UPPSALA
UNIVERSITET

SAFETY STOCK PLANNING AND SUPPLY CHAIN OPTIMIZATION IN STOCK STATUS

Submitted by
Ruoxi Li

A thesis submitted to the Department of Statistics in partial fulfillment of the requirements for Master degree in Statistics in the Faculty of Social Sciences

Supervisor
Yukai Yang

Spring, 2019

ABSTRACT

This paper proposes a safety stock calculation function based on their distribution properties and create a guideline for the stock status optimization problem. The motivation for this paper originates the cooperation with a drilling tools company, Epiroc Drilling tools AB. The safety stock calculation divides all items into three distribution and design the safety stock for each types separately considering the influence of service level value and lead time. During the process of guideline design, complicated production chain framework is taken into account through recursive algorithm. The stock status combination which can give the minimum storage cost is the optimal guideline for stock item and non-stock item. The time for approximating the global minimum through exhaustive search is remarkably reduced due to the application of Parallel programming and statistical model.

Keywords: Safety stock, Statistical distribution, Supply chain optimization, Recursive algorithm, Statistical model

Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Literature review . . . 1

1.3 Purpose . . . 2

1.4 Organization . . . 3

2 Theory and formulas **4**

2.1 Demand forecast . . . 4

2.2 Goodness of model fit . . . 5

2.3 Safety stock for different types of items . . . 6

2.3.1 Normal and Poisson distribution . . . 6

2.3.2 Iteration for negative binomial distribution item . . . 7

2.4 Recursive algorithm . . . 8

2.5 Brute-force search and parse techniques . . . 10

2.6 Parallel programming . . . 10

2.7 Naive and Unsupervised statistical model . . . 11

3 Analysis on supply chain data **12**

3.1 Safety Stock design . . . 12

3.1.1 Data cleaning . . . 12

3.1.2 Variables description . . . 13

3.1.3 Design of the calculation of safety stock . . . 13

3.1.4 Result of safety stock calculation . . . 16

3.2 Guideline design for Stock and Non-Stock item . . . 19

3.2.1 Problem detecting . . . 19

3.2.2 Data cleaning and preparation . . . 20

3.2.3 Structure Design . . . 21

3.2.4 Result of the guideline . . . 23

4 Conclusion and Discussion **26**

5 Acknowledge **27**

6 Appendix **30**

6.1 Appendix A: Safety stock calculation code . . . 30

6.2 Appendix B: Supply chain optimization on stock status . . . 34

1 Introduction

1.1 Background

As the revolutionary development of automation and robotics, the flow of items is accelerated due to the consequence of the increased production rate these days. This also arouses significant interest in the process of inventory optimization in order to reduce cost and increase the ability to cope with uncertainty problems in demand. Anupindi et al. (2004) defined inventory as the number of flow units under the boundaries of the process. Currently, most companies prefer to carrying inventory. This behavior can reduce the delivery cost and increase profit. Mahadevan and Hill (2012 and 2015) mentioned in their book the important functions of the inventory we should focus on when we want to improve inventory management are pipeline and buffer. Companies obtain some raw materials from their suppliers. Most of the time there exists a time difference between reordering time and receipt time as it takes some time for the raw material to transport from supplier to manufacturer. The time difference here is called pipeline. Buffer inventory with another name as safety stock is referred to the number of items stocked to deal with the time difference from supplier to manufacturer. In supply chain optimization area, what is the most suitable safety stock for each stocked item and if we can find approach to distinguish stock item and non-stock item in order to save the money and ensure the process of production chain are the hottest problem at the moment. At the same time, the statistical theorem has grown up and can be applied to different areas. There is a high demand for utilizing statistical approach to solve the supply chain optimization problems.

1.2 Literature review

Inventory can be classified as three stages which are input buffer, in-process buffer and output buffer. In order to be consistent, Anupindi et al. (2004) defined that inflows into inventory are referred to as supply and outflows from the inventory as demand. The main direction of safety stock optimization lies in demand forecasting. Petropoulos and Kourentzes (2015) compared the performance of different ways of predicting intermittent demand. He also developed his own R package to perform SBA approximation for intermittent data to improve the prediction result. Apart from predicted demand, safety stock can also be influenced by other factors. Klosterhalfen and Minner (2010) apply a simulation study to compare the performance between stochastic-service approach and guaranteed-service approach in safety stock optimization. They concluded that mostly lower service level can give better performance. Beutel and Minner (2012) found that linear regression is better than inventory optimization model constrained by cost and service level when exogenous variables influence demand.

When it comes to supply chain optimization, Buffa and Munn (1989) use the recursive algorithm to find the reorder cycle-time which can also give minimal logistics cost. Nagurney (2010) put forward an approach for supply chain network design and redesign. It can not only provide suitable levels of capacity and product flows but also minimize the total cost. Grigoroudis et al. (2014) applied a recursive DEA algorithm to optimize supply chain framework. The optimal supply chain framework is obtained with the principle of minimum cost. Bertrand et al. (2016) proposed a mixed integer linear program model to solve product integration problem in designing the supply chain structure. Inspired by approaches above, recursive algorithm, parallel programming, and probability

technique are used to solve the optimization for stock status.

1.3 Purpose

The paper is written in collaboration with the Supply Chain department of a drilling tools company, Epiroc Drilling Tools AB. For now, Epiroc has already developed its mature safety stock calculation system in an inventory management system based on replenishment order and demand order. It's able to adjust the safety stock according to the information of the item, purchase orders etc. received from the ERP system. However, the proceeding safety stock calculation model in the ERP system is ready to be established. Considering the delay between order point and receipt of items, it's essential to put forward the buffer stock calculation model based on the current fixed network for each item. The model will be able to increase throughput and avoid stock-outs. The prior information provided by the company tells lumpy items belong to the negative binomial distribution, slow-moving items belong to Poisson distribution, fast, positive trend, erratic and new items belong to normal distribution. However, the company doesn't have the classification guideline for different types of items. Therefore, in order to get the distribution of each item, they are fitted to these three distributions. The distribution which can give maximum log likelihood is the optimal distribution for this item. The calculation of safety stock for different distribution is different. Therefore, different safety stock calculation function for each distribution is nested. In the final model, the calculation function for safety stock can be applied for different items in different warehouses and companies with a different service level.

What's more, due to the lack of disciplines for items to be stocked or not, there's an urgent need to set a guideline to separate stock items and non-stock items. The model restriction is that the final products always need to be stocked. Then, the stock status can influence the aggregated lead time and indirectly change the total lead time of each item which will lead to the adjustment for the safety stock and finally has an effect on the total storage cost. It's a supply chain optimization problem. There are several challenges in this supply chain optimization approach. One difficulty is that there exists a complicated linkage effect on aggregated lead time due to the relationship between the final product and its' components. This paper uses the recursive algorithm to construct the network of all final products. From an operation optimization perspective, even though the exhaustive search is the most intuitive way to get the global minimum value, the rows of the matrix increase exponentially. There may be excessive situations sometimes which exceed the boundary of R. For example when it comes to solving the stock status of one product with over 30 components, the exhaustive matrix would have over one billion rows. Obviously, it is the most challenging part of this thesis. Finally, we combine statistical sampling, statistical model together with parallel programming to approximate the global optima in order to solve this problem.

1.4 Organization

The organization of the paper is as follows. Section 2 introduces the theory and formulas used for safety stock calculation and supply chain optimization. Section 3 describes the procedure of designing the safety stock calculation and constructing the framework of supply chain optimization. Moreover, the result of the safety stock calculation and suggested stock status for the final product are displayed in section 3. Section 4 presents the conclusion and discussion about the future improvement aspect.

2 Theory and formulas

2.1 Demand forecast

The items which are diagnosed with negative binomial distribution in the documents provided by the company are lumpy items. In order to generate the p^{th} quantile of the negative binomial distribution, we need to know the predicted demand of each lumpy item. It is quite tricky to make prediction due to the zeros in the transaction demand of these lumpy items. It is inappropriate to use a simple moving average to predict the demand. Syntetos and Boylan (2005) put forward an improvement on Croston's method by utilizing $1 - \frac{\alpha}{2}$ to reduce the bias for intermittent series data and α is the smoothing constant value. It displays the best performance among exponential weighted moving average and simple moving average. The demand is calculated as follows:
If $\alpha = 0$, then we get

$$x_t^{'} = x_{t-1}^{'}$$
$$q_t^{'} = q_{t-1}^{'}$$

When $\alpha \neq 0$,

$$x_t^{'} = \alpha_x x_{t-1} + (1 - \alpha_x) x_{t-1}^{'}$$
$$q_t^{'} = \alpha_q q_{t-1} + (1 - \alpha_q) q_{t-1}^{'}$$

where $0 < \alpha < 1$

$$Y_t^{'} = (1 - \frac{\alpha_q}{2}) \frac{(x_t^{'})}{(q_t^{'})}$$

In this thesis, rather than specify smoothing value, Scaled Mean Squared Error function is selected for optimization.

2.2 Goodness of model fit

When it comes to the evaluation of model fitness, usually there are three criteria always being talked about and they are the most commonly used approaches in model selection. They are Log-likelihood, Akaike information criterion, and Bayesian information criterion. The Log-likelihood is the logarithm of the likelihood function. Pickles (1985) defines that likelihood function is a parametric function that indicates the probability of the observed data being predicted by the model. The higher likelihood value, the better the model. This rule also applies to Log-likelihood.

Both Akaike information criterion and Bayesian information criterion are penalized approach. Akaike information criterion can be expressed as:

$$AIC = 2k - 2ln(L)$$

While Bayesian information criterion has the following expression:

$$BIC = kln(n) - 2ln(L)$$

Here, k represents the number of parameters in the model. n is the number of records in the dataset. $ln(L)$ is Log-likelihood. For model selection, the smallest AIC and BIC tell the best model.

In the distribution selection process, only the previous transaction demand is of our interest so the number of parameters is one. If it's not taken into account, the result won't be significantly affected. Moreover, Log-likelihood is more simple compared with AIC and BIC. So the value of Log-likelihood is selected as the criterion for distribution selection.

2.3 Safety stock for different types of items

According to the prior information, it is easy to know that all items can only display three distributions. However, there is not a specific separation for each type of item. The paper fits the transaction demand data of each item to three distributions and uses log-likelihood as the evaluation criterion of the model fit. The distribution which can give the largest log-likelihood value is the most suitable distribution of this item. After getting the best distribution for each item, the calculation of safety stock can be done for each item based on their distributions.

2.3.1 Normal and Poisson distribution

The safety stock is the stock between reorder time and the time when you get the item to safeguard against unforeseen demands. The lead time is the time span between reorder time and arrival time. The calculation of safety stock aims to prevent the demand fluctuation during lead time. The service level stands for the percentile of the fluctuation that we want to ensure that we have the stock. As it's also influenced by the service level, it is designed as follows:

For Normal distribution,

$$SS = (F^{-1}(s) - \mu) \times \sqrt{t}$$

where:

$$F^{-1}(s) = \mu + \sigma\theta^{-1}(s),$$
$$\theta^{-1}(s) = \sqrt{2}erf^{-1}(2s-1),$$
$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dx$$

For Poisson distribution,

$$SS = (P(s,\lambda) - \lambda) \times \sqrt{t}$$

where:

$$P(s,\lambda) = \sum_{i=0}^{s} \frac{e^{-\lambda}\lambda^s}{\tau(s+1)}$$

Here, SS is the safety stock, s represents the desirable service level that people in charge enter while μ is the mean of normal distribution, λ is the mean of Poisson distribution and t is the weekly lead time.

2.3.2 Iteration for negative binomial distribution item

Different from Normal and Poisson distribution, negative binomial has more strict restriction to the data. Referring to the prior information provided by the company, those items which belong to negative binomial distribution are lumpy items. Therefore, there may be some zero demand inside the data which violate the restriction of inserted negative binomial distribution in R. To get the safety stock for negative binomial distribution item, the probability mass function for different service level regarding different lead time is developed by the company. The equations are as follows.

If n=0 then

$$f(0) = (\frac{1}{z})^{\frac{\lambda}{z-1}}$$

while n>0 then

$$f(n) = \left[\frac{(\frac{\lambda}{z-1}) + (n-1)}{n}\right] \times \left[\frac{z-1}{z}\right] \times [f(n-1)]$$

where

$$z = \left[\frac{LT \times \sigma^2}{\lambda}\right]$$

λ is the demand forecast over lead time, σ^2 is the variance of the previous transaction demand. The safety stock is calculated by $\sum_{i=0}^{n} f(i)$ until the value is larger than or equal to the desired service level. At this time, the safety stock is n. In addition, in order to prevent extreme cases from happening, the following is defined:

If $\lambda = 0$ (no forecast demand value or forecast demand is 0) then safety stock is set to 0.

If $n = 0$ and the item's predicted demand is larger than 0, then n is set to be 1. This setting aims to prevent those with very small numbers of transaction demand from producing no safety stock.

2.4 Recursive algorithm

Rubio-Sanchez (2017) defines a recursion problem to be a process that function $f(n)$ is determined by the i preceding terms of $f(n-1)$, $f(n-2)$, ..., $f(n-i)$, where $i \geq 1$. Thomas (2004) mentioned in the book that iteration is an effective method to find the recurrence relation of the function and it can be applied to solve these problems in two steps. First, it is crucial to detect the recurrence pattern and apply it to iteration. Second, induction needs to be used to prove that the formula correctly holds for every integer n.

The Recursive algorithm used in this optimization problem aims to calculate the aggregated lead time. The framework of each final product is composed of at most four layers and each layer may have several items or raw materials. The data exported from the ERP system are organized from top-level items to the bottom-level items. The stock status of the upper-level items can influence the aggregated lead time of the lower level items. The aggregated lead time of current level item can be influenced by it's stock status and it's upper-level items. In this recursive algorithm, each time when looking upon one component, the paper tries to trace back to the previous row to detect if there is higher level component. If it does have a higher level component, the aggregated lead time is also affected by the stock status of higher level items. Every time for each component, the algorithm stops once it finds the top level item. More intuitively, the full structure of one final product is as the following picture.

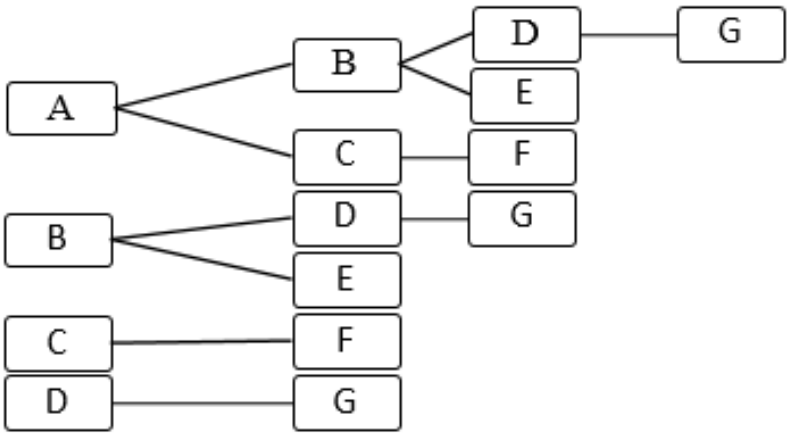

Figure 1: Product Structure

In Figure 1, the final product A which is also in the top layer has components B, C, D, E, F, G. At the same time, B, C, D are parent product of other components.

There are three columns in the data set indicating the framework of the product and components. A_i represents the parent product in i^{th} row. B_i is the previous level component of the component C_i in i^{th} row. s_i is the sum of accumulated days before completion, days before the manufacture of the component finished and lead time. l_{c_i} is named as timeadd of material C_i in i^{th} row which is stock status of the i^{th} item. This variable s_i implies that the stock status of this item.

In this recursive algorithm, the paper iterate among the whole product framework. Every time in i^{th} row, it is checked whether this component is influenced by the stock status of the front level until it reaches the top level. The restriction of this optimization problem is that the top level product needs to be stocked every time. The recursive algorithm structure is as follows.

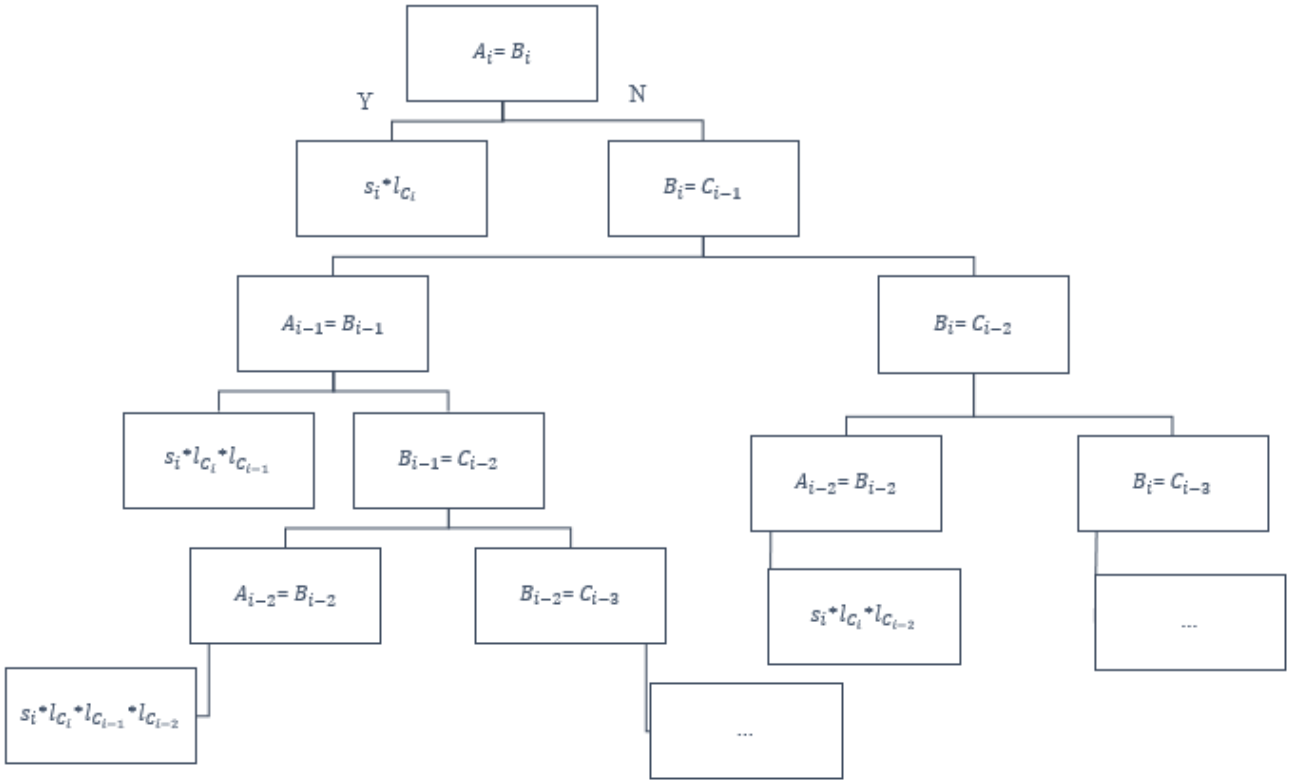

Figure 2: Recursive algorithm design

The correlation of components to product has three characteristics.

- The recursive algorithm ends once when $A_{iter} = B_{iter}$, iter = i, i-1, i-3...
- If $A_{iter} \neq B_{iter}$, then check $B_{iter} = C_{jter}$, iter = i, i-1, i-2,..., jter = i-1, i-2, i-3... and the difference between jter-iter differs from 1 untill jter satisfy $A_{jter} = B_{jter}$
- Whether multiple $l_{C_{iter}}$ to S_{iter} or not and which l_{C_iter} needs to be multiplied are jointly determined by $B_{iter} = C_{jter}$ and $A_{jter} = B_{jter}$

Recursive algorithm can be divided into four types. Hutton (2007) points that if there are two or more functions defined by each other, then it is mutual recursion problem. It certificates that the mutual recursive algorithm can be used to analyze the influence of aggregated time by the correlation between different levels of items.

2.5 Brute-force search and parse techniques

To get the best stock status decision for all items, it should satisfy the criterion that a column of stock status of each component for one final product can give the minimum total cost. The stock status indirectly influences the total storage cost by affecting the aggregated lead time. The aggregated lead time indirectly changes the total lead time of each component which has an impact on safety stock. The total storage cost equals the sum of the safety stock amount of each item times the storage cost per item. There reflects a nonlinear relationship between stock status and total cost due to the inner relationship between each item. Moreover, the recursive algorithm inserted to calculate the aggregated lead time through detecting the relationship structure between components and final product adds more difficulty to this task. In order to get the desirable result of stock status, Brute-force search can be used to solve this problem.

Brute-force search is also called exhaustive search. In this case, the stock status for each component is either 0 (non-stock) or 1 (stock). The exhaustive search goes through all possible situations of stock status and calculates the total costs in all situation. However, how to generate the exhaustive matrix is another tough problem. Then, in this paper, expand.grid and parse techniques are applied to generate exhaustive search matrix.

Expand.grid can generate the Cartesian product of this argument. Grune and Jacobs (2008) define pharsing in their book as the process of generating a linear representation of a given grammar. Paired character (0, 1) is replicated for n times in expand.grid, where n equals the number of components in the product. After that, expand.grid expression is inserted in the parse statement which aims to create the exhaustive search matrix.

2.6 Parallel programming

During the exhaustive search period when there are over 30 components in one product structure, the exhaustive matrix is composed of over one billion iterations. As the exhaustive matrix can be extremely large when it comes to those products with many components, the nested recursive function together with the function to get the variance of transaction demand from previous data set in order to calculate safety stock makes the execution time even longer. By utilizing more CPUs in the computer with the help of parallel programming, the iteration time reduction solution stands out at this time point. Pacheco (2011) mentioned a similar occasion similar to the exhaustive search in this thesis. He said that there is a demand for parallel programming when tremendous data are generated. Nowadays, it's more common to see the laptops are equipped with multiple processor cores. The core can also be called CPU. This development makes conducting parallel programming in one computer possible. A R package SNOW is developed to perform parallel programming in windows system. Even though this SNOW package makes parallel programming in R happen, it requires more thorough knowledge about managing clusters. Knaus et al. (2009) introduced a more handy and flexible R package snowfall to apply parallel programming. In this package, it is not necessary to know cluster implementation and configuration. What's more, the snowfall package allows for loading packages in each cluster. This is important for the successful execution of the nested function to calculate the cost of different stock status in exhaustive search. Considering these advantages of snowfall, this package is chosen for the parallel computing.

2.7 Naive and Unsupervised statistical model

Naive represents a way of using random sampling to get the global optima. The basic principle can be described as follows. The probability of getting global optima of m samples for total n cases is:

$$
\begin{aligned}
&1-(1-\frac{1}{n})(1-\frac{1}{n-1})(1-\frac{1}{n-2})(1-\frac{1}{n-3})\cdot\ldots\cdot(1-\frac{1}{n-m+1}) \\
&=1-\frac{n-1}{n}\cdot\frac{n-2}{n-1}\cdot\frac{n-3}{n-2}\cdot\ldots\cdot\frac{n-m}{n-m+1} \\
&=1-\frac{n-m}{n} \\
&=\frac{m}{n}
\end{aligned}
$$

It's clear that if the sample size m approximates n, the minimum value in the sample has higher probability to be the global optima. Another approach is applied to approximate the global optima for those with many components stems from the principle of simulated annealing (SA). SA is a probabilistic approximation algorithm that can be used for discrete occasions. Bertsimas and Tsitsiklis (1993) introduced the method through a case of cost function on a finite set. The process can be expressed in the following explanation. Firstly, a new position j is generated and it is a neighbor of i. Then, the probability P of changing new position from i to j is calculated.

$$
P=\begin{cases}1 & \Delta J\leq 0\\ e^{-\frac{\Delta J}{T(t)}} & \Delta J\geq 0\end{cases}
$$

Here, $\Delta J=J(j)-J(i)$. $T(t)$ represents the temperature at time t.

For the annealing process, Juan et al. (2003) pointed in their article that it's necessary the simulation should last for a long time. Usually, the process starts with a high temperature which nearly allows for all movements. Long simulation time provides enough time for the temperature to cool down. In other words, this can make sure to reach a steady result. As for the exhaustive search matrix in this thesis, the paper borrows the idea from simulated annealing then create a statistical model to approximate the global minimum.

The model can be expressed as follows.

$$
\eta_{i+1}=\eta_i+\epsilon_i \quad i=1,2,3...n
$$

Where η_{i+1} is the next stock status of all components, η_i is the current stock status. ϵ_i is a column with 0 and the length is the number of components. Every i time, it randomly select one position, if the corresponding η_i is 1 then the value in that position is -1 and if the value in the corresponding η_i is 0 then the value in that position is 0.

3 Analysis on supply chain data

3.1 Safety Stock design

3.1.1 Data cleaning

In the ERP system, there exist two useful data sets for the calculation of safety stock. MITTRA is the daily transaction demand record from 1^{st} Jan 2017 till the most recent date of all items. MMS002 provides the label information that indicates the stock status, lead time, stock transaction type and warehouse type information. In MMTTRA data set, our interest is only on stock transaction type consisting ordering. Thus, rows consist of specific ordering type is filtered before merging two data sets together.

The filtering procedure is followed by joining two data sets together and aims to get the information of all currently stocked items. First, items with stock status records are selected. Then, rows with the warehouse type including 1 are excluded. Next, planning policy involves 1 is filtered. Positive transaction demand records mean those items are returned or are different from the ordinary demand and that's not our focus this time. Finally, only those with negative transaction demand enter.

Even though the effort was made to reduce the amount of data, there still exists millions of rows in the data set. What's more, the ERP system does not mark those days without records with zero transaction demand. Therefore, a weekly time-focused transaction demand data frame is needed. In order to achieve this weekly based data set, three separate data frames are created. The first data frame is the weekly transaction demand data frame transformed from the previous daily joint data. The second one is a vacant data frame with the same week span from 20170101(starting week) till the most recent week for each item. The third data frame is used to detect the initial release date of each item. After filling the information in the second data frame with the data from the first one, the third data frame is utilized as the indicator which tells the suitable starting position for each item to insert zero. At last, the paper deletes those rows with NA values and separate the key to the company number, warehouse number and item number as they will be useful parameters in the future function.

3.1.2 Variables description

df_week_clean	Weekly transaction demand dataframe for all items
compno	Company number
wareno	Warehouse number
df_dis	Distribution table for each items
itemno	Item number
precent	Desired service level

3.1.3 Design of the calculation of safety stock

The calculation of safety stock can be divided into two parts. Firstly, it's important to find suitable distribution for each item so a function which can be applied to find the most suitable distribution is designed. Simultaneously, the useful information such as demand forecast and lead time are generated and subtracted from the weekly transaction data set during this time. Secondly, with the knowledge of the suitable distribution of each item, mean and variance of that distribution can be easy to get and applied to safety stock calculation. Different methods of safety stock calculation based on their distributions for each item are carried out to get the safety stock value.

As the company wants to focus on the items of different companies and warehouses, the company number, warehouse number are set to be parameters of this function together with the weekly transaction demand data set which provides the historical transaction demand and lead time information. The company provides the prior distribution information which tells the possible distributions of all items. In previous studies, the company found that lumpy items belong to the negative binomial distribution, slow-moving items have Poisson distribution characteristics, fast, positive trend, erratic and new items usually display normal distribution trait. Then, the transaction demand data of each item in one company and warehouse is fitted to these three distributions. There are three indexes can be used to evaluate the fitness of each distribution which are AIC, BIC and negative Log-likelihood. AIC is the abbreviation of Akaike information criterion while BIC is the abbreviation of Bayesian information criterion. These two methods share a lot of similarities in measuring the goodness-of-fit and they all involve Log-likelihood to get their results. As there is no huge difference between these three approaches, the Log-likelihood is chosen because it has the simplest formula. For each item, the Log-likelihood value of three distributions is calculated for the evaluation of model fit. Meanwhile, as negative binomial distribution has more strict restrictions for the data, errors of fitting the data to negative binomial distribution are allowed during the distribution fitting process. Then, the error values and NA values are substituted with negative infinity which aims to exclude them from the comparison procedure. Next, the Log-likelihood values of three distributions for each item are compared. The largest Log-likelihood value indicates the optimal distribution.

Considering the calculation for safety stock of negative binomial distributed items requires predicted demand, Syntetos-Boylan approach is also utilized here to forecast the transaction demand for all items. The lead time of each item can also be found this time in the distribution table.

In this stage, the function returns a distribution table. In this table, it has the key, company number (CompNo), warehouse number (WareNo), item number (ItemNo), lead time (Leadt), distribution (Dist) and predicted demand (EOD). The following is part of the distribution table of all items in company 1 and warehouse 001. Because of the confidential reason, there is some blurring in the item number. In distribution column, 1 is Normal distribution,

2 is Poisson distribution and 3 represents Negative binomial distribution.

Table 1: Distribution table for company 1 and warehouse 001

Key	CompNo	WareNo	ItemNo	Dist	Leadt	EOD
1_001_A1	1	001	A1	3	3	18.28
1_001_A2	1	001	A2	1	3	89.64
1_001_A3	1	001	A3	3	3	92.80
1_001_A4	1	001	A4	3	3	14.07
1_001_A5	1	001	A5	3	3	236.80
1_001_A6	1	001	A6	3	35	2.39
1_001_A7	1	001	A7	3	3	2.56
1_001_A8	1	001	A8	3	3	2.56
1_001_A9	1	001	A9	3	60	9.35
1_001_A10	1	001	A10	3	7	75.79
1_001_A11	1	001	A11	3	7	424.48
1_001_A12	1	001	A12	3	7	416.69
1_001_A13	1	001	A13	3	60	75.79
1_001_A14	1	001	A14	3	72	19.96
1_001_A15	1	001	A15	3	7	12578.14
1_001_A16	1	001	A16	3	7	2149.60
1_001_A17	1	001	A17	1	15	407.59
1_001_...	1	001	...	...	...	...

After getting the optimal distribution table for all items in one company's warehouse, all the useful information for safety stock calculation has been obtained. However, the lead time in the distribution table has not converted to weekly lead time. So, it is divided by 7 in order to get a consistent result at the beginning of the safety stock calculation.

The definition of safety stock says that safety stock is the stock between order time and receiving time. For Normal and Poisson distribution items, safety stock is equal to the lead time multiply the difference between the quantity of desired service level and Mean.

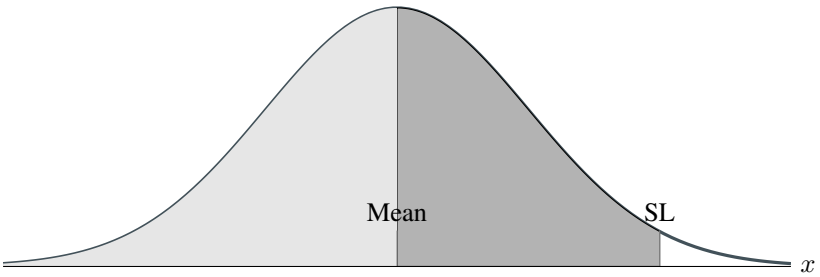

Figure 3: Normal Distribution

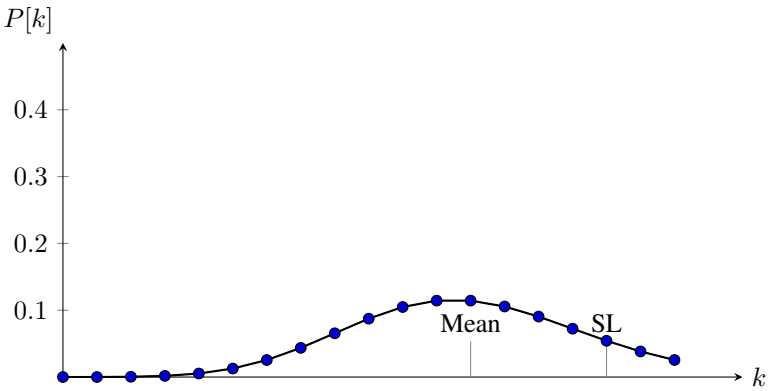

Figure 4: Poisson Distribution

In two graphs above, SL represents the service level which is the $n\%$ quantile of that distribution.

For those belongs to the negative binomial distribution items, the distribution is also a sign for lumpy item type. Lumpy items usually have intermittent demand. Sometimes, the transaction demands are zero during one period for one item. The existence of zero transaction demand prevent us from applying common inserted negative binomial function in R. Thus, a new calculation formula is characterized. The principle of designing the algorithm is summing the probability value generating from 0 to n stock amount until the sum is larger than or equal to the required target service level.

The introduction of detailed formulas is in Section 2.2.

3.1.4 Result of safety stock calculation

The result of the safety stock calculation function consists of three main elements in one list. The first is the proposed safety stock based on previous transaction demand and the optimal distribution. Till now, the current safety stock values in the ERP system are set manually based on the operation experience. There is no precise contrast can be used to evaluate the accuracy of the safety stock we get from this algorithm. As there are some new items with less than 10 transaction demand records, this may influence the accuracy of the proposed transaction demand. So, the second is the number of non-zero transaction records of this item. This can briefly describe the reliability of the result. What's more, the logistics data may have interaction with time and different types of items may display different trend in transaction demand. Apparently, this also needs to be taken into account especially when it comes to those items showing a decreasing or increasing characteristic. Then, every time when we calculate the safety stock for certain items, the results come with a histogram plot. The histogram plot set week as the x-axis and transaction demand as the y-axis. This gives an intuitive impression of the trend of transaction demand and will be helpful for the final decision for the safety stock.

The calculation algorithm can be applied to all items in different warehouses and companies. After getting the distribution table of one warehouse in one company, you can use the designed function to get safety stock for specific items and service level in that warehouse. Moreover, it's also feasible when there is a table with different item number and service level in one warehouse of one company. It's easy to calculate the safety stock with the help of loop.

Below, it is an application case of the safety stock calculation function for one item A in company 1 and warehouse 001. With the information in the distribution table we get in the previous example, the function gives a result which advises us to have 677 A in the stock. The decision is based on 101 none zero values. Additionally, the histogram of this item displays the trend of weekly transaction demand of the item and will be beneficial for the final decision about safety stock.

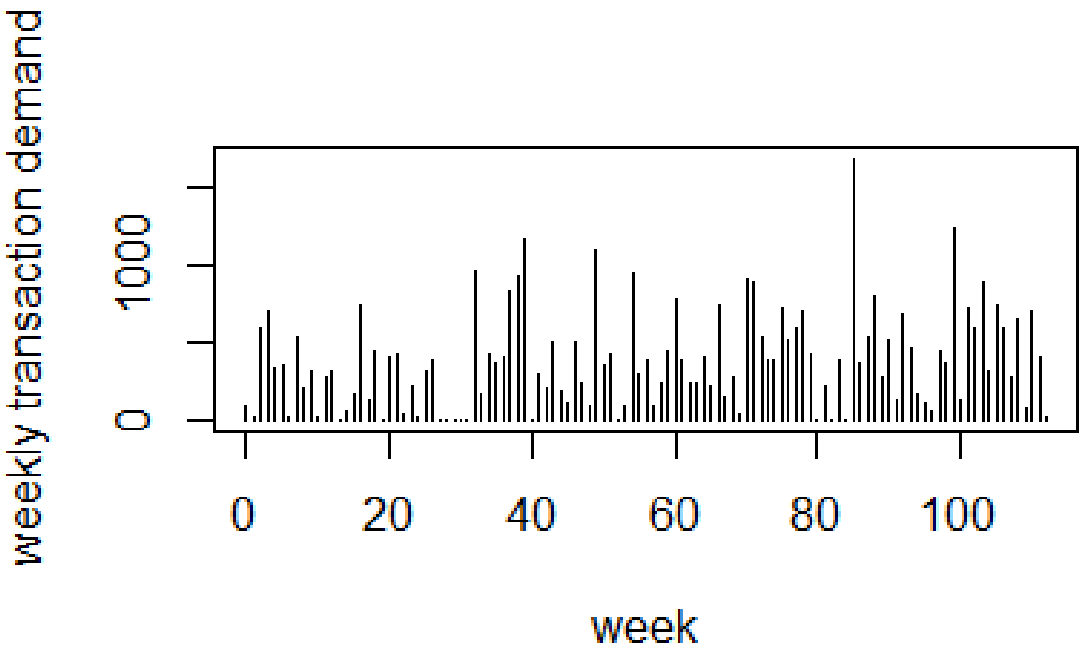

Figure 5: Histogram of transaction demand

Another application of this safety stock function is that it can be used in a loop to calculate the safety stock for different items or analyze how different service level affect the safety stock for one item.

Table 2: Safety stock table

CompNo	WarehouseNo	ItemNo	Service level	Safety stock
1	001	A	0.9	25
1	001	B	0.9	156
1	001	B	0.8	94
1	001	C	0.9	3410.87
1	001	D	0.9	1204
1	001	D	0.8	677
1	001	D	0.75	525
1	001	E	0.9	286.75
1	001	E	0.8	123.67
1	001	E	0.75	79.43

In Table 2 above, it is obvious that the change of service level may have a remarkable influence on the proposed safety stock. Looking upon item B, D, and E in distribution table, B and D are both diagnosed with negative binomial distribution while the transaction demand records of E suits normal distribution best. This reflects that the huge differences between safety stock for the same item with different service level may happen no matter what distribution they are. It is a common problem that the decision maker should pay more attention and make decisions on the trade-off between high storage cost and stock-out cost.

Compared with other approaches uesd to calculate the safety stock, the calculation function developed by this paper solves the problem creatively when we only know the demand information. The separation of different distributions for all items makes the calculation for the safety stock more specific. Especially, the calculation for lumpy items is abstracted from the probability mass which can tolerate the existence of zero transaction demand.

3.2 Guideline design for Stock and Non-Stock item

3.2.1 Problem detecting

To get the desirable guideline for stock status for different items, it is critical to detect what kind of problem it is and apply suitable approaches to solve the problem. The target in this stage is to get the total minimum storage cost of the final product and all components in the final product framework. There is an indirect linkage effect between stock status and the total cost of stocking all items. Every time there is a tiny change in the stock status of one component, the final cost of stocking all components and final product change simultaneously.

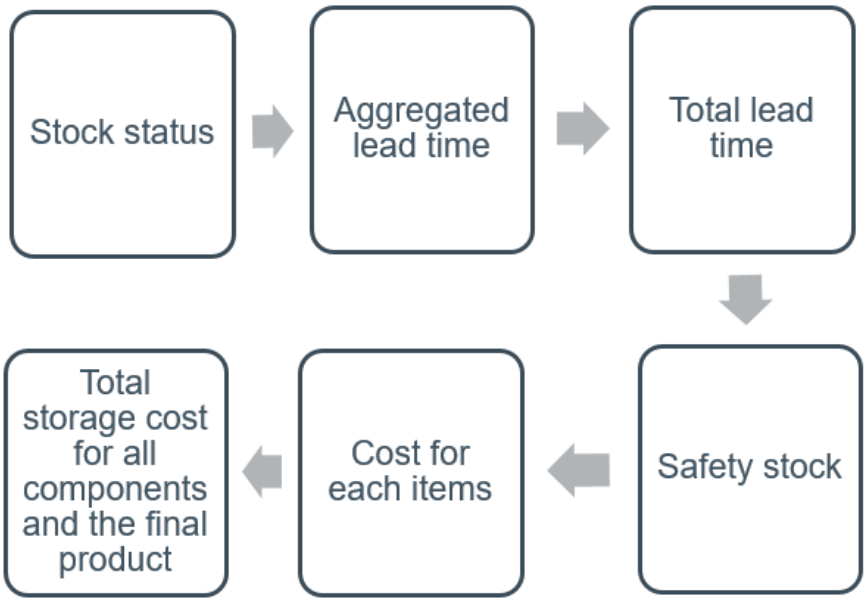

Figure 6: Linkage effect

Firstly, the restriction of this problem is that the final product in each product framework ought to be stocked. The stock status of each component has an effect on the aggregated lead time according to the composition structure of each component and final product. The common framework of one product is introduced in the theory part. The structure of the final product consists of at most four layers. The stock status of higher layer items and the stock status of the current item jointly determine aggregated lead time. The total lead time is the maximum aggregated lead time in the current layer as each layer may have several items.

In order to calculate the safety stock, it is essential to know the distribution of each item besides the total lead time. During this task, it is assumed that all items obey the normal distribution. The storage cost of each item equalize the storage cost per item multiply the safety stock while the total storage cost of the final product is the sum of the cost of stocking itself and its' components.

Essentially, designing the stocking guideline for each item is a supply chain optimization problem. In one framework of final product k, the problem can be described as follows.

- S_{ik}: safety stock of i component in the framework
- C_{ik}: cost of stocking i component

- l_{ik}: total lead time of component i
- s_{ik}: stock status of component i
- d_{ik}: normalized service level of component i
- σ_{ik}: standard deviation of transaction demand of component i

$$minimize \sum_{i}^{n} S_{ik} C_{ik}$$

$$S_{ik} = \sqrt{l_{ik}} d_{ik} \sigma_{ik} s_{ik} \quad \forall k, i$$
$$l_{ik} = f(s_{nk}) \quad \forall k, n = i, i-1, i-2, ...$$
$$s_{ik} \text{ binary and } s_{1k} = 1 \quad \forall k, i$$

As the complexity of the structure is unable to be written in a clear formula, it is introduced in the form of a tree plot in section 2.4. The relationship framework of components in each layer is given an example in this part. The calculation of aggregated lead time based on stock status is also explained. The target of this problem is to find the optimal stock status for each component which can achieve the minimum storage cost goal. However, the indirect linkage relationship adds the difficulty of this problem. Exhaustive search simplifies this problem by listing all the possible combination of stock status for all components. Every time the iteration can give one result for one stock status situation. With the knowledge of the final cost for all stock status situation, the minimum cost can give the optimal stock status.

3.2.2 Data cleaning and preparation

There are three data sets need to be used in this optimization problem. One is the previous weekly transaction data in safety stock design. However, as in the first task, the only focus is on stocked items, this time the research area has extended to all items regardless of their stocking status. Furthermore, the transaction demand data become extremely large due to the decrease in the filtering criterion. It's necessary to come up with a solution to reduce the useless data. This time, the core is to find the standard deviation of transaction demand for each item in the product structure data set. So the irrelevant rows are deleted if they are not products or components in product structure data set before I convert the daily transaction demand data set to weekly.

In order to get the standard deviance of transaction demand for each item, a function is created. The result of the function includes two parts. The first is the standard deviation of previous transaction demand for each item. The second is the lead time for each item. Later, this function can be nested in the structure design functions to get aggregated lead time.

Apart from the weekly transaction data set and the product structure data set, the data set involves the storage cost

for each item is also imported. As the focus is the prize of stock items, the final data set only filter item number and storage cost out.

3.2.3 Structure Design

The idea of designing the structure to get the minimum storage cost can be divided into three steps. At first, it's critical to construct two tables. First table need to tell the structure information of the final product. The calculation of the aggregated lead time for each layer is based on Table 3 and stock status in Table 4. Next, the tables indicates the lead time, service level and storage cost for each component is generated. By comparing the aggregated lead time for each layer's components and their own lead time and selecting the maximum one, it's easy to get the total lead time then calculate the safety stock for all components and the final product in the second step.The explanation for the first two steps is combined with an example below.

Table 3: Product structure chain

PSPRNO	PSPRN1	PSMTNO	PSACDB	PSMDBF	PSLEAL	Agg
A	A	B	0	10	30	40
A	A	C	0	4	5	9
A	C	D	4	5	6	0
A	D	E	9	6	20	0
C	C	D	0	5	6	0
C	D	E	5	6	20	0
D	D	E	0	6	20	26

Table 4: Item information

Itemno	M9UCOS	Status	LTAdd	LT	TotaLT	STDdev	SL	SS	Cost
A	1000	1	0	10	40	10	95%	104.03	104029.70
B	200	0	1	30	30	15	95%	0	0
C	20	0	1	5	5	15	95%	0	0
D	18	1	0	6	26	20	95%	167.74	3019.37
E	40	0	1	20	20	10	95%	0	0

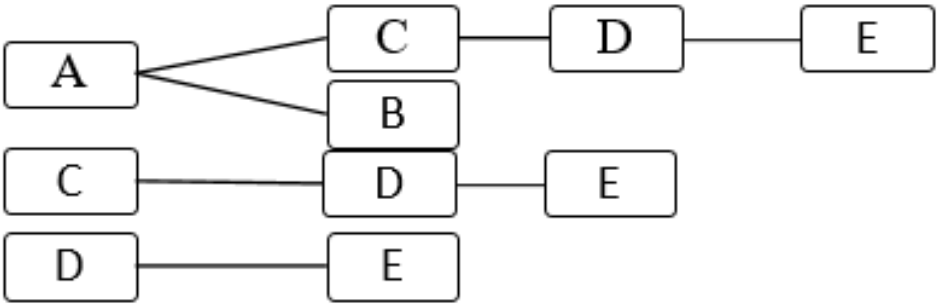

Figure 7: Product A structure plot

Here in this example, A is the final product. Table 3 is one subset of the product structure data set. The first three columns indicate the relationship and structure of the final product. In the product structure table, all data are arranged from high-level item to low-level item. As shown in Figure 7, each column stands for one level. Not only the chain of final product A should be considered, but the possibility that any components can be the final product in its own chain ought to be noticed. The value of aggregated lead time (Agg) in Table 3 is jointly determined by the relationship of the item and stock status (Status) in Table 4. In Table 3, if we sum PSACDB, PSMFBF, and PSLEAL, it gives the lead time for each final product in PSPRNO based on different components in PSMTNO.
In Table 4, LTAdd equals 1 minus value in Status column which means that if the item is stocked, then the lead time for this item is zero. Furthermore, high-level items can influence the stock status of low-level items. For example, if item C is stocked then the lead time for D and E are all zero. It is necessary to find the relationship of all higher level items for the current component in PSMTNO. Then, the recursive algorithm is put forward to solve this problem.
Three functions are nested to achieve recursive algorithm. In the beginning, the function uses the value in PSPRNO and PSPRNO test if it reaches the top level item. If they are equal, the aggregated lead time of this component only determines by its own stock status. If they are not equal, then it enters the next function which focuses on the value in PSPRN1 and PSPRNO and aims to find all higher level items. All the effect of stock status for the aggregated lead time of the current item is considered in the calculation formula. Beside the LTAdd of itself, all the correlated LTAdd of higher level items become a multiplier for aggregated lead time calculation in the third function. Briefly speaking, the recursive algorithm iterates over each component and traces back to the previous rows to detect whether there are some higher level items or not. It stops once it reaches the top level item. After the calculation of aggregated lead time, the construction of table 3 finishes.
The total lead time (TotalLT) column in table 4 is the actual lead time of each item considering the influence of stock status and item relationship.The total lead time of each item is maximum within aggregated lead time among their chain as the compared with their own lead time. The maximum value is the real lead time of producing the final product because the final products may have several branches which may take different time. For those components in the lowest level, their lead time is the same as the value in LT column. If we don't want to stock one item then we don't need any the safety stock for this item. So, the formula of safety stock calculation includes Status column as a multiplier. The total storage cost for each component is easy to get once we know the cost

of per stock (M9UCOS). By summing the total cost (Cost), the final storage of this final product A based on this stock status situation (Status) comes out.

Last but not least, it is critical to generate the exhaustive search matrix in order to get the total storage cost for the different stock status situation. Expand.grid and parse techniques are combined to get this matrix. Expand.grid can generate a data frame from all combinations of supplied factors. Paired character (0, 1) is replicated for n times inside expand.grid expression. Afterwards, the expression is evaluated to get the exhaustive search matrix.

With the exhaustive search matrix, it is possible to get the total storage cost for all stock status combinations. The rows of exhaustive search matrix exponentially increase according to the number of components in the final product. The most complicated product in the company has 39 components.The matrix can be extremely large when the number of components exceeds 26 and the execution time will be several days. To relieve the heavy calculation burden, parallel programming is first used. The idea of parallel programming is allowing the computation running at different CPUs at the same time to save time. The company has 8 CPUs on their laptops. By applying 7 CPUs to the exhaustive search process, the total time is significantly shortened by one 1/7. With the help of the snowfall package, it's inevitable that all functions and data frame in the global environment should be replicated in all other 6 CPUs. After that, it's possible to run the function parallelly in 7 CPUs.

Even though parallel programming helps a lot in reducing the time for iteration, still the time for getting the suggested stock status is long for those complicated products. Both naive and the statistical model can be applied to approximate the global minimum. Naive and the statistical model are based on different principles. They display different performance. Seen from the result, the statistical model takes much longer time than naive approach and sometimes it traps in the same stock status combination for several iterations which indicates lower efficiency. It is reasonable because the simulated annealing usually work well in a continuous situation. However, this time the paper makes some adjustments to it to adapt to the discrete case. Piecewise stock status reduces the accuracy of the algorithm itself and increase the time to find the minimum cost combination. In conclusion, parallel programming together with naive sampling approach can efficiently approximate the global minimum value and get the optimal stock status combination.

3.2.4 Result of the guideline

The result is intuitive for the stock status optimization. All costs based on different combinations of stock status are calculated. The minimum cost can be found among all costs. Moreover, the corresponding stock status of each components can be easily derived once we know the position of the minimum cost. Finally, there are two tables. One is the structure chain of the item which displays the structure of the final product and relationship between each components. Another is the composition of the final product which involves the suggested stock status, safety stock and storage cost per item. Following is an example of one final product A. The exact number of A is blurred due to confidential reason.

In this product structure table, the aggregated time is calculated by the recursive algorithm according to the stock status in Table 6.

Table 5: Product Structure

PSCONO	PSPRNO	PSPRN1	PSMTNO	PSACDB	PSMDBF	PSLEAL	AGG
1	A	A	B	9.31	20.52	29.83	29.83
1	A	B	C	11.31	20.55	7.00	38.86
1	A	B	D	11.31	2.27	9.50	23.08
1	A	D	E	13.58	10.58	5.75	29.91
1	A	E	F	24.16	5.75	8.00	37.91
1	A	A	G	0.00	9.31	9.92	19.23
1	A	G	H	11.31	9.98	3.00	24.29
1	A	A	I	0.00	9.31	21.64	30.95
1	A	I	J	13.31	21.72	3.00	38.03
1	A	A	K	0.00	9.31	3.00	12.31
1	A	A	L	0.00	9.31	3.00	12.31
1	A	A	M	0.00	9.31	3.00	0.00
1	A	A	N	0.00	9.31	3.00	12.31
1	A	A	O	0.00	9.31	3.00	12.31
1	G	G	H	0.00	9.98	3.00	12.98
1	I	I	J	0.00	21.72	3.00	24.72
1	E	E	F	0.00	5.75	8.00	13.75
1	D	D	E	0.00	10.58	5.75	16.33
1	D	E	F	10.58	5.75	8.00	24.33
1	B	B	C	0.00	20.55	7.00	27.55
1	B	B	D	0.00	2.27	9.50	11.77
1	B	D	E	2.27	10.58	5.75	18.60
1	B	E	F	12.85	5.75	8.00	26.60

Table 6: Product Composition

Itemno	Cost per item	LT	Std	Status	Timeadd
A	8034.76	10.00	6.523637	1	0
B	1707.16	20.52	21.290000	0	1
C	713.09	7.00	36.870000	0	1
D	14.00	9.50	442.170000	0	1
E	96.30	5.75	74.580000	0	1
F	25.00	8.00	350.700000	0	1
G	814.55	9.92	25.150000	0	1
H	282.38	3.00	48.160000	0	1
I	4977.16	21.64	18.800000	0	1
J	1647.11	3.00	39.640000	0	1
K	35.03	3.00	34.820000	0	1
L	49.44	3.00	31.180000	0	1
M	7.72	3.00	30.540000	1	0
N	2.83	3.00	30.780000	0	1
O	2.67	3.00	84.000000	0	0

From the result in product composition Table 6, it's clear that the final product A consists of 14 components. The storage cost per item for each component are listed in the second column. The lead time and standard deviation are also in the third and fourth column. The stock status guideline based on the compulsory stocking restriction of final product A is in the fifth column. It suggests that M and A are stock items while other components are non-stock item which approximate the total minimum storage cost for all components and the final component A. This guideline for separating stock item and non-stock item can be applied to all final products which satisfy the restriction. The total operation time for 1000 iterations is saved within 3 minutes. The application of recursive algorithm to construct the relationship of items in this paper stems from the previous studies. It is innovatively

nested in the function in order to give the product framework table and composition table. what's more, parallel programming, probabilistic sampling, and statistical model are firstly used to reduce the operation time and approximate the global minimum in supply chain optimization. The final function can be applied to products with complicated composition and give the clear result involving both the structure relationship of components and the composition information like storage cost and lead time.

4 Conclusion and Discussion

This paper addresses the safety stock calculation for each item and supply chain optimization of stock status for each final product. By introducing statistical distribution, probability sampling, and statistical functions, the paper successfully tackles the uncertainty of the inventory and generate a guideline to solve the stochastic optimization problem which is defined as the process of minimizing the value of a statistical function with one or more random parameters (cole, 2017). It's beneficial to apply statistical approaches to a large amount of logistics data. Not only the statistical approaches improve the efficiency of data cleaning, but also it accelerates the process of getting desirable result and guideline.

For the safety stock calculation, the paper mainly focuses on stocked items. With the prior information provided by the company, all items can be divided into three categories. One are items that belong to normal distribution, another are items that belong to poison distribution and the rest are items that belong to negative binomial distribution. Considering the influence of lead time and service level, the paper uses three safety stock calculation formulas for three different distributions. Especially for the negative binomial distribution, the probability mass function is abstracted including demand forecast over lead time in case of the occurrence of zero transaction demand in some weeks.

As for determining the guideline to separate stock item and non-stock item, it is apparent that this problem belongs to supply chain optimization. After detecting the influential relationship between stock status and the total storage cost for each final product, the most challenging part lies in the calculation of aggregated lead time. Then, the paper applies the recursive algorithm to describe the effect of product structure in calculating aggregated lead time. The total lead time is the maximum value among all aggregated lead time in this layer compared with the actual lead time of this item. Furthermore, the total lead time has an effect on the safety stock. The amount of safety stock determines the total storage cost of the final product and all components in the framework of this product. In order to get the stock status which can derive the minimum total storage cost, exhaustive search is utilized by listing all possible combination of stock status of each component. Moreover, probability sampling approach Naive and statistical model are applied to shorten the time of getting the global optima together with parallel programming.

Even though the paper has solved two problems successfully, there still exist two facets that need further analysis. It is shown in the result of the safety stock calculation design that the proposed safety stock is sensitive to the value of service level. The determination of suitable service level can be influenced by several factors like the preference, stock-out cost, and the storage cost. In future studies, it will make the safety stock calculation model more efficient if the company find the exact service level value in their studies. The second problem lies in the complexity of the structure of some final products. The technique used to reduce the iteration time only can approximate the global minimum. Sometimes it's unsure if the result is the real global minimum or not. In order to get the optimal global minimum, the algorithm needs further improvement. Binary algorithm can be tried to build the framework of the list of stock status of all items and it may reduce the execution time. What's more, Optimizing the statistical model is another way we can work on.

5 Acknowledge

I am especially grateful for the instruction and support from my supervisor Yukai Yang and the logistical background support by my two supervisors Sebastian Alila and Johanna Dahlman at the company. What's more, I would like to thank my manager at Supply chain department Jonas Jern for giving me this chance to apply what I have learned in my master to make some contributions to the operation of the department.

References

[1] Anupindi, R. Chopra, S. Deshmukh, S. and Zemel, E. (2014). *Managing business process flows: principles of operations management*. PEARSON Prentice Hall, New Jersey, 2nd edn.

[2] Hill, A., Hill, T. (2012). *Operations management*. Palgrave Macmillan, New York.

[3] Mahadevan, B. (2015). *Operations management – theory practice*. Pearson Education, New Delhi, 3rd edn.

[4] Petropoulos, F. Kourentzes, N. 2015. Forecast combinations for intermittent demand. *The Journal of the Operational Research Society*, vol. 66, no. 6, pp. 914-924.

[5] Klosterhalfen, S. Minner, S. 2010. Safety stock optimisation in distribution systems: a comparison of two competing approaches. *International Journal of Logistics Research and Applications*, vol. 13, no. 2, pp. 99-120.

[6] Beutel, A. Minner, S. 2012. Safety stock planning under causal demand forecasting. *International Journal of Production Economics*, vol. 140, no. 2, pp. 637-645.

[7] Buffa, F.P. Munn, J.R. 1989. A Recursive Algorithm for Order Cycle-Time that Minimizes Logistics Cost. *The Journal of the Operational Research Society*, vol. 40, no. 4, pp. 367-377.

[8] Nagurney, A. 2010. Optimal supply chain network design and redesign at minimal total cost and with demand satisfaction. *International Journal of Production Economics*, vol. 128, no. 1, pp. 200-208.

[9] Grigoroudis, E., Petridis, K. Arabatzis, G. 2014. RDEA: A recursive DEA based algorithm for the optimal design of biomass supply chain networks. Renewable Energy, vol. 71, pp. 113-122.

[10] Baud-Lavigne, B., Agard, B. Penz, B. 2016. Simultaneous product family and supply chain design: An optimization approach. International Journal of Production Economics, vol. 174, pp. 111-118.

[11] Syntetos, A.A. Boylan, J.E. 2005. The accuracy of intermittent demand estimates. *International Journal of Forecasting*, vol. 21, no. 2, pp. 303-314.

[12] Pickles, A. (1985). *An Introduction to Likelihood Analysis*. W. H. Hutchins , Norwich.

[13] Rubio-Sanchez, M. (2017). *Introduction to Recursive Programming*. CRC Press, S.l. , 1st edn.

[14] Hutton, G. (2007). *Programming in Haskell*. Cambridge University Press, Cambridge.

[15] Grune, D. Jacobs, C.J.H. (2008). *Parsing techniques: a practical guide*. Springer, New York, 2nd edn.

[16] Pacheco, P.S. (2011). *An introduction to parallel programming*. Morgan Kaufmann, Amsterdam; Boston, 1st edn.

[17] Knaus, J., Porzelius, C., Binder, H., Schwarzer, G. 2009. Easier parallel computing in R with snowfall and sfCluster. *The R Journal*, 1(1), 54-59.

[18] Bertsimas, D., Tsitsiklis, J. 1993. Simulated annealing. *Statistical science*, 8(1), 10-15.

[19] De Vicente, J., Lanchares, J., Hermida, R. 2003. Placement by thermodynamic simulated annealing. *Physics Letters A*, 317(5-6), 415-423.

[20] Cole, B.M. (2014). *Supply chain optimization under uncertainty: supply chain design for optimum performance*, Vernon Press, Wilmington, Delaware.

6 Appendix

6.1 Appendix A: Safety stock calculation code

Due to the confidentical reason, the data cleaning part is not included.

```
get_dis <- function(df_week_clean, compno, wareno){

  ##parameters
  #df_week_clean: weekly data that merge mittra and MMS002 together, filter status = 20
  #compno: company number
  #wareno: warehouse number

  df_loop <- df_week_clean[df_week_clean$MTCONO == compno & df_week_clean$MTWHLO == ware
  loglik <- matrix(0, length(unique(df_loop$key)), 9)
  item <- unique(df_loop$key)



  # check the distributions of items
  # As negtive binomial has POSITIVE AND OTHER restrictions,
  # the code tolerates the errors to carry on the iteration.
  for (i in 1:length(unique(df_loop$key))){
    loglik[i, 1] <- fitdistr(as.vector(df_loop[df_loop$key == item[i], 6]), 'normal')$lo
    loglik[i, 2] <- fitdistr(as.vector(df_loop[df_loop$key == item[i], 6]), 'Poisson')$l
    loglik[i, 3] <- try(fitdistr(as.vector(df_loop[df_loop$key == item[i], 6]), 'negativ
    loglik[i, 4] <- item[i]
    loglik[i, 6] <- df_loop[df_loop$key == item[i], 'leadtime'][1]
    if(sum(df_loop[df_loop$key == item[i], 6] != 0) < 3){
      loglik[i, 7] <- 0
    } else {
      loglik[i, 7] <- crost(data = df_loop[df_loop$key == item[i], 6], h = 1, nop = 2, t
    }
  }

  #set the NA and error values to be negative infinite to exclude them in comparison per
  loglik[is.na(loglik[, 1]) == TRUE, 1] <- -Inf
  loglik[str_detect(loglik[, 3], 'Error') == TRUE, 3] <- -Inf

  #Find the maximum loglik which means the distribution fits the data best
  for (i in 1:length(item)){
    loglik[i, 5] <- which.max(loglik[i, 1:3])
```

```
  }
  df_res <- data.frame(key = loglik[, 4], compno = compno, wareno = wareno,
                       itemno = gsub("^.*_", "", loglik[, 4]), distribution = loglik[, 5
                       EOD = round(as.numeric(loglik[, 7]), digits = 2))
  return(df_res)
}


#item distribution chart
df_dis <- get_dis(df_week_clean, '1', '001')


get_ss <- function(df_week_clean, df_dis, compno, wareno, itemno, percent){

  leadt <- as.numeric(df_dis[df_dis$itemno == itemno, 'leadt'])/7
  EOD <- as.numeric(df_dis[df_dis$itemno == itemno, 'EOD'])
  pre_transaction <- df_week_clean[df_week_clean$MTCONO == compno & df_week_clean$MTWHLO
  pre_trans <- df_week_clean[df_week_clean$MTCONO == compno & df_week_clean$MTWHLO == w
  StdDev <- sqrt(var(as.numeric(pre_transaction))) #TSL, leadt, StdDev, EOD
  TSL <- percent

  if(df_dis[df_dis$itemno == itemno, 'distribution'] == 1){
    fit <- fitdistr(pre_transaction, densfun = 'normal')
    item_mean <- fit$estimate[1]
    item_var <- fit$estimate[2]
    ss <- sqrt(as.numeric(leadt)) * (qnorm(1-percent, mean = item_mean, sd = sqrt(item_v
  } else if(df_dis[df_dis$itemno == itemno, 'distribution'] == 2){
    fit <- fitdistr(pre_transaction, densfun = 'Poisson')
    item_mean <- fit$estimate[1]
    ss <- sqrt(as.numeric(leadt)) * (qpois(1-percent, lambda = item_mean, lower.tail = F
  } else{
    PROBacc <- 0.0
    if(StdDev == 0){
      TSLa <- TSL
      ROP <- 2
    }
    else if(StdDev > 0){
      ROP <- 1
      Z <- (sqrt(leadt) * StdDev) ^ 2 / EOD
```

```
      a <- EOD / (Z - 1)
      B <- (Z - 1) / Z
    }
    while(TRUE){
      d <- ROP + 1.0
      if (d < 3){
        PROBd <- (1.0 / Z) ^ a
        PROBdr <- PROBd
      }
      else if(d == 3){
        PROBdr <- PROBd * (EOD / Z)
      }
      else{
        PROBdr <- PROBdr * B * (a + (d - 2 - 1)) / (d - 2.0)}
      PROBacc <- PROBacc + PROBdr
      d <- d + 1.0
      TSLa <- PROBacc
      ROP <- ROP + 1
      if(TSLa >= TSL){break}}
    ss <- ROP-2
  }
  p.plot <- plot(x = pre_trans$week, y = pre_trans$weeklyqt, type = 'h',  xlim = c(min(c
                 xlab = 'week', ylab = 'weekly transaction demand')
  records <- sum(pre_transaction != 0)
  return(list(ss, records, p.plot))
}

get_ss(df_week_clean, df_dis, '1', '001', 'A', 0.8)
##result
#1.safety stock
#2.number of non-zero transaction demand records
#3.histogram plot of weekly transaction demand vs week


#example for export ideal ss for one warehouse
input <- read_excel('import.xlsx')

ss <- matrix(0, length(input$compno), 1)
```

```
for(i in 1:length(input$compno)){
  ss[i, 1] <- as.numeric(get_ss(df_week_clean, df_dis, as.character(input[i, 1]), as.ch
}

output <- cbind(input, ss)
output
```

6.2 Appendix B: Supply chain optimization on stock status

```
#the final product you want to analyze
prodnum = 'B'

get_table <- function(status){
  #get chain
  item <- book1_fin[book1_fin$PSPRNO == prodnum, ]
  criterion <- item[item$PSPRN1 != prodnum, 'PSPRN1']
  children <- book1_fin[book1_fin$PSPRNO %in% criterion$PSPRN1, ]
  chain <- rbind(item, children)
  rownames(chain) <- c()

  #get product_gui
  #get cost of this product and its components

  material <- tibble(itemno = unique(chain$PSMTNO))
  material[nrow(material)+1, ] <- NA
  product <- material %>%
    mutate(itemno = as.character(lag(itemno)))
  product[1, 1] <- prodnum


  product <- left_join(product, book2_clean, by = c('itemno' = 'M9ITNO'))
  product_1 <- left_join(product, unique(chain[, c('PSMTNO', 'PSLEAL')]), by = c('itemno



  input  <- cbind(compno = '1', wareno = '001', itemno =  unique(book1_fin[book1_fin$PSPR

  #get std
  stdev <- matrix(0, length(input$compno), 1)

  for(i in 1:length(input$compno)){
    stdev[i, 1] <- as.numeric(get_var(df_week_clean, as.character(input[i, 1]), as.chara
  }

  output <- data.frame(itemno = input$PSMTNO,std = round(as.numeric(stdev), digits = 2))
  output[nrow(output)+1, ] <- NA
```

```
output <- output %>%
  mutate(itemno = as.character(lag(itemno)), std = as.numeric(lag(std)))

output[1, 1] <- prodnum
output[1, 2] <- get_var(df_week_clean, '1', '050', prodnum)[1]

product_l_std <- left_join(product_l, output, by = 'itemno')
product_l_std[1, 3] <- as.numeric(get_var(df_week_clean, '1', '050', prodnum)[2])



dt <- matrix(NA , length(product_l_std$itemno), 2)
dt[1, 1] <- 1
dt[2:length(product_l_std$itemno), 1] <- status
dt[, 2] <- 1 - dt[, 1]
product_gui <- cbind(product_l_std, dt)

colnames(product_gui)[5] <- 'status'
colnames(product_gui)[6] <- 'timeadd'

# recursive algorithm
# note, no Agg in chain
sum = with(chain, PSACDB + PSMDBF + PSLEAL)

fAB <- function(iter){

  if(chain$PSPRNO[iter] == chain$PSPRN1[iter]) return(1)

  if(iter<2) stop("row number reaches the upper bound")

  return(fBC(iter, iter-1))

}


fBC <- function(iter, jter){

  if(chain$PSPRN1[iter] == chain$PSMTNO[jter])
    return(fAB(jter) * product_gui[product_gui[,"itemno"] == chain$PSMTNO[jter], "time
```

```
    if(jter<2) stop("row number reaches the upper bound")

    return(fBC(iter, jter-1))

  }

  ftmp <- function(iter){

    return(fAB(iter) * sum[iter] * product_gui[product_gui[,"itemno"] == chain$PSMTNO[it

  }

  #chain <- cbind(chain, agg = sapply(1:length(chain$PSPRNO), ftmp))
  chain$agg = sapply(1:length(chain$PSPRNO), ftmp)


  return(list(chain, product_gui))
}


#function used to calculate the total storage cost for different stock status
get_guideline <- function(num){
  tmp_sub <- t(tmp[num,])
  ret <- get_table(tmp_sub)
  chain <- ret[[1]]
  product <- ret[[2]]

  product$totalLT <- NA

  for(i in 1:nrow(product)){
    tmpp <- chain[chain$PSPRNO == product[i, 'itemno'], 'agg']
    if(nrow(tmpp) == 0) {tmpp <- 0}
    product[i, 7] <- max(tmpp, product$PSLEAL[i])
  }

  product$ss_level <- 0.85
  product$ss <- qnorm(product$ss_level)*sqrt(product$totalLT)/7*product$std*product$stat
  product$cost <- round(product$M9UCOS * product$ss, digits = 2)
```

```
  sum_cost <- sum(product$cost, na.rm = TRUE)
  return(sum_cost)
}

len <- nrow(book1_fin[book1_fin$PSPRNO == prodnum,])

tmp <- paste("expand.grid(", paste(rep("c(0,1)",len), collapse = ', '),")")

tmp <- eval(parse(text=tmp))

#exhaustive search (can be applied to those with less than or equal to 10 components)

min_indicator <- which.min(sapply(1:nrow(tmp),get_guideline))
table1 <- as.data.frame(get_table(t(tmp[min_indicator, ]))[1])
table2 <- as.data.frame(get_table(t(tmp[min_indicator, ]))[2])

################################################
####naive_sampling(if components exceed 10)####
################################################

set.seed(10)
icomp <- len # how many components
iN <- 2^icomp
ih <- 2000
subsamp <- sample(1:iN, ih)

################################
#####parallel programming######
################################

sfInit(parallel=TRUE, cpus=7)
sfLibrary(tibble)
sfLibrary(magrittr)
sfLibrary(dplyr)
sfExportAll()
ptm <- proc.time()
ret <- sfLapply(subsamp, get_guideline)
proc.time() - ptm
sfStop()
```

```
res <- which.min(ret)
get_table(t(tmp[subsamp[res], ]))

table1 <- as.data.frame(get_table(t(tmp[subsamp[res], ]))[1])
table2 <- as.data.frame(get_table(t(tmp[subsamp[res], ]))[2])



#########################
####statistical model####
#########################

get_neighbour <- function(point){
  ipos = sample(1:icomp, 1)
  if(point[ipos] == 0) point[ipos] = 1
  else point[ipos] = 0
  return(point)
}

move <- function(start){

  current = tmp[start, ]
  ctemp = get_guideline(start)
  point = get_neighbour(current)

  get_row <- function(row){
    if(all(tmp[row, ] == point) == TRUE)
    {return(row)}else{return(0)}
  }

  row <- lapply(1: nrow(tmp), get_row)
  point_n <- as.numeric(row[row!=0])
  ptemp = get_guideline(point_n)

  if (ptemp < ctemp){
    return (point_n)
  }else{return(start)}
}
```

```
ret2  <- sample(1:iN, 1)
step_max <- 100
iter <- 1

while(iter < step_max){
      print(ret2)
      ret2 <- move(ret2)
      iter <- iter+1
   }

res2 <- last(ret2)
get_table(t(tmp[subsamp[res2], ]))

table2.1 <- as.data.frame(get_table(t(tmp[subsamp[res2], ]))[1])
table2.2 <- as.data.frame(get_table(t(tmp[subsamp[res2], ]))[2])
```