Approximation Algorithms for Stochastic Inventory Control Models

Hao Yuan Feng Wei Blake Miller

Github:github.com/blakeapm/stochastic-inventory

I. Introduction

Proper inventory, stocking, and purchasing policy are paramount to the success of supply, distribution, and retail businesses. The challenges of minimizing costs in an environment that is dependent upon many stochastic processes presents businesses with a unique challenge: ordering the right amount of material at the right time. This problem is important for any company that seeks to minimize cost of holding excess inventory while minimizing backorder costs (unmet demand). Especially in industries where the demand environment is highly dynamic. (i.e. Apple's supply-chain for solid state drives, seasonal products such as rock salt).

The inventory control problem is very difficult to solve because oftentimes demand is unpredictable and cost is hard to predict due to the stochastic nature of demand. It is particularly difficult to develop an approximation algorithm that is appropriate to the risk profile of companies operating under dynamic demand environments. In this paper, we will investigate the performance of a new algorithm[2] for the inventory control problem using C++ for implementation and R for data visualization.

II. The Model

First, let's look at a simplified model with a lead time of $L = 0$. This means the product will arrive immediately upon order placement. We then consider a finite time horizon T:

At each time period $t \in \{1, \cdots, T\}$, the following costs are incurred:

$$\begin{aligned} & L_t(x_t, d_t, q_t) \\ := \; & c_t q_t + h_t(x_t + q_t - d_t)^+ + p_t(x_t + q_t - d_t)_t^- \end{aligned} \tag{1}$$

where

- c_t: per-unit ordering cost.
- h_t: per-unit holding cost.
- p_t: per-unit backlogging penalty.
- x_t: net inventory (state).
- q_t: ordering quantity (control).
- d_t: demand quantity (we observe d_t after decide q_t).

In Eq. (1), $x_t + q_t - d_t$ is the net inventory at the end of period t. If it is positive, we will incur a holding cost $h_t(x_t + q_t - d_t)^+$, otherwise there will be backlogging cost $p_t(x_t + q_t - d_t)^-$. Together with the ordering cost $c_t q_t$, we define $L_t(x_t, d_t, q_t)$ to be our local cost at time t, in other words:

$$\text{Local Cost} = \text{Ordering Cost} + \text{Holding Cost} + \text{Backlogging Cost.}$$

At each time t, our goal is to choose q_t to minimize the expected total cost from period t to T. To do this we will find out the minimizer q_t (or approximate minimizer) of the quantities below:

$$\begin{aligned} & V_T(x_T, d_{1..T-1}) \\ = \; & \min_{q_T \geq 0} E[L_T(x_T, D_T, q_T) | d_{1..T-1}] \\ & V_t(x_t, d_{1..t-1}) \\ = \; & \min_{q_t \geq 0} E[L_t(x_t, D_t, q_t) \\ & + V_{t+1}(x_t + q_t - D_t, d_{1..t-1}, D_t) | d_{1..t-1}] \end{aligned} \tag{2}$$

where V_t is the minimum expected cost from period t to T, $d_{t_1..t_2}$ is the demand vector from t_1 to t_2. Since at time t, the future demand depends on previous demand (each future demand is a random variable), we use D_t to denote unknown demand. Then, this gives us two natural ways to "solve" the problem:

- Dynamic Programming: solve for q_t backward recursively using the formula above.

- Myopic: At time t, ignore V_{t+1} and only minimize $E[L_t(x_t, D_t, q_t)|d_{1..t-1}]$.

However, using dynamic programming leads to computational difficulties, as period $s > t$, D_s is a random variable that depends on previous demand $d_1, d_2, ..., d_{t+1}, ..., d_{s-1}$. We also need to work on $q_t, ..., q_T$ which are our unknown future decisions. These above problems make the dynamic programming approach intractable. On the other hand, although the myopic approach works extremely well in many cases[1], it may perform extremely poorly in other cases, which will be shown at the end of the paper.

We also need to consider leading time $L \neq 0$, i.e. it takes L periods to receive our order. Then we need define x_t = Net Inventory + $\sum_{j=t-L+1}^{t-1} q_j$ to be our inventory position, where $\sum_{j=t-L+1}^{t-1} q_j$ is our undelivered orders. We also need to modify our Local Cost to be

$$\begin{aligned} & L_t(x_t, d_{t..t+L}, q_t) \\ := \quad & c_t q_t + h_{t+L}(x_t + q_t - d_{[t,t+L]})^+ \\ & + p_{t+L}(x_t + q_T - d_{[t,t+L]})^- \end{aligned} \tag{3}$$

where $d_{[t,t+L]} = \sum_{s=t}^{t+L} d_s$.
In the simulation, we can always assume $c_t = 0$ by performing the transformation ($c_{T+1} := 0$):

$$\begin{aligned} \hat{h_t} &:= h_t + c_t - c_{t+1} \\ \hat{p_t} &:= p_t - c_t + c_{t+1} \end{aligned} \tag{4}$$

III. Dual Balancing Algorithm

In 2007, R. Levi, M. Pal, R. Roundy and D. Shmoys introduced a new approach[2] for the inventory control problem called Dual Balancing. Recall Local Backlogging Cost:

$$\Pi_t(x_t, q_t) = p_{t+L}(x_t + q_T - D_{[t,t+L]})^- \tag{5}$$

Now, we define Marginal Holding Cost:

$$H_t(x_t, q_t) := \sum_{j=t+L}^{T} h_j(q_t - (D_{[t,j]} - x_t)^+)^+. \tag{6}$$

Now, the Dual Balancing Algorithm as follows:
(i) Transform using Eq. (4);
(ii) Start at $t = 1$;
(iii) Solve the convex optimization problem:

$$\min_{q_t > 0} \max E[H_t(x_t, q_t)|d_{1..t-1}], E[\Pi_t(x_t, q_t)|d_{1..t-1}]; \tag{7}$$

(iv) Observe d_t, update $x_t, d_{1..t}$ and set $t = t + 1$;
(v) If $t < T$, go to step (iii).

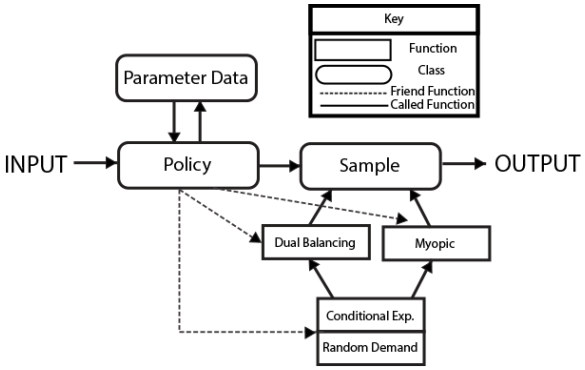
The above algorithm is called Dual Balancing since the minimizer q_t in Eq. (7) will satisfy

$$E[H_t(x_t, q_t)|d_{1..t-1}] = E[\Pi_t(x_t, q_t)|d_{1..t-1}] \tag{8}$$

Since both H_t, Π_t do not depend on q_s for $s > t$, the Dual-Balancing algorithm can be computed online, i.e. we can compute q_t without computing q_{t+1}, q_{t+2}.... It has been proven that the expected total cost in Dual-Balancing algorithm is at most two times the optimal cost[2].

IV. Implementation

Figure 1: *Software Design Diagram*



We set out to implement the Dual Balancing Algorithm and test its performance for some common demand distributions. We modularized our program by separating programming tasks into constituent parts. The `ParameterData` class saves parameters for the problem, while its child class, `Policy` contains various information for the algorithm, including, demand distribution type, distribution parameter, demand, ordering and cost for all time periods, and so on. Class `Sample` organizes all functions, classes and data collection.
Once data is read from input files, the program constructs a `Sample` class for each data set, the `Sample` class will construct a `Policy` class and update the `Policy` class as the `DualBalancing` and `Myopic` functions are called. The main challenge is to compute the conditional expectations in Eq. (5) (6) (8) using the function

`ConditionalExp` for some given distribution. Once we calculate the conditional expectations, we solve the minimization problem at current time t, then we receive a random demand from `RandomDemand` and continue. After all computation is finished, the `Sample` class will collect the data and the output will be written to a file.

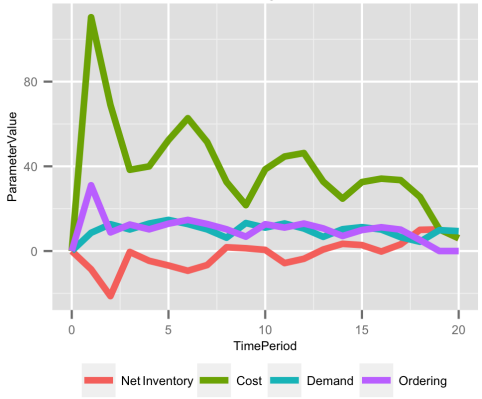
V. Results

In simulations, the default value of parameters are as follows:

- $c_t = 3, h_t = 1, p_t = 2, L = 5, x_0 = 0$.
- The sample size is 20.
- We considered four different demand distributions:
  1. $D_t = N(50, 5)$ are iid.
  2. $D_t - D_{t-1} = N(5, 1.581)$ with $D_0 = 30$.
  3. $D_t - D_{t-1} = B(10, 0.5)$ with $D_0 = 30$.
  4. $D_t - (D_{t-1} + D_{t-2} + D_{t-3})/3 = N(5, 1.581)$ with $D_0 = 30$.

Single Sample Path Behavior

Figure 2: *Dual Balancing Parameters*

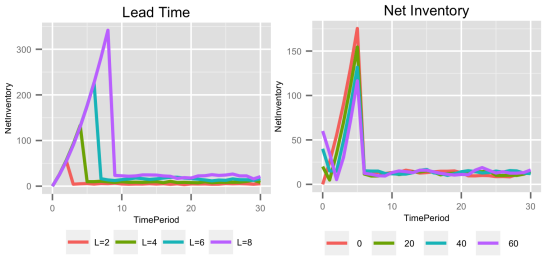


We choose values $T = 20, L = 2, D_t = N(20, 2.24)$ i.i.d., and compute using Dual Balancing. In the Fig. 2, we only show the result of one sample path. Since the initial net inventory is 0 and $L = 2$, the demand is going to accumulate and induce a nonzero holding cost. However, due to large order volume at the initial period, the cost decreased significantly when the first order arrived at $t = L + 1 = 3$. After that, the order volume stabilized and oscillated according to the change of d_t. The net inventory[1] is also stabilized and oscillates around 0. This means the holding cost and backlogging cost are close to zero. Most of the cost is due to orders and the cost curve is almost proportional to demand curve.

Since, the demand includes randomness, we will never be able to control our net inventory exactly at zero. Thus, Fig. 2 actually implies that the solution of Dual Balancing could be very close to the optimal solution.

Average Behavior

Figure 3: *Stabilization Across Different Initial Values of Net Inventory and Lead Time*

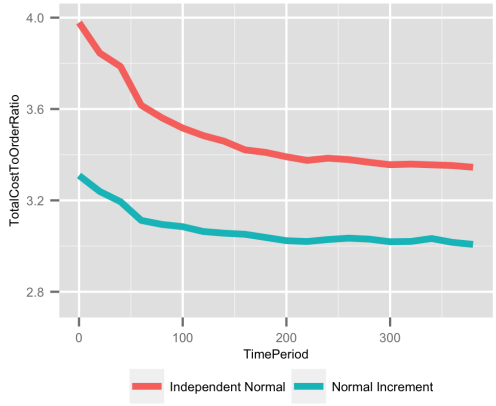


In Fig. 3, we use the Dual Balancing algorithm with $T = 30, D_t - D_{t-1} = N(5, 1.581)$ and $D_0 = 20$. These two plots demonstrate good convergence behavior. In the left figure, we choose initial inventory $x_0 = 0$ and plot net inventory for different lead times. In the left plot, we see no matter how large our lead time is at period $L + 1$ (the time we receive our period 1 order), the net inventory drops down to a low level immediately. Although all long-run net inventory values are close to zero, the larger leading time still leads to a slightly higher volume of net inventory. In the right figure, we fix $L = 5$ and change the initial net inventory value. If initial net inventory $x > 0$, it will support demand for multiple periods, and after consuming all initial net inventory, the absolute net inventory increases. However at $t = L = 1$ we always have a stable net inventory close to 0. And we can see the stabilized net inventory will not depend on x_0.

[1] Net inventory is computed by averaging absolute value of net inventory in each sample. Small net inventory indicates good policy.

Long Term Cost/Demand Ratio

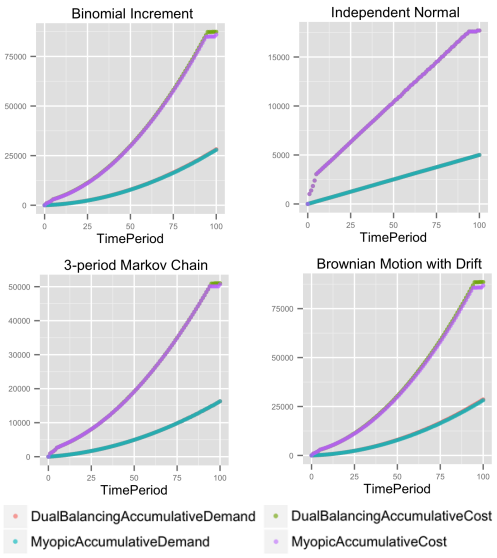
Figure 4: *Total Cost to Order Ratio: Independent Normal vs. Incrementing Normal Distribution*



Another thing we are interested in is whether Dual Balancing still performs well as T increases. In Fig. 4, both curves plotted the total cost over the total demand for different distributions. We can see that, in the long run, the cost per demand converges to a central value and will not blow up. This implies that in long run, the algorithm's output stabilizes. Moreover, since our $c_t = 3$, this ratio is at least 3. You may see for the "normal increment" distribution, the ratio converges to exactly 3, demonstrating that the Dual Balancing algorithm is nearly optimal.

Comparison to Myopic

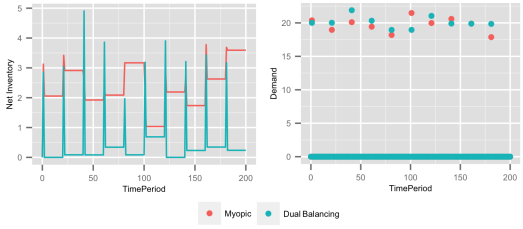
Figure 5: *Accumulative Demand and Accumulative Cost*



In Fig. 5, we compare the averaged sample paths of the two approaches. We notice that for all four demand distributions, the accumulative demand for both algorithms overlaps on the green line and accumulative costs overlap on the purple line. This indicates that for all four distributions, the performance, measured by accumulative cost, of dual balancing and myopic approaches are almost exactly the same. As demonstrated earlier[1], in many cases, the myopic algorithm works extremely well. Thus, Fig. 5 indicates that the dual balancing algorithm performs well.

Bad Example for Myopic

Figure 6: *Myopic vs. Dual-Balancing*



In all of the following figures, we averaged the sample paths. As shown in Fig. 6, we consider a periodic demand distribution: if $t \in \{0, 20, 40, 60, ...\}$, $D_t = N(20, 3.16)$, otherwise, $D_t = 0$. In Fig. 6, we can see that at critical periods (i.e. $t \in \{0, 20, 40, 60, ...\}$), net inventory spikes for for both algorithms. After this spike, the Dual Balancing algorithm's net inventory value drops to a much lower level than the value produced by the myopic approach.

Figure 7: *Myopic vs. Dual-Balancing: Ordering*

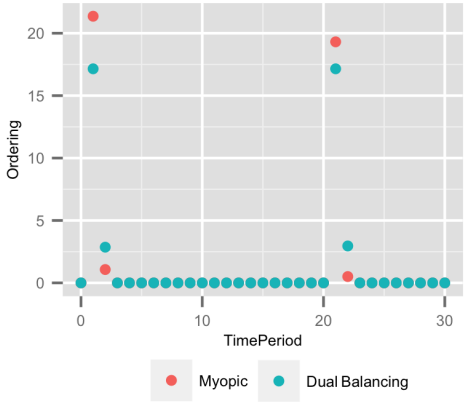
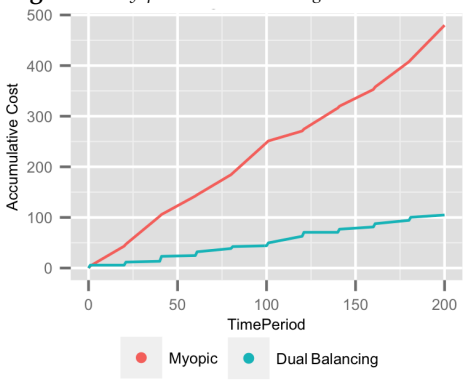


Fig 7 explains why dual balancing algorithm performs better than the myopic alternative. Comparing to the myopic approach, dual balancing always orders less than myopic at each critical period, giving a slightly higher expected backlogging cost at this critical period. However, the dual balancing approach successfully avoided possible holding costs during all the periods before next demand appears 20 days later as we see in Fig. 7. Then in next period, dual balancing ordered a little more product to eliminate backlogging costs. Then in following 18 periods, its cost is almost 0.

Figure 8: *Myopic vs. Dual-Balancing: Accumulative Cost*



As a result of this good strategy, Fig. 8 tells us that in this case, the Myopic is about 5 times the total cost of Dual Balancing in the long run. In fact, we could prove that the Myopic can be K times the optimal cost where K is the period of demand.

VI. Conclusion

In this paper, we have shown that the Dual Balancing algorithm behaves very well in many ways. It converges to equilibrium very quickly, the solution is almost optimal and is very stable as T increases. Dual balancing performs as well as the myopic alternative for all tests and simulations we performed, especially in our simulation of rapidly changing periodical demand. In this specific case dual balancing performs much better than the myopic approach.

VII. Contributions

Hao and Feng contributed the majority of work implementing both algorithms in C++. Blake managed source control (via GitHub), integration of project modules, as well as data visualization, and document formatting in R and LaTeX. Hao organized meetings and identified the paper we used as the source of our project. Feng structured the C++ code and collected all the data.

References

[1] Iida, T., P. H. Zipkin. 2006. Approximate solutions of a dynamic forecast-inventory model. Manufacturing Service Oper. Management 8 407-425.

[2] Retsef Levi, Martin Pal, Robin Roundy and David Shmoys, 2007. Approximation Algorithms for Stochastic Inventory Control Models, Mathematics of Operations Research, Volume 32 (2), pages 284-302.