Using Total Unimodularity to Solve a Warehouse Problem

Zachary Sorenson

November 30, 2019

Abstract

With a daily increase in the usage of online retailers like Amazon, the operations of warehouses are becoming more and more valuable. One way to create an efficient warehouse is to effectively place items that have yet to be shipped. The primary goal of this project is to develop a mathematical model that could create a scheme of inventory layout in a warehouse.

From a mathematical standpoint, the original model I propose is an integer programming model. In order to solve any integer program of a reasonable size, one would need a significant amount of memory and a significant amount of time. A natural next step then is to relax the developed integer program for a more efficient solution. We do so by using a mathematical concept called "totally unimodularity" to prove that the optimal vertex solutions of a linear programming relaxation are integral.

We can reformulate our model by using two variables with $y_{i,j}$ which is how many small bins in cluster i contain item j and $z_{i,j}$ which is how many large bins in cluster i contain item j. s_i and l_i will store the number of small bins in cluster i and the number of large bins in each cluster i, respectively. b_g will store the number of bins a small or large bins an item must be split in to and q_t will store the number of large bins an item will go in to.

$$\sum_{j=1}^{n} y_{i,j} + p_i = s_i \quad \forall i = 1, .., k$$

$$\sum_{t=1}^{m+n} z_{i,t} + f_i = l_i \quad \forall i = 1, .., k$$

$$\sum_{i=1}^{k} y_{i,g} + \sum_{i=1}^{k} z_{i,g} = b_g \quad \forall g = 1, ..., n$$

$$\sum_{i=1}^{k} z_{i,t} = q_t \quad \forall t = n+1, ..., m+n$$

$$y_{i,j} \geq 0 \quad z_{i,t} \geq 0 \quad p_i, f_i \geq 0 \quad l_i, s_i \in \mathbb{Z} \quad b_g \in \mathbb{Z} \quad q_t \in \mathbb{Z} \quad (i \leq q,\ j \leq n,\ g \leq n,\ t \leq m+n)$$

Let $C_{k,n}$ be all matrices with k rows that contain strings of length n and each row is shifted by n elements. Let $I_{n,k}$ be k consecutive, horizontally placed, $n \times n$ identity matrices. The constraint matrix will look like

$$A = \begin{bmatrix} C_{k,n} & 0 & 0 & I & 0 \\ 0 & C_{k,m+n} & & 0 & I \\ I_{n,k} & I_{n,k} & 0 & 0 & 0 \\ 0 & 0 & I_{m,k} & 0 & 0 \end{bmatrix}$$
$$\underbrace{}_{\text{(I)}} \quad \underbrace{}_{\text{(II)}} \quad \underbrace{}_{\text{(III)}} \quad \underbrace{}_{\text{(IV)}}$$

Integer programs can be difficult to solve. One valuable property of linear programs is that the optimal solution will appear on a vertex. Unfortunately, this property does not apply to integer programs. The reason for this is that optimal solutions for integer programs exist in an integer hull which may not lay on the boundary of any polyhedron that can be defined by linear constraints. This is where the idea of Totally Unimodular Matrices becomes important.

Definition. *(Unimodular Matrix)- A matrix is "Unimodular" if all of its upper left square submatrices have determinants of 1, -1, or 0. A matrix is considered "Totally Unimodular" if every square submatrix of the original matrix has a determinant of 1, -1, or 0.*

We can remove the integrality of our model if we can be certain that the optimal solution of the linear model will be integral. There are two conditions for linear models that will imply an integer optimal solution. The first is that the constraint matrix is totally unimodular. The second is that the right hand side of the constraints will be integral. [1] If we our able to relax our integer model into a linear model we can solve larger versions of our model more quickly.

By definition, the second condition is already met by our model. What we must now show is that our constraint matrix is totally unimodular. Let's start by stating the following proposition.

Proposition 1. *A matrix is totally unimodular if no more than 2 nonzeros are in each column, and if the rows can be partitioned into two sets such that if a column has two entries of the same sign, their rows are in different sets of the partition. [2]*

Let's look at our matrix A, column by column, to show that the two conditions of Proposition 1 are met. As can be seen in the definition of the matrix, A contains $2k + m + n$ rows. Let A_u be the first $2k$ rows of matrix A which contains $C_{k,n}$, $C_{k,m+n}$, and two identity matrices. Let A_l be the last $n + m$ rows of matrix A which contains two $I_{k,n}$ matrices and one $I_{k,m}$ matrix. The first (I) columns of matrix A will have a 1 in the $a_{1,j}$ entry, a 1 in the $a_{2k+i,j}$ entry, and zero everywhere else in the column. Since A_u contains the rows $i \leq 2k$, $a_{1,j}$ will appear in A_u. Since A_l contains the rows $2k + 1 \leq i \leq 4k$, $a_{2k+i,j}$ will appear in A_l. For the next k columns our matrix will have a 1 in the $a_{2,i+k}$ entry and a 1 in the $a_{2k+i,k+i}$ entry. These which will appear in the A_u and A_l matrix respectively. We can continue the process for the first nk columns of our matrix to conclude that there are only two 1's in each column and each of these 1's appear in separate partitions.

For the next (II) columns, matrix A will have a 1 in the $a_{k+1,nk+1}$ entry and a 1 the $a_{2k+i,nk+1}$ entry and will have a zero everywhere else in the column. These will appear in the A_u, A_l partitions respectively. We can continue the same process outlined in the previous paragraph to find that the remaining $k(m+n)-1$ columns will continue to meet the sufficient conditions of Proposition 1.

For the remaining (III) and (IV) columns of matrix A is comprised entirely of identity, or zero, sub-matrices which will have only one entry that has a value of 1 in each column. Since there is only one non-zero entry in each column, which has a value of 1, the columns of A_u and A_l could only contain a single 1. This shows that the sufficient conditions for Proposition 1, will be met for these columns.

Since we have shown that the sufficient conditions for proposition hold for every column of A, we can conclude that A must be totally unimodular.

A AMPL Code

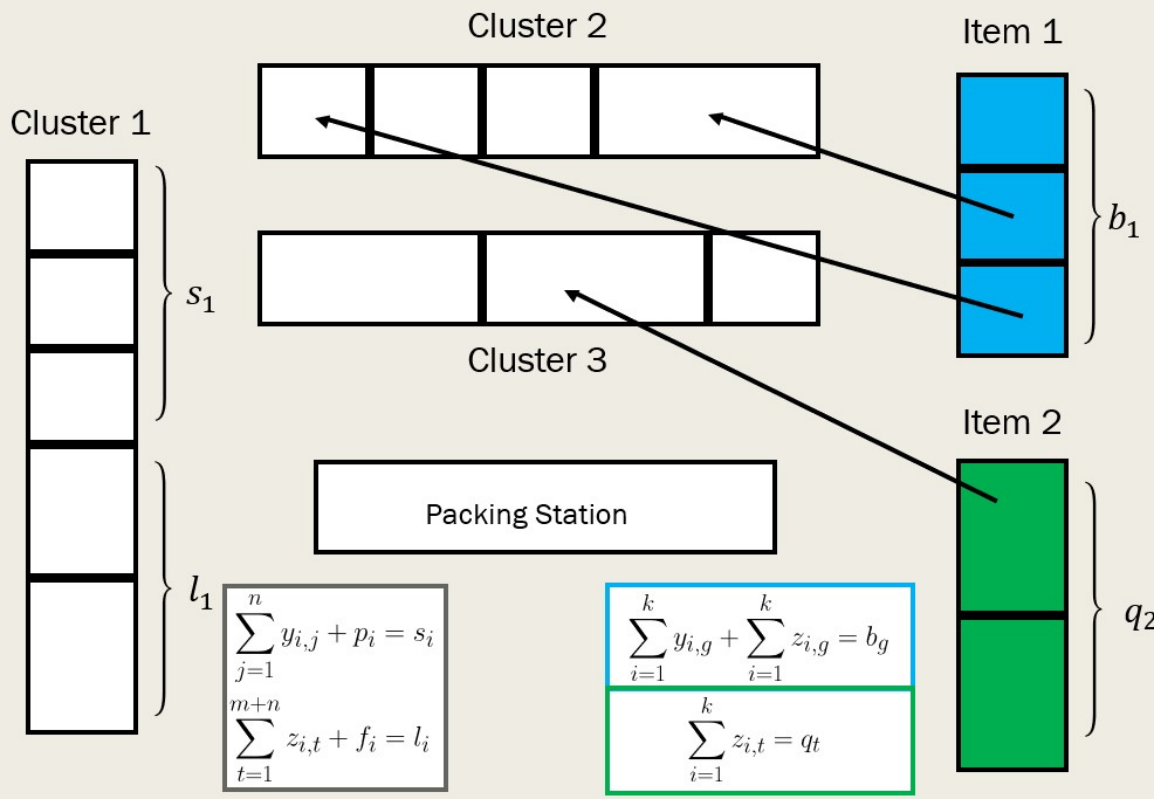
ware.mod;

```
param m >= 0;
param n >= 0;
param k >= 0;
set Clustsma; #stores the size of the small clusters
set Clustla; #stores the size of the large clusters
set Binsizma; #stores the amount of bins a small item fits into
set Binsizla; #stores the amount of bins a large item fits into
param mn = m + n;
param Clustsizema{Clustsma} >= 0;
param Clustsizela{Clustla} >= 0;
param Binsla{Clustla} >= 0;
param Binssma{Clustsma} >= 0;
var y {1..k, 1..n} >= 0, <=1;
var z {1..k, 1..mn} >= 0, <=1;
var s1 {1..k}; #slack variable
var s2 {1..k}; #slack variable

minimize Total_distance: #This is arbitrarily defined
    sum {j in 1..n, i in 1..k} (y[i,j])+sum {j in 1..mn, i in 1..k} (z[i,j]);

subject to SmallbinClust{i in 1..k}: #constraint 1
sum {j in 1..n} (y[i,j])+s1[i]=Clustsizema[i];

subject to LargebinClust{i in 1..k}: #constraint 2
sum {j in 1..mn} (z[i,j])+s2[i]=Clustsizela[i];

subject to Largebins {j in 1..n}: #constraint 3
sum {i in 1..k} (z[i,j]+y[i,j])=Binssma[j];

subject to smallbins {j in n+1..mn}: #constraint 4
sum {i in 1..k} (z[i,j])=Binsla[j];
```

ware.dat;

```
data;

param n := 3 ;

param: Binsizma: Binssma :=
        1           1
        2           2
        3           1;

param m := 1 ;

param: Binsizla: Binsla := #there needs to be n+m options here
        1           2
        2           2
        3           1
        4           5;

param k := 5; #number of clusters

param: Clustsma: Clustsizema :=
          1             1
          2             2
          3             0
          4             1
          5             0;


param: Clustla: Clustsizela := #must be the same length as Clustsma
          1             2
          2             0
          3             1
          4             1
          5             1;
```

B Warehouse Diagram



C Alternate Models

Initial Model:

For the first attempt at a model, I took $s_1, s_2, ..., s_k$ be the upper bounds of the amount of items that can be put in a cluster, and let $y_{i,j}$ be a decision variable to determine whether or not we pick an item to put in our cluster. In our model, we can use $v_1, v_2, ..., v_k$ to store the volume of each cluster and $p_1, p_2, ..., p_n$ to store the volume of each individual item. The model that I propose is

$$\sum_{j=1}^{n} y_{i,j} \leq s_i \quad \forall i = 1, .., k$$

$$\sum_{i=1}^{k} y_{i,j} = 1 \quad \forall j = 1, .., n$$

$$\sum_{j=1}^{n} p_j y_{i,j} \leq v_i \quad \forall i = 1, .., k$$

$$y_{i,j} \in \{0, 1\} \quad \forall i = 1, .., k \quad \forall j = 1, .., n$$

The first constraint is used to provide an upper bound for the clusters.

The second constraint is used to make every item goes into some cluster.

The third constraint is use to make sure the that the total volume of the items will be less that the total amount of space in a cluster.

There are some additional ideas to consider for our model. If we want to have a constraint that only allows items that are small enough to fit into the correct size of individual bins then we could introduce $m_{i,1}, ..., m_{i,t}$ to store the sizes of each bin t in cluster i. We would then need add the constraint

$$p_j y_{i,j} \leq m_{i,l} \quad \forall i = 1, .., k \quad \forall j = 1, .., n \quad \forall l = 1, .., t$$

to our model.

Comment: This model was a good starting point, but there were a couple of issues. One of these issues was that integrality was not guaranteed for the solution. Additionally, this model didn't account for the amount of bins in a cluster.

3 Index Model:

For another attempt at a model we let $y_{i,j,k}$ be a decision variable to determine whether or not we pick an item j, put it into k bins in any of our clusters i. We let $s_1, s_2, ..., s_r$ be the upper bounds of the amount of items that can be put in a cluster. $b_1, ..., b_n$ will store the amount of bins any item goes into. In order to store which type of item and which type of bin we have, we can use a variable $t_1, ..., t_n$(1 for big, -1 for small) and $m_1, ..., m_n$(1 for big, -1 for small) respectively. Finally, to ensure that there is equality in the first and last constraint, we must include the addition of a slack variables $p_i, d_{i,j,k}$.

$$\sum_{j=1}^{n} \sum_{k=1}^{b_j} y_{i,j,k} \leq s_i \quad \forall i = 1, .., r$$

$$\sum_{i=1}^{r} \sum_{k=1}^{b_j} y_{i,j,k} = b_j \quad \forall j = 1, ...n$$

$$y_{i,j,k} = \frac{t_j}{m_l} + (-1)\frac{t_j}{m_l} \cdot d_{i,j,k} \quad \forall i = 1, .., r \quad \forall j = 1, ..., n \quad \forall k = 1, .., b_j \quad \forall l = 1, ..., s_i$$

$$y_{i,j,k} \in \{0, 1\} \quad d_{i,j,k} \in \{0, 1\}$$

The first constraint is used to provide an upper bound for the clusters.

The second constraint is used to make every item go into the correct amount of bins.

The last constraint is used to make sure that the correct size of item goes to the correct size of bins. The right side will become positive when the sizes both $t_i$andr m_j are big (1 and 1) and when t_i or m_j are small(-1 and -1) and will be zero otherwise.

Comment: This model made it difficult to prove that the constraint matrix was TU. One thing that I was particularly proud of with this model is that the last constraint functions as a "switch" that insured big items went into big bins and small items went into small bins. This constraint was eventually replaced by one that allowed for small and large items to be placed into large bins.

References

[1] Cunningham, William H., and Geelen, James F.. "Integral Solutions of Linear Complementarity Problems." Mathematics of Operations Research, vol. 23, no. 1, 1998, pp. 61–68.

[2] Chandrasekaran, R. “Total Unimodularity of Matrices.” SIAM Journal on Applied Mathematics, vol. 17, no. 6, 1969, pp. 1032–1034.