(m_1, N_1)} n_k)$ accomplishes the manufacturing objective.

3 Tackling the problem using IP

What we see next is that, if fixing $n < n_{bound}$, we can find an equivalent IP problem for the original problem.

We start by defining the matrix $A \in \mathbb{N}^{2\times k}$ as:

$$\begin{pmatrix} m_2N_1(p_1) - m_1N_2(p_1) & \cdots & m_2N_1(p_k) - m_1N_2(p_k) \\ m_1N_2(p_1) - m_2N_1(p_1) & \cdots & m_1N_2(p_k) - m_2N_1(p_k) \\ m_3N_2(p_1) - m_2N_3(p_1) & \cdots & m_3N_2(p_k) - m_2N_3(p_k) \\ m_2N_3(p_1) - m_3N_2(p_1) & \cdots & m_2N_3(p_k) - m_3N_2(p_k) \\ 1 & \cdots & 1 \end{pmatrix} \tag{1}$$

y el vector $\vec{c} \in \mathbb{N}^k$

$$\begin{pmatrix} C(p_1) \\ \vdots \\ C(p_k) \end{pmatrix} \tag{2}$$

We can state the following problem as a classic IP problem:

Find $\vec{n} \in \mathbb{Z}^k$ that minimizes $\vec{c} \cdot \vec{n}$, subject ot the constrains:

1. $A\vec{n} \leq (0, 0, n_{bound})^t$
2. $\vec{n} \geq \vec{0}$

It's clear that:

1. Restriction 2. makes $\vec{n} \in \mathbb{N}^k$
2. It is clear that

$$\begin{array}{lll} (A\vec{n})_0 \wedge (A\vec{n})_1 \leq 0 & \iff & \sum_{i=1}^{k}(m_2N_1(p_i) - m_1N_2(p_i)) \times n_i = 0 \\ & \iff & m_2(\sum_{i=1}^{k} N_1(p_i) \times n_i) = m_1(\sum_{i=1}^{k} N_2(p_i) \times n_i) \end{array}$$

Similarly, we get

$$(A\vec{n})_2 \wedge (A\vec{n})_3 \iff m_2(\sum_{i=1}^{k} N_1(p_i) \times n_i) = m_1(\sum_{i=1}^{k} N_2(p_i) \times n_i).$$

Finally, we get the restriction 1. of the original problem on $j_0 = 1, j_1 = 3$ for free:

- By multiplying the first equality by m_3 we get
$$m_3 m_2 (\sum_{i=1}^{k} N_1(p_i) \times n_i) = m_3 m_1 (\sum_{i=1}^{k} N_2(p_i) \times n_i)$$
- By multiplying the second equality by m_1 we get
$$m_1 m_3 (\sum_{i=1}^{k} N_2(p_i) \times n_i) = m_1 m_2 (\sum_{i=1}^{k} N_3(p_i) \times n_i)$$
- By transitivity and cancelling out m_2, we get
$$m_3 (\sum_{i=1}^{k} N_1(p_i) \times n_i) = m_1 (\sum_{i=1}^{k} N_3(p_i) \times n_i)$$
which implies restriction 1. of the original problem.

This completes the equivalence for condition 1.

3. It's clear that $(A\vec{n})_2 \leq n_{bound}$ means exactly $n_1 + \cdots + n_k < n_{bound}$, which is equivalent to restriction 1. of the original problem.

We can conclude that the problem we just presented is therefore equivalent to the original problem that we want to solve.

4 Implementation

The implementation of the IP problem stated above is done with the CPLEX software.
The only caveats that we can mention from the implementation process are:

- Generating the optimal partitions is done following a left-to-right "greedy" algorithm, which will return them in the standard lexicographic order.
- As the trivial solution ($\vec{n} = 0$) is always an optimal solution (and also the one chosen by the solver), we needed to add a final row $(-1 \cdots -1)$ with -1 as the corresponding restriction; which will enforce the solution not being trivial.

The implementation can be accessed on Github.

5 Further research

The fact that we are using the set $\hat{P}$ instead of the plain P is a relaxation for the problem we actually stated informally in section 1, since any solution using a non-optimal partition p can be done at least as effectively by replacing it with $\hat{p}$ that has at least the same multiplicities at less cost (sorting the lengths in p and then filling the original clipping respecting the optimality).

But, at first sight, the equality of condition 1. in the original formulation from section 2 is not a relaxation, since we could, in theory, have a solution that wastes less wicker by affording some pieces of lengths in $\{l_1, l_2, l_3\}$ to be wasted, along with the clippings.

Regarding this last point, it's probably not difficult to found a reasonable bound to the inequalities that serve as valid relaxations, and rewrite the IP problem accordingly using this bound for the values $c_0, \ldots, c_3$ instead of just 0.