Brief Description of the DP Algorithm Design

Objective

To estimate the maximum expected profit in T days given the current storage condition.

Mathematical Model

The entire storage can be divide into m categories and the items belonging to the ith category are expiring in i days where $i \in [0, m-i]$. Note the maximum number of items within each category is limited to k. We store the number of items in all the categories in a m dimensional vector where the first element represents the number of items to be expired today and the last element represents the number of items to be expired in $m-1$ days. We denote the vector as X and X_t is the storage status of the t-th day.

Everyday in the evening, the shop manager has to clean up the items that has been expired on that day and each item has a disposal cost θ. Next he shall decide the depletion amount. For each item depleted, an amount of salvage denoted as s will be recovered and counted into benefits. And after the depletion, all the items remained in the storage will be charged a holding cost each which is denoted as h. And in the morning of the following day, the manager will decide the total amount of items to order, the order cost is c per item. During the opening hours, those items which are going to be expired will be sold based on the possion distribution, and the selling price is r. Therefore the net income for the entire period can be expressed as

$$\begin{aligned} Profit = \; & + \; numDeplete \cdot s \\ & - \; totalNum \cdot h \\ & - \; numOrder \cdot c \\ & + \; min(\mathbf{D},\; totalNum + numOrder) \cdot r \\ & - \; X_0 \cdot \theta \end{aligned}$$

where

- $numDeplete$ is the number of items been depleted on the night before
- $totalNum$ is the number of items remained in the storage after depletion but before the next ordering
- D is the random variable to denote the number of items sold in a day. And it follows the Possion distribution.

So in order to find out the maximum expected value, we have the following recursion relation:

$$V_t(X_t) = max_{z \in Z, q \in Q}(\mathbf{E}[Profit(X_t, z, q, \mathbf{D})] + \alpha \mathbf{E}[V_{t+1}(S(X_t, z, q), z, q, \mathbf{D})])$$

where

- z represents the number of depletion with range Z, and q is the number of ordering with the limit Q
- S is the function to evaluate X_{t+1} given X_t, z, q

Therefore V_t is attainable once given V_{t+1}.

We shall start from the table V_T, in which all the entries equal to zero, to recursively update the table V_i $(i = T-1, T-2, \dots)$ and go all the way down to $i = 0$, which is the table of today.

Approaches

The problem itself has some fine features leading to a simpler solution.

- **Inherited Relation of Depletion**
  All the state values within the table shall be organized in a tree, where the optimal depletion number of a child node has some relation with its parent node.
  For example, for a problem with $m = 3,\ k = 2$, the following structure can be obtained.

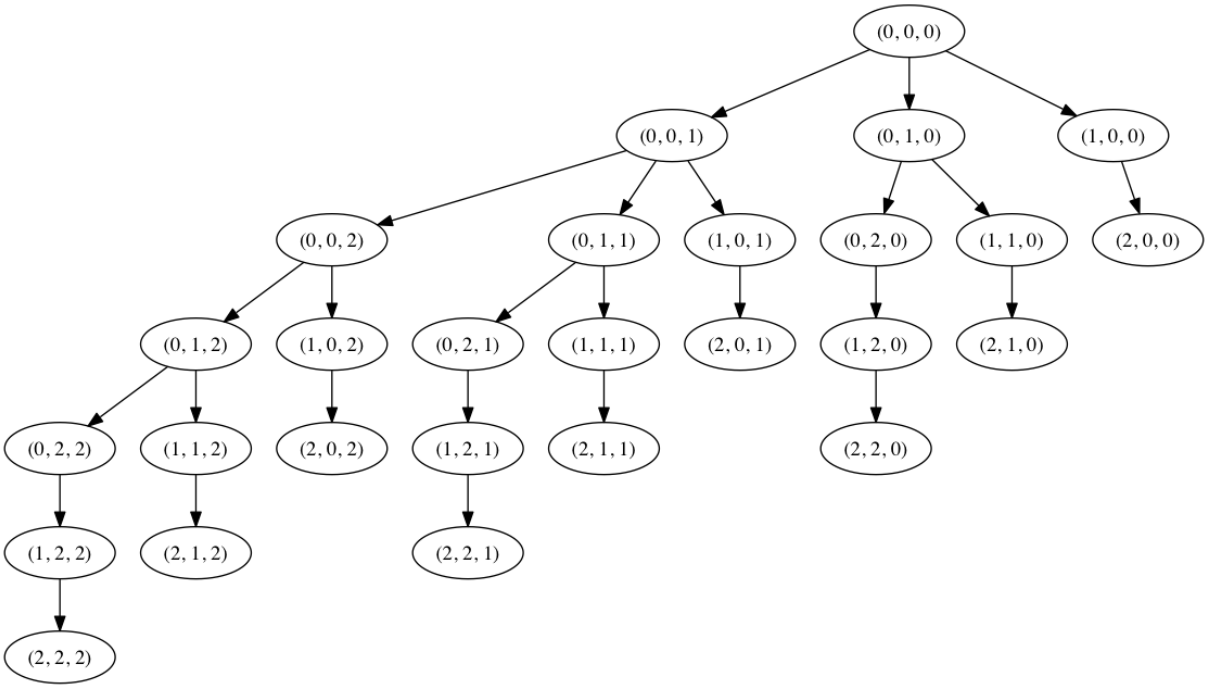

Figure 1: The tree structure for $m = 3$, $k = 2$

- **Limited Demands per day**
  There is a upper limit for the number of customers that may visit the shop everyday. And maximum number should be rather small (hopefully less than a hundred). So instead of launching the kernels for thousands of times to get better estimation, we may simply calculate the result for every demand within the range and adjust the result by the weights given by the Possion distribution.

Based on the above analysis of the problem, a parallel process to deal with the problem can be described as follows. Take the problem discussed above as the example.

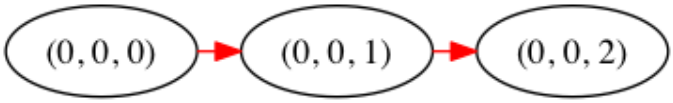

Figure 2: step 1

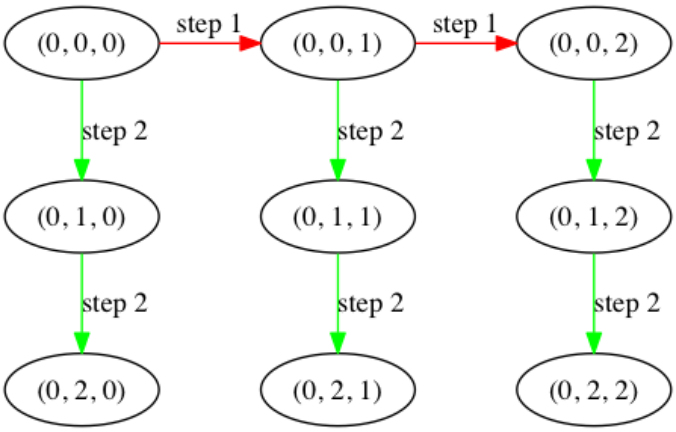

Figure 3: step 2

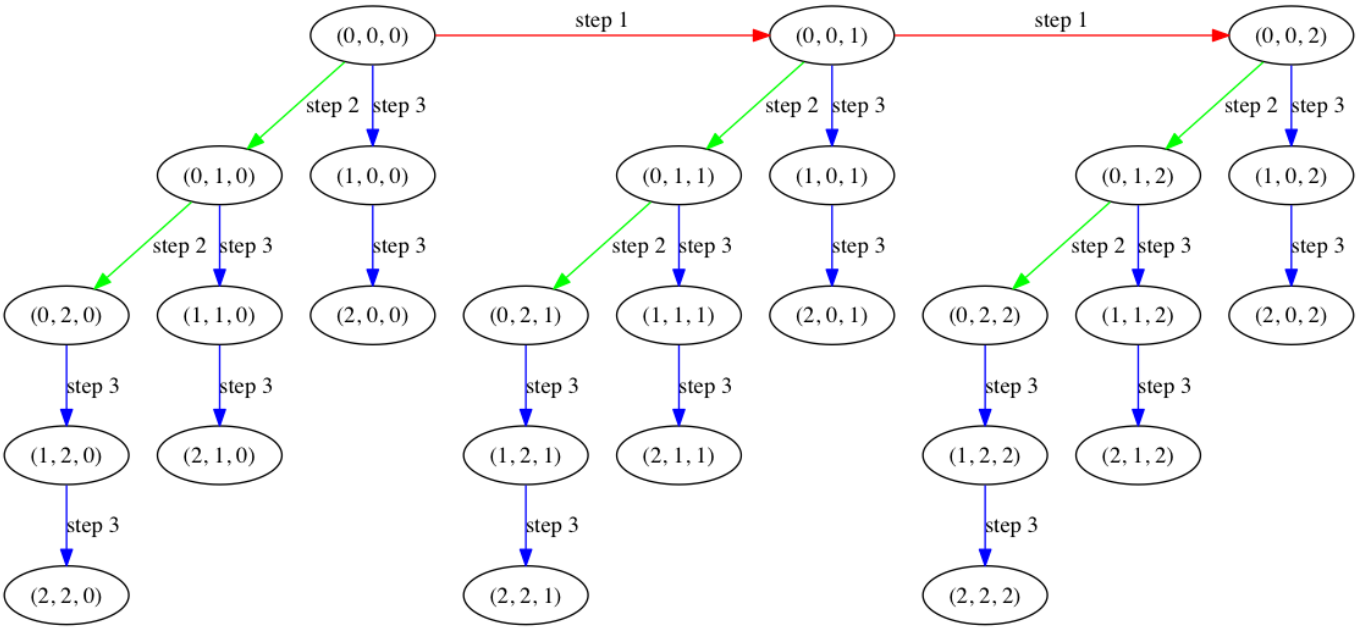

Figure 4: step 3

So in total m iterations are needed and within each iteration k^{i-1} threads can be ran in parallel where i is the index of the iteration.

Algorithm Implementation

There are three tables kept in the GPU memory all the time, the V_{t+1}, the V_t, and a table holding the all the depletion amount for each state of V_{t+1}. And what we are going to do here is to update the table of V_t based on V_{t+1} and the depletion information.

As has been discussed above, the V_t table will be updated in m iterations and within each iteration, k_{i-1} threads will be in use