ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics
Bachelor Thesis Econometrics and Operations Research

The Impact on Stock-Outs of Incorporating Drones in the Supply Chain of Essential Drugs in Zambia

Name student: Ingrid Pool
Student ID number: 432357
Supervisor: Rijn, L. van
Second assessor: prof.dr. A.P.M. Wagelmans
Date: July 05, 2019

Abstract

This research aims to replicate results obtained in a paper by Leung et al. concerning stock-outs of essential medicines in Zambia and the effect of several inventory policies on these stock-outs. The second goal of this study is to investigate the benefits of incorporating drones in the distribution system of these medicines to lower stock-outs. A simulation model is used as a method to perform the replication. This simulation model is then extended to incorporate drones. This research concludes that recommended inventory policies cause stock-outs by not properly accounting for demand seasonality and lead time problems. It is shown that a drone system in sub-Saharan countries is feasible and can reduce stock-outs and inventory levels at health clinics. However, this research reveals several complicating factors regarding the implementation of a drone system that need to be taken into consideration when deciding on whether a drone system is the best way to lower stock-outs of essential medicines.

Contents

1 Introduction

The (lack of) availability of medicines in sub-Saharan Africa received substantial attention over the last decades, because the unavailability of essential medicines can lead to preventable deaths (Cameron et al., 2009; Pasquet et al., 2010). Lower respiratory tract infections, HIV/Aids, diarrhoeal diseases, malaria, and tuberculosis, which all are preventable and treatable given adequate healthcare systems and resources, together accounted for more 3.2 than million deaths (35.1% of all deaths) in sub-Saharan Africa in 2016 (World Health Organization, Geneva, 2018).

Several causes of drug shortages have been addressed and researched. The first cause being insufficient procurement financing and processes. This is critical to acquiring sufficient medicines in the country in the first place. But even if there are enough resources within the country, frequent stock-outs of medicines in local health clinics remain a problem (Vledder et al., 2015a). A main contributor to local unavailability of medicines is the distribution within countries.

Recently, multiple studies addressed several factors of the supply chain to determine ways to improve the availability of essential drugs and to lower stock-outs. Researched factors include financing, procurement, staff training, distribution capacity, information systems and resupply frequency (Kangwana et al., 2009; Waako et al., 2009; Yadav, 2015). Specifically, Leung et al. (2016) researched the impact of inventory management policies as recommended by the USAID | DELIVER project (USAID | DELIVER PROJECT, 2011a,b; Watson et al., 2014).

The study by Leung et al. (2016) is the base of this research and the first part of this paper is dedicated to replicating their results. Their research studies the stock-outs of anti-malarial drugs (arthemether-lumefantrine) in Zambia. These are essential drugs which need to be distributed to all health clinics and demand depends on the time of the year. In Zambia drugs are first distributed from the national warehouse to the districts and then from districts to local health clinics. The order quantities are determined by a max-min inventory policy and orders are placed by the health clinics.

Leung et al. construct a simulation model to simulate demand, issues and deliveries in a health clinic. This simulation model allows them to study inventory levels and stock-outs under different inventory management policies recommended by USAID | DELIVER (USAID | DELIVER PROJECT, 2011a,b). Using the simulation model they also conduct sensitivity analysis regarding demand seasonality and delivery lead times. Leung et al. conclude that stock-outs of essential medicines occurred in Zambia's health clinics, while these products were available in the national warehouse. Therefore, they attribute these stock-outs to the max-min inventory control policy used in Zambia.

More specifically, they find three root causes of stock-outs with this policy. First, the demand forecasts used fail to capture demand seasonality. Ignorance of predictable changes in delivery lead times over time is identified as a second key driver of stock-outs. Lastly, the replenishment targets are computed with past consumption data instead of demand data. From sensitivity analysis Leung et al. find that simple changes in parameter values (degree of seasonality; lead times) or calculation methods (using historical

issues or demand; multiplication factor for estimated demand to calculate order quantity) are limited in improving max-min inventory policies. Increasing the multiplication factor used in calculating the replenishment target can improve service levels, but also significantly increases average and maximum inventory levels. If these levels are even feasible, they are inefficient and costly.

Instead of altering the max-min inventory policy another approach to reducing stock-outs is to improve the factors that cause stock-outs with the current inventory policy. As concluded by Leung et al. one of these factors is the varying delivery lead times. The other factor is demand seasonality, which is also related to lead times because of the following. The order quantity, which is already badly in tune with actual demand because it insufficiently accounts for demand seasonality, is delivered later than the period for which the order quantity was computed. The majority of the lead time consists of the time it takes to transport medicines from district warehouses to local health clinics. The medicines are mainly transported by trucks. In many sub-Saharan countries trucks and roads are in poor condition and unfavourable weather conditions cause blocked roads. As a consequence, lead times are long and highly uncertain. Regarding delivery lead times in sub-Saharan Africa, several solutions have been proposed. These include a vehicle maintenance project proposed by the Riders for Health, reorganization of the distribution system, improving road infrastructure and improving communication (Bossert et al., 2007; Chen et al., 2016; Vledder et al., 2015b; Yadav and Babaley, 2011).

In this research an alternative solution will be examined, namely the incorporation of drones in the supply chain of essential drugs. Drones have several potential benefits regarding the distribution of drugs in sub-Saharan countries. Firstly, drones can reduce lead times of regional deliveries. Consequently, a placed order is received quicker compared to truck delivery. Secondly, the variability in lead times will be much lower compared to trucks. Drones can fly even if it rains and are not delayed by blocked roads through for example unfavourable weather conditions. Lastly, the fact that lead times are both shorter and more constant gives the opportunity to adjust the inventory policy in a way that allows for more frequent deliveries. This might enable lower inventory levels at the health clinics.

There are of course also a few possible downsides to the incorporation of drones. The first point of caution is the limited range and cargo of the drones. Secondly, drone stations have to be installed and people have to be trained, which can be costly and time-consuming. Thirdly, drones have significant costs per flight.

This research aims to determine the extent of the potential benefits by characterizing the impact of supply strategies that incorporate drones on stock-outs of essential drugs in Zambia's health clinics. Several studies have researched different applications of drones for medical purposes. Tatham et al. (2017) find no insuperable challenges regarding the use of long endurance remotely piloted aircraft systems to support the provision of medical supplies to remote locations. Before that, Thiels et al. (2015) already suggested that the use of unmanned aerial vehicles could be a viable mode for the transport of medical products in times of critical shortage. In addition, there are other studies and projects investigating the use of drones which take place in sub-Saharan African countries

(Haidari et al., 2016; Zipline). These examples show that it is possible to implement these high-technology devices in low or middle income countries.

Zipline already uses drones daily in Rwanda and Ghana for the delivery of vital products such as blood and vaccines (Zipline). When a medicine is needed, orders can be placed through an app. The order then is packed and directly shipped with the battery-powered drone within approximately 10 minutes from receiving the order. When the drone arrives at its destination within 45 minutes it drops the package with a parachute. The whole process from placing to receiving the order takes less than an hour. When the drones return they are quickly prepared for the next flight, which makes it possible to deliver hundreds of orders per day from each distribution center in all weather conditions.

The main focus of this paper is to replicate the results obtained by Leung et al. (2016). To achieve this, data published by Leung et al. will be used and a simulation model is constructed. The second objective of this research is to characterize the impact of incorporating drones in the supply strategies on stock-outs of essential drugs and to analyze the effectiveness of this approach. Results of this study confirm the conclusions stated by Leung et al.. Furthermore, it shows that incorporating drones in the distribution of essential medicines is feasible and that it can reduce stock-outs while at the same time reducing inventory levels at health clinics.

The rest of this paper is organized as follows. The paper will start with a description of the data in Section 2. The first part of this section elaborates on the data needed for the original analysis. The second part shows the additional data needed to investigate the incorporation of drones. In Section 3 the methods used to replicate the results from Leung et al. (2016) and extend this replication with drones will be explained. Furthermore, the results of this research are stated in Section 4. Section 5 contains a discussion of the results and in Section 6 final conclusions about this research are stated.

2 Data

To compare the research results, the same products will be investigated as in the study by Leung et al. (2016). These products concern the four dosage forms (strips of 6, 12, 18 and 24 tablets) of artemether-lumefantrine (AL), an anti-malarial drug. Data regarding these drugs is available over a one year period. Herein, a month is approximated by 4 weeks and consequently a year is approximated by a 48 week period.

2.1 Replication

Stock-levels of medicines in Zambia's health clinics were reported by commodity planners on stock cards. To gather the data, photographs of stock-cards were collected between June 2009 and June 2010. Information on the stock card of a product consists of dates and quantities of issues, deliveries and stock counts. This information is included in the dataset for 11 districts and 152 health clinics.

The dataset also includes estimates of average weekly demand for a typical clinic.

Leung et al. (2016) adjusted the raw daily demand rates to correct for censored data. After summing all four dosage forms of AL and averaging over 17 clinics from 5 different districts they obtained an estimated average daily demand in a typical clinic (all four dosage forms combined), from which the weekly averages can be derived. Leung et al. conclude that weekly demand is best described by a lognormal distribution with coefficient of variation equal to 50% and mean equal to the estimated average weekly demand. These average weekly demand estimates are given in Table A1 in the Appendix. Concerning demand values, Leung et al. define the seasonality factor as 'ratio of peak demand mean to average mean through the year' and report a seasonality factor of 2.2 obtained from the data.

Distribution of medicines to local clinics in Zambia is divided into two stages. In the primary stage the products are distributed from the national warehouse to the districts. In the secondary stage, the products are further distributed from the district stores to health clinics. The lead time of the primary distribution is assumed to be 2 weeks, because it has a fixed monthly schedule and is reliable. The dataset contains a table with delivery dates per district as well as delivery dates per health clinic. This historical data can be used to estimate the secondary delivery times. These secondary delivery times also include delay due to (weather-related) access problems. Clinic accessibility probabilities per month are estimated by Leung et al. (2016) by averaging the weekly subjective probabilities surveyed from clinic staff. The resulting monthly estimated probabilities from January to December are 0.78, 0.76, 0.78, 0.85, 0.93, 0.97, 0.97, 0.98, 0.99, 0.99, 0.95 and 0.85. Leung et al. model the secondary lead times without access problems as a geometric random variable. The mean is estimated from historical data and is 3.8 weeks. This value is used as the first moment of the geometric random variable.

2.2 Drone Statistics

Data regarding the drones will be estimated by assuming the use of the Zipline drone (Zipline). The newly introduced Zipline drones have a cruising speed of more than 100 kilometers per hour, can reach a distance of 160 kilometers (80 kilometers up and 80 kilometers back), and can carry up to 1.75 kilograms of cargo (3 units of blood) (Zipline). The shipping box is estimated to have a volume of $10 \times 20 \times 30 = 6000$ cm^3. The 160 kilometer range means that one drone can cover an area of approximately 20,106 km^2 within an hour. The average district size in Zambia is 10,170 km^2 with a variance of 7,828 km^2 (2012). This means that with approximately 1 drone station per district (and if strategically placed possibly less) all healthcare centers can be reached. Zipline is a commercial company that charges per delivery, and does not charge any costs for setting up drone stations. The cost of a delivery is influenced by product weight as well as urgency and distance of the delivery. Depending on these factors 15$ to 45$ is charged per delivery (Wakefield, 2017). For example, Zipline will install four drone stations in Ghana. With Ghana's government, they agreed upon 600 deliveries per day for a four year period. Zipline will be paid per successful delivery, which means that Ghana will pay Zipline a total of about $12.5 million for their services (Asiedu, 2019).

A package of Coartem 24 (AL product with 24 tablets) is estimated to be 15 grams

with a volume of $11 \times 2 \times 4 = 88$ cm^3. Based on the maximum cargo weight, a Zipline drone can carry up to 116 packages of Coartem 24. The volume of the packages, however, restricts the maximum number of Coartem 24 packages carried by a single drone to $\frac{6000}{88} \approx 68$ Coartem 24 packages, which is equal to 68 adult regimens.

3 Methods

3.1 Simulation

To investigate the stock-outs of AL products and the impact of the incorporation of drones, a discrete-event simulation model will be used. Simulation is a convenient tool to investigate differences in performance of a process when altering certain components of the process as it allows the user to make adjustments and re-simulate the process multiple times. In this simulation, the stock-levels of AL-products of a typical health clinic will be simulated. This entails the simulation of replenishment orders, demand, and order deliveries, while keeping track of stock-levels and outstanding replenishment orders.

The simulation has time increments of a week. For each week, the following sequence of steps is taken.

1. Replenishment orders to be received that week are processed, that is, the ordered quantity is added to the inventory and the outstanding order quantity is updated;

2. A new order is placed if this week is the first week of the month. This means that an order quantity is calculated according to the active inventory policy. The average demand over the specified historical months is calculated and multiplied by the multiplication factor. From this, the current and outstanding inventory is subtracted, to obtain the order quantity. To find the delivery week of the order, a random lead time is generated from the assumed primary distribution, geometric secondary distribution and delay probabilities as described above in Section 2.1;

3. The last step is to process demand. Demand is generated from a lognormal distribution as described in Section 2.1 with the first moment equal to the average weekly demand estimate for this week. This demand is then covered by issues for as much as available inventory allows, and inventory is updated.

The key performance measure is service level, which is defined as the fraction of satisfied demand over the total demand. This means that the service level is directly related to the number and duration of stock-outs, since during a stock-out no demand can be satisfied. To minimize the impact of the initial state of the simulation on the performance measures, a warm-up period of two years is chosen. Over the three simulated years after the warm-up period, data is collected to compute performance measures. Moreover, the simulation contains several random variables. In order to obtain consistent results, each simulation includes 100,000 replications, from which the collected data is averaged.

First, this simulation will be implemented with the inventory policies as recommended by USAID | DELIVER PROJECT (2011a,b) (for Java code see Appendix B). Inventory policies are described following a specific notation. General notation is as follows: a×X[b,c]. The first number, a, represents the multiplication factor. X∈{I, D} determines whether the historical average of issues (I) or demand (D) is used in calculating the replenishment target The interval [b,c] shows over which historical months to calculate average issues/demand, and must be interpreted in relation to the current month. Sensitivity analysis will be conducted for lead times and the seasonality factor. Results from the inventory policy analysis as well as the sensitivity analysis will be compared with the reported results by Leung et al. (2016).

3.2 Extension Incorporating Drones

For the second part of this research, the simulation will be extended to incorporate drones (Java code is included in Appendix B). A drone delivery takes less than an hour, so in the simulation model this will be approximated by immediate delivery after the order is placed. At the end of any week, after demand of this week has been processed, a drone order can be placed if the inventory level is below a certain value and the maximum frequency of drone usage (per week or month) has not yet been reached. The amount of the drone order is the minimum of the needed medicines and maximum cargo. If the maximum drone frequency allows for another drone order this week and the amount of medicines needed cannot be carried by one drone, extra drone orders can be placed until maximum drone frequency is reached or all needed medicines are ordered. The total ordered amount is directly added to the inventory, so that it is available at the beginning of the next week. Drone deliveries are independent from truck deliveries, which still follow the same schedule.

To find the most effective way to incorporate drones a sensitivity analysis will be used. This sensitivity analysis will entail allowed frequency of use, maximum transport capacity and the inventory level value from below which placing a drone order is allowed (critical inventory level). The simulation will then be used to obtain performance measures for different parameter values. For all methods in the extension, the standard inventory policy is applied. More specifically, all results are obtained with simulation of the 4×I[-3,0] inventory policy. Based on the results, a few other inventory policies are selected to simulate with the incorporation of drones. These inventory policies are selected based on highest service level and lowest inventory levels.

4 Results

4.1 Replication

As a first step, the emergence of stock-outs is examined. Figure 1 shows how seasonality in weekly demand values causes stock-outs of AL products for the 4×I[-3,0] policy, which is the policy currently used in Zambia's health clinics. Demand rises starting from December through January and February and reaches its peak in April and May.

The order quantities at the moment of placing the order are approximately one month behind, starting to rise in January and peaking in April. Because of the long lead time, inventory levels only start to rise again from April onwards. This means that clinics suffer from increased stock-outs around March and April, which corresponds to the observed increased stock-outs in Q1 and Q2 of 2010.

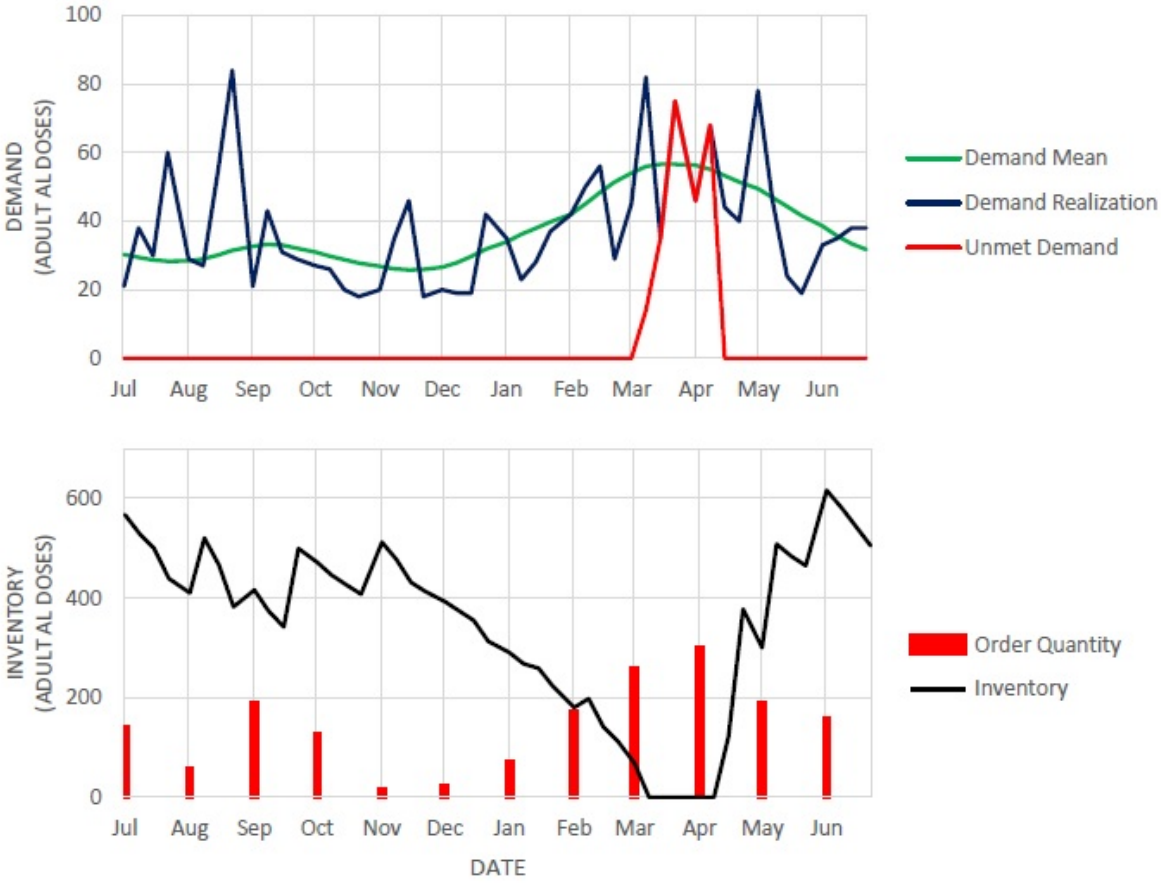

Figure 1: Random simulated sample path of the 4×I[-3,0] policy with a warm-up period of 100 years. The top graph shows the mean estimated demand, actual generated demand and unmet demand for all AL products over a period of a year, starting in July. The bottom graph visualizes the order quantities and inventory levels over the same period.

Average service level for the 4×I[-3,0] policy is approximately 93%. This is slightly higher than the 88% reported by Leung et al. (2016) Furthermore, they report average and maximum yearly inventory levels of 2.0 and 4.3 months of demand, respectively. Results from the simulation in this research resulted in 1.9 months of demand for the average yearly inventory level and 4.0 months of demand for the maximum yearly inventory level. This means that on average the inventory levels are slightly lower than the average inventory levels simulated by Leung et al.

As reported in Section 2.1, Leung et al. find a seasonality factor of 2.2. The weekly average demand estimates as reported in Table A1, however, contain a seasonality factor of 1.51, so this value is considered the 'base' in this paper. Accordingly, the seasonality factors used for sensitivity analysis by Leung et al. are adjusted to match the alternative 'base seasonality factor' found here. The considered values for the seasonality factor in this paper range from 1.0 to 2.02. For the 4×I[-3,0] policy, lowering the mean secondary lead time from 3.8 weeks to 2.2 weeks, increases service level, average inventory level and maximum inventory level to 97%, 2.3 months and 4.5 months respectively (Figures 2A, 2C and 2E). On the other hand, increasing mean secondary lead time with approximately one week to 4.87 weeks decreases the simulated annual service level to 89%. This 4% decrease is approximately of the same order of magnitude as the decrease from 88% to 83% as obtained by Leung et al. when increasing mean secondary lead time with one week. Considering demand seasonality, service level decreases at a faster than linear rate, when demand seasonality increases (Figure 2B). Maximum inventory level slightly increases with demand seasonality for the 4×I[-3,0] policy, while average inventory level stays approximately the same (Figures 2D and 2F). Concluding, demand seasonality and lead times have substantial influence on the service level for the 4×I[-3,0] policy, but do not impact average and maximum inventory levels very much. Both mean secondary lead time and demand seasonality have less effect on the service level for higher multiplication factors, since the higher multiplication factors, and thus higher inventory levels, create a buffer for demand fluctuation or delivery delay. However, the impact on average inventory level and maximum inventory level is larger.

In the current situation (mean secondary lead time of 3.8 weeks and demand seasonality of 1.51), the 4×D[-12,-9] policy performs best in terms of service level out of all inventory policies recommended by USAID | DELIVER PROJECT (2011a), with a score of 98% (Figures 3A and 3B). Compared to the other inventory policies this policy leads to a reasonable level of average and maximum inventory, namely 2.0 and 4.0 months respectively (Figures 3C- 3F). Using historical demand instead of historical issues to compute replenishment targets increases service level for all policies, but the improvements are small. Changing from the 4×I[-3,0] policy to 4×D[-3,0] only increases service level with 0.6% (Figure 3A and 3B). All together, reorganizing inventory policies improves service levels with maximum 5% and decreases service levels with at most less than a percent (Figures 3A and 3B). Adjusting the historical monthly issues used in the inventory policy decreases average inventory level with less than 0.1 months for 4×I[-6,0], 4×I[-12,0] and 4×I[-12,-9]. The 4×I[-1,0] policy results in an increase of average inventory level from 1.9 months to 2.4 months (Figures 3C and 3D). Similarly, for the policies 4×I[-6,0], 4×I[-12,0] and 4×I[-12,-9] maximum inventory levels decrease from 4.0 months to 3.5, 3.2 and 3.9 months respectively, while 4×I[-1,0] results in a higher maximum inventory level of 5.2 months (Figures 3E and 3F). All these results are comparable to the results obtained by Leung et al. (2016). This means that recommended inventory policies have limited impact on the service level and that policies with higher service levels also result in higher inventory levels.

Altering demand seasonality and mean secondary lead time seems to have the most

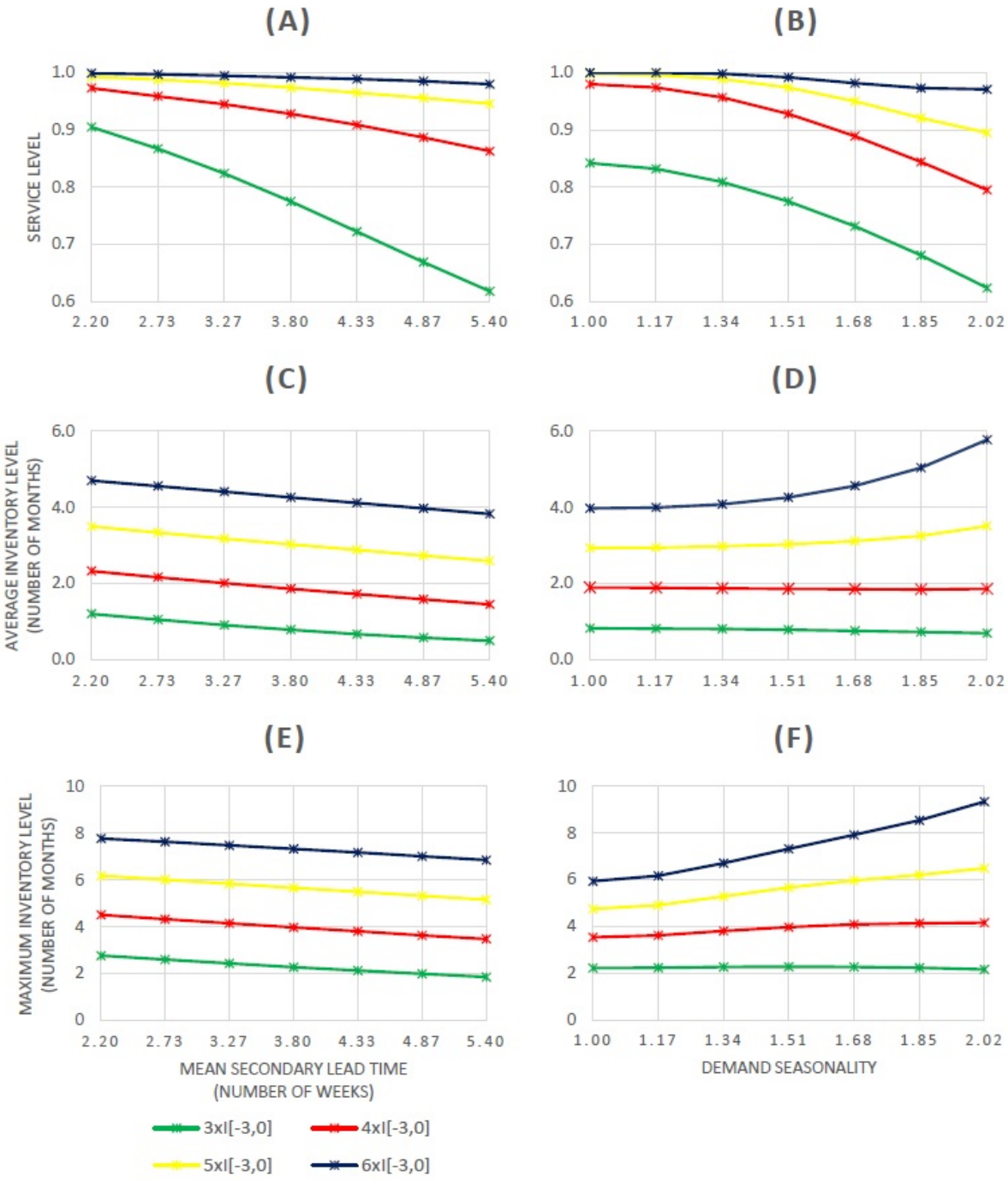

Figure 2: Results of sensitivity analysis for the I[-3,0] policies with multiplication factors ranging from 3 to 6. Subfigures A, C and E show the effect of changing the mean secondary lead time on service level, average inventory level and maximum inventory level respectively for a fixed demand seasonality of 1.51. Subfigures B, D and F show the effect of altering the demand seasonality on service level, average inventory level and maximum inventory level respectively for a fixed mean secondary lead time of 3.8.

impact on inventory policies that use the previous three or six months to calculate replenishment rates. Since these policies do not use demand or issues from the same month a year ago to estimate future demand, they perform worse when demand seasonality increases. Especially, increasing demand seasonality from 1.51 to 2.02 decreases service level with 8%-15% to levels below 85% or even 80%. In contrast, increasing demand seasonality has basically no impact on the performance of 4×I[-12,-9] and 4×D[-12,-9] (Figures 3A and 3B).

4.2 Extension

When the allowed frequency of drone usage changes from none to the minimum frequency of once a month, service level increases from 92.8% to 96.5% for the 4×I[-3,0] inventory policy (Figure 4A). Assuming maximum drone capacity of 68 adult regimens and using a critical inventory level of 0, unmet demand decreases with more than 50% for a maximum of 1 drone delivery per month compared to no drone usage. Even though a frequency of once a month is allowed, on average only 1.4 drone deliveries per year are used, with a total of on average 91.4 adult regimens delivered by drone per year (Figures 4C and 4E). Allowing for a maximum frequency higher than four drone deliveries per month or two per week does not result in higher service levels nor increases the average number of drone deliveries per year or the average number of medicine doses delivered by drone per year (Figures 4A, 4C and 4E). Operating a critical inventory level of 0 adult regimens thus seems to limit the service level to 97.7%, even for unlimited drone usage (Figure 4B).

Service level can however be increased further when the critical inventory level is raised. Allowing for drone deliveries from an inventory level below 40, which is close to average weekly demand, instead of operating a critical inventory level of 0, increases service level from 97.7% to 99.2% when there is no restriction on the frequency of drone deliveries (Figure 4B). For critical inventory levels of 40 or higher the difference between service level between unrestricted and restricted drone usage is negligible. However, the average number of drone deliveries per year increases with between 0.98 and 1.26 deliveries for unrestricted drone usage in the range of critical inventory levels between 0 and 60 adult regimens (Figure 4D). Despite this, the average number of products delivered by drone per year with restricted drone usage approaches the numbers obtained with unrestricted drone usage when critical inventory level increases, similarly to service levels (Figure 4F).

Apart from the rise in service level, drone usage also increases average and maximum inventory levels (Figures 5A- 5D). A maximum drone delivery frequency of 2 per week or 4 per month increases average inventory level from 1.9 to 2.1 months of demand. Also, these frequencies increase the maximum inventory level from 4.0 to 4.3 months of demand. Consistent with the stagnated increase of drone deliveries and delivered products by drone in Figures 4C and 4E, allowance for more frequent drone usage does not influence inventory levels (Figures 5A and 5C). Increased critical inventory levels lead to higher inventory levels with little difference between restricted and unrestricted drone usage. A critical inventory level of 40 adult regimens puts average inventory levels to 2.2

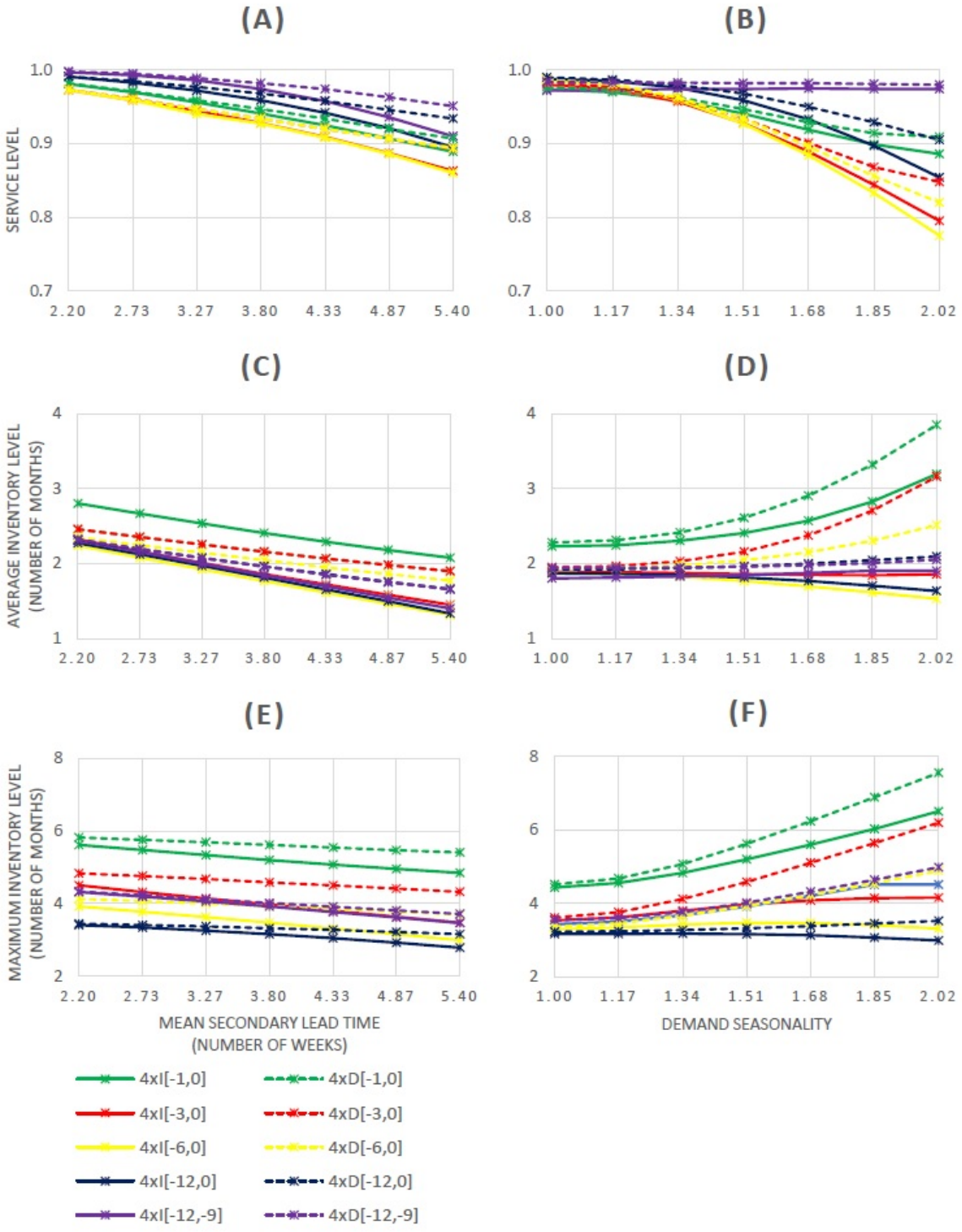

Figure 3: Results of sensitivity analysis for all historical policies with multiplication factor 4. Subfigures A, C and E show the effect of changing the mean secondary lead time on service level, average inventory level and maximum inventory level respectively for a fixed demand seasonality of 1.51. Subfigures B, D and F show the effect of altering the demand seasonality on service level, average inventory level and maximum inventory level respectively for a fixed mean secondary lead time of 3.8

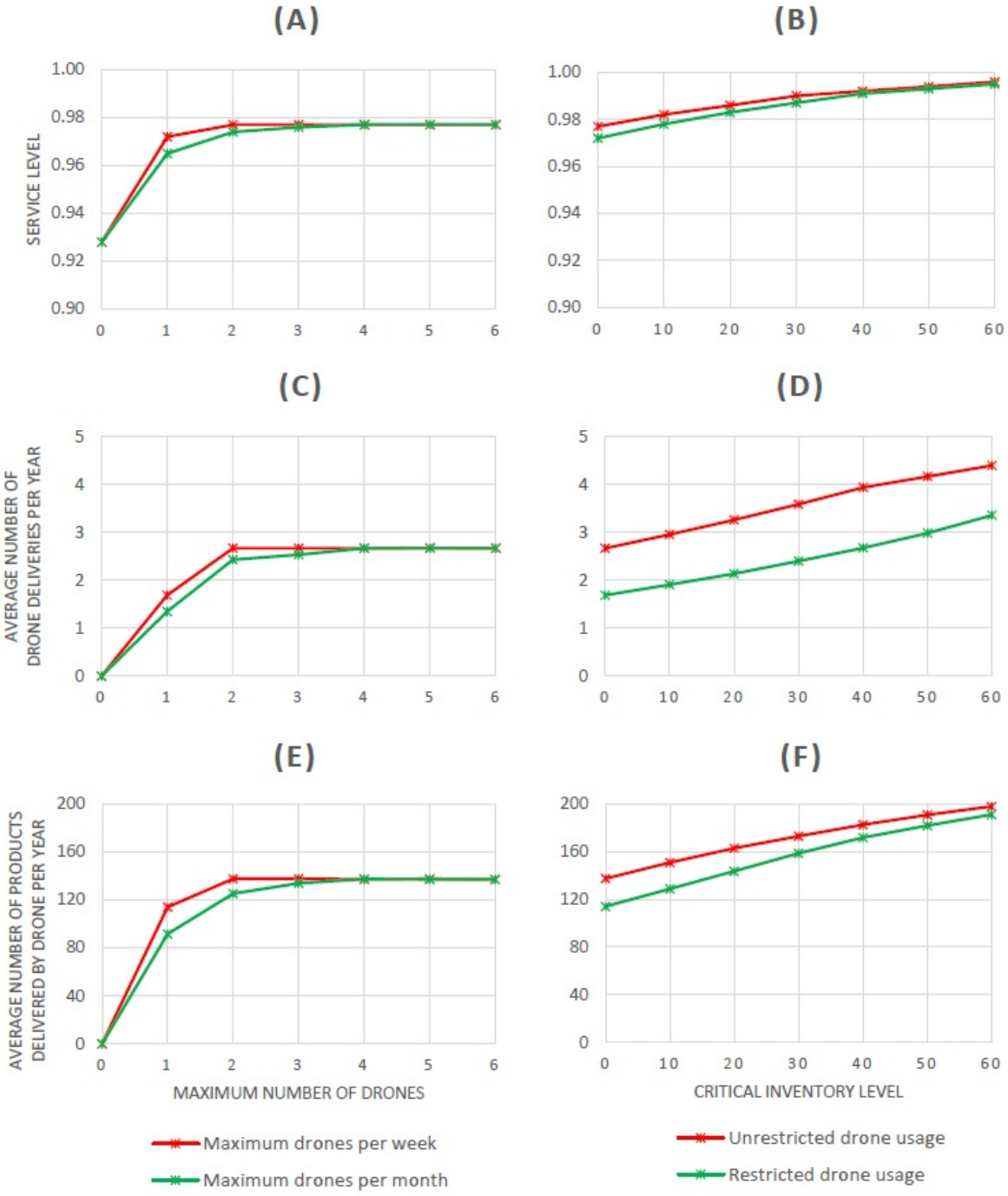

Figure 4: Results for the 4×I[-3,0] inventory policy with standard seasonality and lead time values, when it is possible to use drones for medicine transportation. The graph on the top left shows the effect of the maximum number of drone usages per week/month per health clinic on service level. In the graph on the top right, the service level is shown for different values for the critical inventory level. The four graphs below state the corresponding average number of drone deliveries per year and the average number of products delivered by drone per year. All results are obtained with a drone capacity of 68 adult regimens. For the results in the graphs on the left a critical inventory level of 0 is used. For the restricted drone usage the maximum is set to 1 drone per week and 4 drones per month.

months of demand and maximum inventory levels reach 4.5 months of demand (Figures 5B and 5D). This is an increase of only 0.1 and 0.2 months of demand respectively compared to a critical inventory level of 0.

For a maximum drone delivery frequency of 1 per week and 4 per month, higher drone capacity leads to on average less drone deliveries, but more products delivered by drone per year (Figures 6D and 6E). Consequently, service level increases with drone capacity, as do average and maximum inventory levels (Figures 6A-6C). Specifically, decreasing drone capacity to 38 adult regimens per drone decreases service level with 0.8%, average and maximum inventory levels with 0.1 months of demand, average number of products delivered per year with 28.7 adult regimens, and increases the number of drone deliveries with 0.55 per year. The other way around, increasing drone capacity to 98 increases service level with 0.4%, average and maximum inventory levels with less than 0.1 months of demand, average number of products delivered per year with 19.6 adult regimens, and decreases the number of drone deliveries with 0.25 per year (Figure 6).

The results in Section 4.1 show that the 4×D[-12,-9] inventory policy scores highest in terms of service level (98.2%), with average and maximum inventory levels of 2.0 and 4.0 months of demand respectively. The inventory policy with lowest inventory level is 3×I[-1,0] with a service level of 77.5% and average and maximum inventory levels of 0.8 and 2.3 months of demand respectively. In Figure 7 the results of the simulation are shown when including drones in these two inventory policies. In this figure the results of including drones in a combination of the two policies, that is 3×D[-12,-9], is included as well. For a complete comparison the last inventory policy included in Figure 7 is 3×D[-12,-9] without the use of drones. In these simulations the standard drone capacity of 68 adult regimens is applied. Drone usage frequency is restricted to avoid excessive use of drones. The results in Section 4.1 show that a frequency higher than two drone deliveries per week does not significantly increase service levels any further, so therefore a maximum drone delivery frequency of two per week is chosen.

The service level of the 4×D[-9,-12] inventory policy was already high without drones (Figure 3). Including drones further increases service level from 98.2% to 99.3-99.9% depending on the critical inventory level (Figure 7A). This is accompanied with an almost negligible increase in average and monthly inventory levels of less than 0.1 months (Figures 7B and 7C). The average number of drone deliveries and delivered products by drone per year range from 0.84 to 2.2 and from 42.50 to 73.08 respectively (Figures 7D and 7E). Including drones in the 3×I[-3,0] inventory policy increases service level with more than 17% to 95.1-99.2% (Figures 3 and 7A). This substantial increase is induced by a high number of drone deliveries and products delivered by drone per year (Figures 7D and 7E). Even though average and maximum inventory levels increase compared to this inventory policy without drones, they remain relatively low with levels of 1.24-1.39 and 2.97-3.18 months of demand respectively (Figures 7B and 7C). The 3×D[-12,-9] inventory policy has a multiplication factor of 3, which resulted in low inventory levels for the 3×I[-3,0] inventory policy, and uses historical demand of the next three months as a demand forecast, which resulted in high service levels for the 4×D[-9,-12] inventory policy. This results in a policy with a service level in between those of the 3×I[-3,0] and

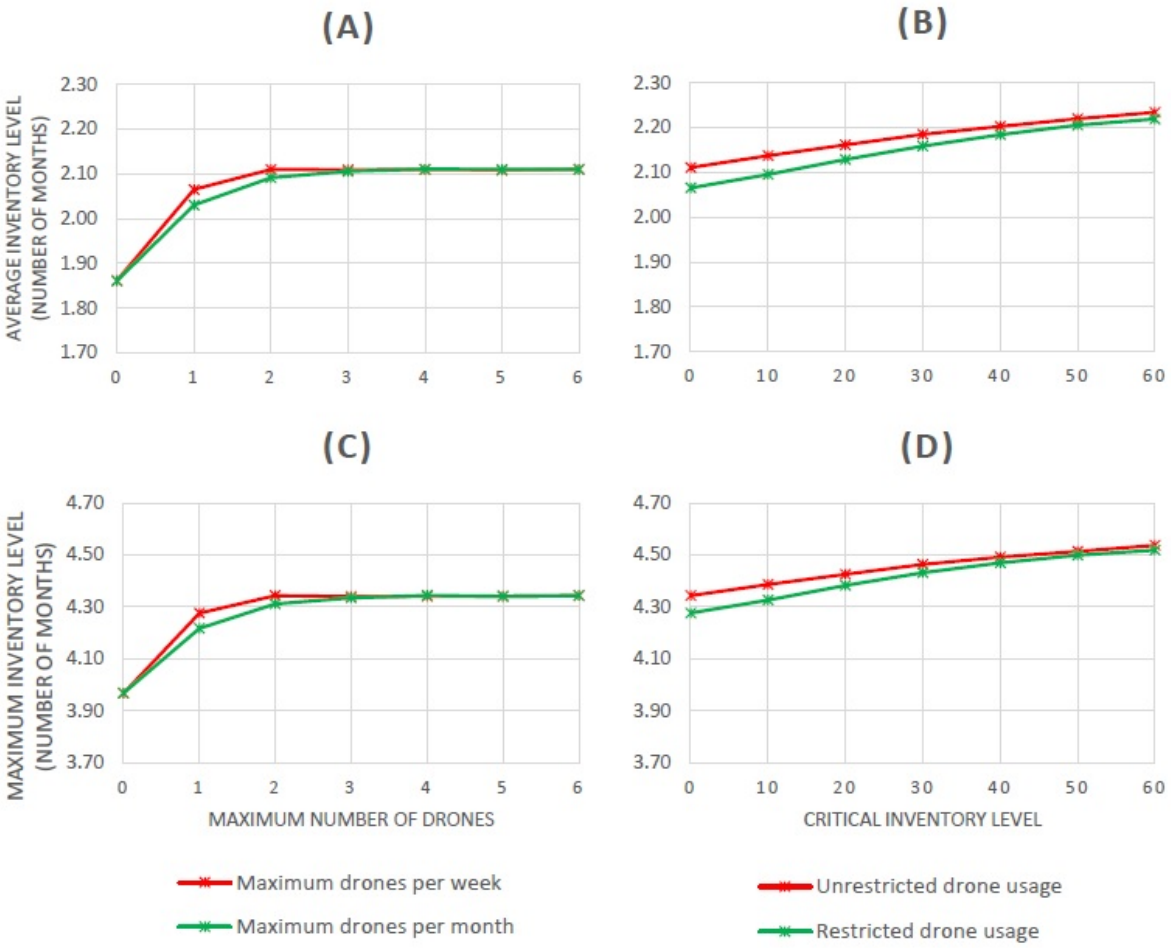

Figure 5: Results for the 4×I[-3,0] inventory policy with standard seasonality and lead time values, when it is possible to use drones for medicine transportation. The two graphs on the left show the effect of the maximum number of drone usages per week/month per health clinic on average and maximum inventory level in terms of months of demand. These results are obtained with a drone capacity of 68 adult regimens and a critical inventory level of 0. The graphs on the right state average and maximum inventory levels for different values for the critical inventory level and a drone capacity of 68 adult regimens. For the restricted drone usage the maximum is set to 1 per week and 4 per month.

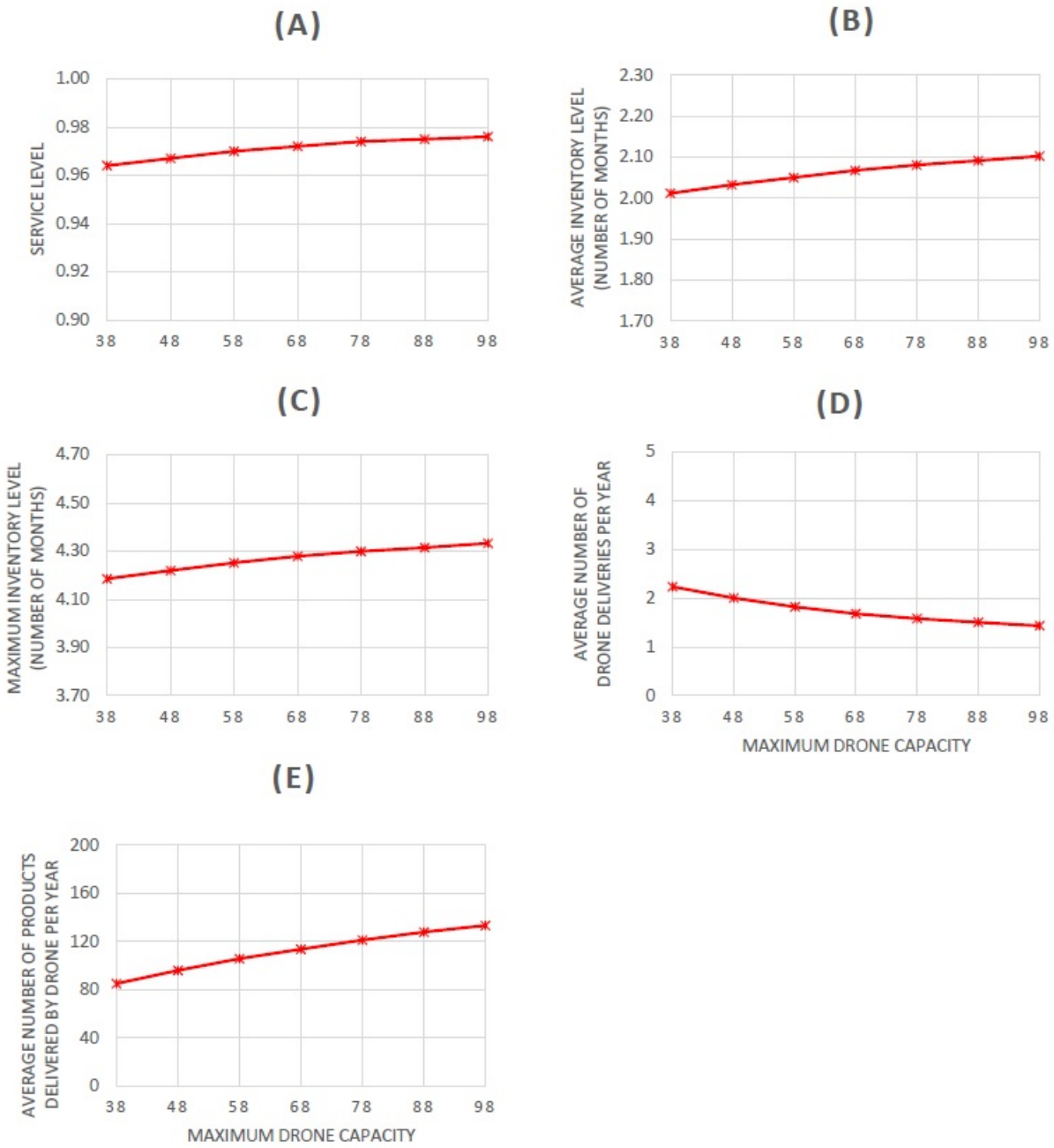

Figure 6: Effect of several drone capacity levels on service level (A), average inventory level (B), maximum inventory level (C), average number of drone deliveries per year (D), and average number of products delivered by drone per year (E) for the 4×I[-3,0] inventory policy. Standard seasonality and lead time values are applied. Maximum drone usage is set to maximum 1 drone per week and 4 drones per month, with a critical inventory level equal to 0.

4×D[-9,-12] inventory policies ranging from 97.1% to 99.6% (Figure 7A). Average and maximum inventory levels are lower than for both 3×I[-3,0] and 4×D[-9,-12] (Figures 7B and 7C). The average number of deliveries is approximately two deliveries less than for 3×I[-3,0], but 3-6 deliveries more than with the 4×D[-9,-12] inventory policy (Figure 7D). In comparison, the 3×D[-12,-9] without drones even has lower average and maximum inventory levels, but this comes with the cost of a service level of 91.7% which is 5.4% to 7.9% lower (Figures 7A-7C).

From the inventory policies investigated here, the 3×D[-9,-12] policy is best suited to incorporate drones, due to it's unique combination of relatively high service level and low inventory level.

5 Discussion

Simulation results for inventory policies recommended by USAID | DELIVER PROJECT (2011a,b) slightly differ from the results obtained by Leung et al.. Weekly average demand estimates contain a seasonality factor of 1.51, while Leung et al. find a seasonality factor of 2.2. Possibly, Leung et al. computed the seasonality factor from daily average demand estimates, since daily fluctuations are relatively larger than weekly fluctuations in demand. This could, however, not be checked due to the unavailability of daily average estimated demand data. The different demand seasonality factor should not cause different results from the simulation in the 'base case'. That is, for a seasonality factor of 1.51 (2.2 in Leung et al. (2016)) demand estimates are not adjusted since this is the natural level of seasonality observed from the data, so that the used weekly average demand estimates used in the simulation are identical and results should be as well. However, in this research slightly different results are observed. Generally, service level results in this paper are slightly higher with differences of 2-6% depending on the inventory policy. Average and maximum inventory levels are slightly lower with 0.1-0.3 and 0.3-0.5 months of demand respectively. Maybe this is due to a slightly different interpretation of how the simulation is implemented, since Leung et al. do not precisely specify every step and calculation.

The results from this study clearly show benefits of incorporating drones in the distribution of essential medicines on the district level induced by the fact that drone deliveries have short and constant lead times. The main question is, however, if it is feasible to implement a drone delivery system in a low-income, widespread country such as Zambia. As mentioned earlier, factors to take into account include the limited range and cargo of the drones, the (costs of) installation of drone stations and training of personnel, and the costs per flight.

In Section 2.2 is explained how with less than one drone station per district, all of Zambia can be covered by the drone delivery system. In that case, limited range is not a problem and drones can even travel further by recharging at another drone station and then continuing the trip. One additional factor to take into account is that sufficient inventory is needed at the drone stations. However, drones allow inventory levels at health clinics to be significantly lower, reducing costs at the health clinic level.

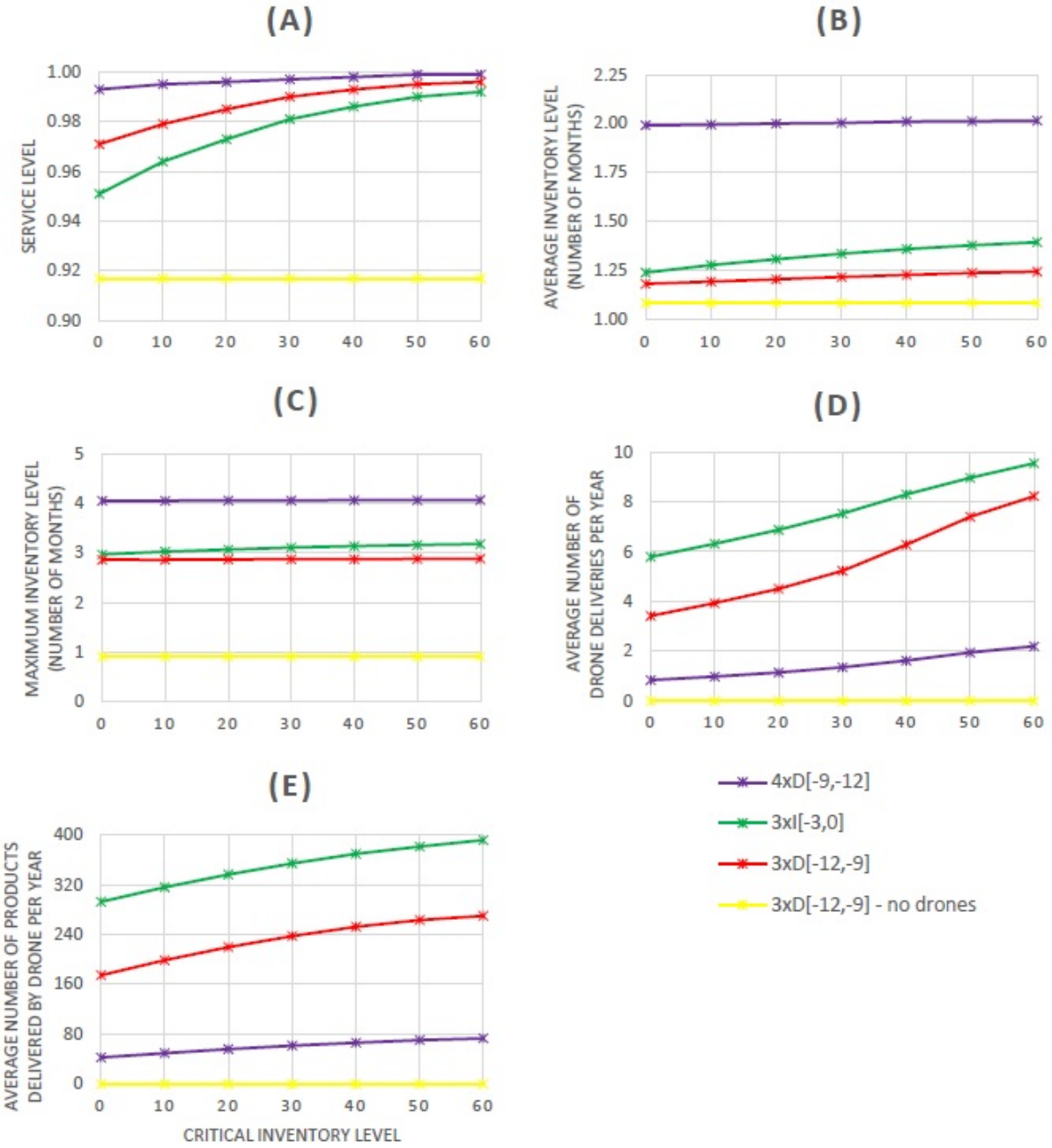

Figure 7: Results of performance measures from simulation are shown for different inventory policies and critical inventory levels. Performance measures included are service level (A), average inventory level in months of demand (B), maximum inventory level in months of demand (C), average number of drone deliveries per year (D), and average number of products delivered by drone per year in adult regimens (E). All policies include drones except for the '3×[-12,-9] - no drones' policy. Results are obtained with a drone capacity of 68 adult regimens and maximum drone delivery frequency of two per week.

To prevent the unavailability of drones and high costs due to overuse of drones, a maximum drone delivery frequency per health clinic can be imposed. This research shows that a maximum frequency of two per week works well for anti-malarial drugs, improving service levels and keeping the number of drone orders limited. This means that with a smartly chosen maximum drone delivery frequency, the maximum cargo of the drones is not a problem.

The costs per drone delivery are about the same as for delivery by car, at least in case of emergency and if the health care product has a short shelf life or is rarely needed (Wakefield, 2017). This means that for the delivery of for example blood, drones are more efficient, since they are much faster and do not cost significantly more than trucks. However, for regular non-emergency medicines, such as anti-malarial drugs, one could argue that drones are not necessary if demand forecasts are precise and delivery is on time (Foth, 2017). Yet, this has shown to be a challenge, especially in low-income countries and for health products with demand fluctuating over the year (Leung et al., 2016). Programs trying to solve this cost multiple millions of dollars (Foth, 2017). Still, many see drones as a quick fix, instead of solving the real problem: a lack of investment in infrastructure. Solving this issue has benefits that reach far beyond better health product availability, such as better access to markets, work and schools and the creation of jobs (Asiedu, 2019; Foth, 2017). These opportunity costs should be taken into account when considering the implementation of a drone delivery system in sub-Saharan countries. Yet, many African governments remain interested in a drone delivery system, because it is significantly more effective in the short to medium term compared to investing in infrastructure (Asiedu, 2019).

Besides Zipline other commercial drone companies have shown interest to operate in health care systems of sub-Saharan African countries (Bright and Stein, 2018). If this interests starts to develop further, competition will probably lower prices. Among the commercial drone companies, local companies start to pop up as well. If the governments of sub-Saharan countries manage to work with these local companies, this will boost the economy of their country. Besides competition, technical development is another factor that might reduce drone delivery costs in the short term. This technical development can for example show in better or cheaper batteries, longer travel distance and larger or heavier cargo.

A last factor of concern might be communication. In the case of frequent drone orders and immediate deliveries, clear communication is of critical importance. This might seem an issue in low-income countries, but Zipline shows, for example in Rwanda, that it is possible to achieve this via an online app or WhatsApp message.

One limitation of this study is the fact that the drone system is only used for anti-malarial drugs. The effectiveness of the drone system, however, highly depends on the combination with drone delivery of other health products. Another limitation to take into account is the ignorance of extra inventory needed at the drone stations. If not managed correctly this can be a limiting factor for the drone deliveries. Lastly, service levels obtained for policies incorporating drones might actually be lower than they would be in reality. For simplicity reasons, drone orders can only be placed at the end of each

week in the simulation model. In reality, this would not make sense, because whenever an inventory level below the critical value is observed, a drone order can be placed. This reduces the number of stock-outs between the time critical inventory levels are reached and the time of the drone delivery.

6 Conclusion

This study confirms the conclusions stated by Leung et al. (2016). Simulation sample paths show how stock-outs of essential medicines in Zambia's health clinics can be explained by not correctly accounting for demand seasonality in the inventory policy and long delivery lead times.

Despite the slight differences in results compared to Leung et al. (2016), most of the conclusions are identical. Consistent with the findings of Leung et al. 4×D[-12,-9] is the best performing inventory policy in terms of service level out of all recommended inventory policies by USAID|DELIVER. Altogether, the ability of these inventory policies to reduce stock-outs is limited, also because the higher service levels are paired with higher inventory levels. One difference is however that, because of the higher service levels obtained in this research, the 4×D[-12,-9] inventory policy reaches reasonable performance with a service level of 98% and average and maximum inventory levels of 2.0 and 4.0 months of demand, which is an increase of less than 0.1 months of demand compared to the 4×I[-3,0] policy.

Incorporating drones in the distribution of essential medicines to health clinics in Zambia can raise service levels and lower stock-outs. Drones are incorporated by the possibility to place a drone delivery order at the end of the week if inventory reaches a critical level, and are complementary to the existing truck delivery schedule. Drone orders are delivered before the start of the next week. For the inventory policy with the highest service level (4×D[-12,-9]), the number of stock-outs can be reduced to less than half by incorporating drones. For inventory policies with lower multiplication factors, incorporating drones can extensively improve the service level, while maintaining relatively low inventory levels at health clinics. However, arguably the best results are obtained when combining these two outcomes.

Simulation results show that incorporating drones in the distribution process gives the ability to maintain lower inventory levels in health clinics, while reaching a respectable service level. This might be the biggest advantage of incorporating drones, in combination with the fact that these results can be obtained with a limited number of drone deliveries per health clinic. When implementing the 3×D[-12,-9] policy with drones, a critical inventory level of minimally 30 results in service levels of 99% and higher. This is under the assumption that drones have a cargo of 68 adult regimens and operating a maximum drone delivery frequency of two per week. Average and maximum inventory levels for this policy are 1.2 and 2.9 months of demand respectively, which is well below the current inventory levels of the 4×I[-3,0] inventory policy. On average a health clinic will order 5 to 8 drone deliveries per year, with a total of 237 to 270 adult regimens delivered by drone.

All together, this research shows that it is feasible and in numerous ways beneficial to incorporate drones in the distribution of essential medicines at district level in Zambia. Whether this is the best solution for the country and its economy as a whole is hard to say, because factors to be taken into account reach far beyond the number of stock-outs. Firstly, a more extensive cost analysis should be conducted. But even then, decisions on whether the drone system is worth the money are a consideration between short and long term benefits. Another factor influencing the decision is how drone systems and local drone companies develop over the next few years.

Although this research solely focuses on using drones to deliver anti-malarial drugs, results will also to a large extent hold for other health care products suffering from stock-outs at health clinics, especially for health care products with seasonal demand. Results in this research show that only a few drone deliveries per health clinic per year are needed to significantly improve availability of anti-malarial drugs. If a drone delivery system is implemented, it seems logical to also use this system for the delivery of other health products. This will lead to a much more efficient drone delivery system and is therefore an interesting subject for future research.

The results of this research will also be relevant for regional distribution of essential medicines in other low and middle income countries with bad road infrastructure and rural populations, which is the case for most sub-Saharan countries.

References

Asiedu, K. G. (2019). An ambitious drone delivery health service in ghana is tackling key logistics challenges. https://qz.com/africa/1604374/ziplines-drone-delivery-launches-in-ghana-with-vaccines/.

Bossert, T. J., Bowser, D. M., and Amenyah, J. K. (2007). Is decentralization good for logistics systems? evidence on essential medicine logistics in ghana and guatemala. *Health Policy and Planning, 22(2)*, pages 73–82.

Bright, J. and Stein, S. (2018). African experiments with drone technologies could leapfrog decades of infrastructure neglect. https://techcrunch.com/2018/09/16/african-experiments-with-drone-technologies-could-leapfrog-decades-of-infrastructure-neglect/.

Cameron, A., Ewen, M., Ross-Degnan, D., Ball, D., and Laing, R. (2009). Medicine prices, availability, and affordability in 36 developing and middle-income countries: a secondary analysis. *The lancet, 373(9659)*, pages 240–249.

Chen, L., Kim, S. H., and Lee, H. L. (2016). Enabling healthcare delivery through vehicle maintenance. *History*.

Foth, J. (2017). We haven't considered the true cost of drone delivery medical services in africa. https://qz.com/africa/1090693/zipline-drones-in-africa-like-rwanda-and-tanzania-have-an-opportunity-cost/.

Haidari, L. A., Brown, S. T., Ferguson, M., Bancroft, E., Spiker, M., Wilcox, A., ..., and Lee, B. Y. (2016). The economic and operational value of using drones to transport vaccines. *Vaccine, 34(34)*, pages 4062–4067.

Kangwana, B. B., Njogu, J., Wasunna, B., Kedenge, S. V., Memusi, D. N., Goodman, C. A., ..., and Snow, R. W. (2009). Malaria drug shortages in kenya: a major failure to provide access to effective treatment. *The American journal of tropical medicine and hygiene, 80(5)*, pages 737–738.

Leung, N. H. Z., Chen, A., Yadav, P., and Gallien, J. (2016). The impact of inventory management on stock-outs of essential drugs in sub-saharan africa: secondary analysis of a field experiment in zambia. *PloS one 11(5)*.

Pasquet, A., Messou, E., Gabillard, D., Minga, A., Depoulosky, A., Deuffic-Burban, S., ., and Yazdanpanah, Y. (2010). Impact of drug stock-outs on death and retention to care among hiv-infected patients on combination antiretroviral therapy in abidjan, côte d'ivoire. *PloS one, 5(10)*.

Tatham, P., Stadler, F., Murray, A., and Shaban, R. Z. (2017). Flying maggots: a smart logistic solution to an enduring medical challenge. *Journal of Humanitarian Logistics and Supply Chain Management, 7(2)*, pages 172–193.

Thiels, C. A., Aho, J. M., Zietlow, S. P., and Jenkins, D. H. (2015). Use of unmanned aerial vehicles for medical product transport. *Air medical journal, 34(2)*, pages 104–108.

USAID | DELIVER PROJECT (2011a). *Guidelines for Managing the Malaria Supply Chain: A Companion to the Logistics Handbook.*

USAID | DELIVER PROJECT (2011b). *The Logistics Handbook: A Practical Guide for the Supply Chain Management of Health Commodities.* Second edition.

Vledder, M., Friedman, J., Sjöblom, M., Brown, T., and Yadav, P. (2015a). Optimal supply chain structure for distributing essential drugs in low income countries: results from a randomized experiment. *Ross School of Business Paper, (1269)*.

Vledder, M., Friedman, J., Sjöblom, M., Brown, T., and Yadav, P. (2015b). Optimal supply chain structure for distributing essential drugs in low income countries: results from a randomized experiment. *Ross School of Business Paper, (1269)*.

Waako, P. J., Odoi-Adome, R., Obua, C., Owino, E., Tumwikirize, W., Ogwal-Okeng, J., ..., and Aupont, O. (2009). Existing capacity to manage pharmaceuticals and related commodities in east africa: an assessment with specific reference to antiretroviral therapy. *Human Resources for Health, 7(1)*.

Wakefield, J. (2017). Us drone company eyes tanzania for medical deliveries. https://www.bbc.com/news/technology-40935773.

Watson, N., Bausell, L., Ingles, A., and Printz, N. (2014). Malaria seasonality and calculating resupply: Applications of the look-ahead seasonality indices in zambia, burkina faso, and zimbabwe. Technical report, USAID | DELIVER PROJECT.

World Health Organization, Geneva (2018). Global health estimates 2016: Deaths by cause, age, sex, by country and by region, 2000-2016. Technical report.

Yadav, P.and Tata, H. L. and Babaley, M. (2011). The world medicines situation 2011.

Yadav, P. (2015). Health product supply chains in developing countries: diagnosis of the root causes of underperformance and an agenda for reform. *Health Systems Reform, 1(2)*, pages 142–154.

Zipline. https://www.flyzipline.com/.

Appendix

A Weekly Demand Estimates

Week	Demand	Week	Demand	Week	Demand	Week	Demand
1	42.0	13	49.5	25	28.4	37	26.8
2	45.0	14	46.9	26	29.0	38	26.1
3	48.4	15	44.3	27	30.1	39	25.8
4	51.4	16	41.6	28	31.5	40	25.9
5	54.0	17	38.6	29	32.7	41	26.6
6	55.9	18	35.7	30	33.2	42	27.9
7	56.6	19	33.4	31	33.0	43	29.7
8	56.6	20	31.7	32	32.1	44	31.8
9	56.3	21	30.3	33	31.0	45	34.1
10	55.1	22	29.4	34	29.8	46	36.2
11	53.3	23	28.7	35	28.8	47	37.9
12	51.4	24	28.3	36	27.8	48	39.8

Table 1: Weekly demand estimates for AL products (adult doses) as estimated by Leung et al. Leung et al. (2016). The week is defined as the week of the year, week 1 being the first week of January and assuming 4 weeks per month.

B Simulation Code

B.1 Main

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Simulates the inventory policy of an average health clinic in Zambia
 * @author Ingrid Pool
 */
public class SimulationClinic
{
    public static void main(String[] args)
    {
        // settings
        int warmupPeriod = 2; // length of warm-up period in years
        int dataColPeriod = 3; // length of data collection period in years
        boolean issuesOrDemand = true; // issues = true, demand = false
        int timeIntervalStart = -3; // first month of historical interval for
            ↪ calculating averages
```

```
        int timeIntervalEnd = 0; // next to last month of historical interval for calculating averages
        int multiplicationFactor = 4; // multiple of average to calculate replenishment target
        double seasonalityFactor; // ratio of peak demand mean to average demand mean
        double meanSecLeadTime; // average number of weeks it takes to deliver from district center to local clinic
        boolean useDrones; // true if the use of drones is allowed, false otherwise
        double standardCriticalInvLevel = 0.0; // value of inventory below which a drone order can be placed
        double criticalInvLevel = standardCriticalInvLevel; // value of inventory below which a drone order can be placed
        double standardMaxDroneCapacity = 68.0; // maximum number of adult regimens per drone delivery
        double maxDroneCapacity = standardMaxDroneCapacity; // maximum number of adult regimens per drone delivery
        int standardMaxDronesPerMonth = 4; // maximum amount of drone deliveries per month
        int maxDronesPerMonth = standardMaxDronesPerMonth; // maximum amount of drone deliveries per month
        int standardMaxDronesPerWeek = 1; // maximum amount of drone deliveries per week
        int maxDronesPerWeek = standardMaxDronesPerWeek; // maximum amount of drone deliveries per week

        // Iterate over multiple seasonality factors and lead times without the use of drones
        useDrones = false;
        List<Double> seasonalityFactors = new ArrayList<Double>(Arrays.asList(1.0, 1.17, 1.34, 1.51, 1.68, 1.85, 2.02));
        double standardSeasonalityFactor = 1.51;
        List<Double> meanSecLeadTimes = new ArrayList<Double>(Arrays.asList(2.2, 2.7333, 3.2666, 3.8, 4.3333, 4.8666, 5.4));
        double standardMeanSecLeadTime = 3.8;
        meanSecLeadTime = standardMeanSecLeadTime;
        for (int s = 0; s < seasonalityFactors.size();s++)
        {
            seasonalityFactor = seasonalityFactors.get(s);
            System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean secondary lead-time: " + meanSecLeadTime);

            // 100.000 replications
            performReplications(warmupPeriod, dataColPeriod, issuesOrDemand, timeIntervalStart, timeIntervalEnd, multiplicationFactor, seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel, maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
        }
        System.out.println("");
        seasonalityFactor = standardSeasonalityFactor;

        for (int m = 0; m < meanSecLeadTimes.size(); m++)
        {
            meanSecLeadTime = meanSecLeadTimes.get(m);
```

```
53              if (meanSecLeadTime != standardMeanSecLeadTime)
54              {
55                  System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean
                        ↪ secondary lead-time: " + meanSecLeadTime);
56
57                  // 100.000 replications
58                  performReplications(warmupPeriod, dataColPeriod, issuesOrDemand,
                        ↪ timeIntervalStart, timeIntervalEnd, multiplicationFactor,
                        ↪ seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel,
                        ↪ maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
59              }
60          }
61          System.out.println("");
62          meanSecLeadTime = standardMeanSecLeadTime;
63
64          // Iterate over different drone rules with standard seasonality factor and lead
                ↪ time
65          useDrones = true;
66          // Iterate over maximum drones per month without weekly restriction
67          List<Integer> maxDronesPerMonthList = new ArrayList<>(Arrays.asList(0, 1, 2, 3,
                ↪ 4, 5, 6));
68          for (int d = 0; d < maxDronesPerMonthList.size(); d++)
69          {
70              maxDronesPerMonth = maxDronesPerMonthList.get(d);
71              maxDronesPerWeek = maxDronesPerMonth;
72              System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean
                    ↪ secondary lead-time: " + meanSecLeadTime + ", Max drones per month:
                    ↪ " + maxDronesPerMonth + ", Max drones per week: " + maxDronesPerWeek
                    ↪ + ", Critical inventory level: " + criticalInvLevel + ", Maximum
                    ↪ drone capacity: " + maxDroneCapacity);
73
74              // 100.000 replications
75              performReplications(warmupPeriod, dataColPeriod, issuesOrDemand,
                    ↪ timeIntervalStart, timeIntervalEnd, multiplicationFactor,
                    ↪ seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel,
                    ↪ maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
76          }
77          System.out.println("");
78          maxDronesPerMonth = standardMaxDronesPerMonth;
79          maxDronesPerWeek = standardMaxDronesPerWeek;
80
81          // Iterate over maximum drones per week without monthly restriction
82          List<Integer> maxDronesPerWeekList = new ArrayList<>(Arrays.asList(0, 1, 2, 3,
                ↪ 4, 5, 6));
83          for (int w = 0; w < maxDronesPerWeekList.size(); w++)
84          {
85              maxDronesPerWeek = maxDronesPerWeekList.get(w);
86              maxDronesPerMonth = 4*maxDronesPerWeek;
87              System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean
                    ↪ secondary lead-time: " + meanSecLeadTime + ", Max drones per month:
                    ↪ " + maxDronesPerMonth + ", Max drones per week: " + maxDronesPerWeek
                    ↪ + ", Critical inventory level: " + criticalInvLevel + ", Maximum
                    ↪ drone capacity: " + maxDroneCapacity);
```

```
88
89              // 100.000 replications
90              performReplications(warmupPeriod, dataColPeriod, issuesOrDemand,
                    ↪ timeIntervalStart, timeIntervalEnd, multiplicationFactor,
                    ↪ seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel,
                    ↪ maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
91          }
92          System.out.println("");
93          maxDronesPerMonth = standardMaxDronesPerMonth;
94          maxDronesPerWeek = standardMaxDronesPerWeek;
95
96          // Iterate over critical inventory levels for using drones with standard or
                ↪ adjusted weekly or monthly restrictions
97          List<Double> criticalInvLevelList = new ArrayList<>(Arrays.asList(0.0, 10.0,
                ↪ 20.0, 30.0, 40.0, 50.0, 60.0));
98          for (int c = 0; c < criticalInvLevelList.size(); c++)
99          {
100             // possibly adjust drone frequency restrictions
101             maxDronesPerWeek = 2;
102             maxDronesPerMonth = 4*maxDronesPerWeek;
103             criticalInvLevel = criticalInvLevelList.get(c);
104             System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean
                    ↪ secondary lead-time: " + meanSecLeadTime + ", Max drones per month:
                    ↪ " + maxDronesPerMonth + ", Max drones per week: " + maxDronesPerWeek
                    ↪  + ", Critical inventory level: " + criticalInvLevel + ", Maximum
                    ↪ drone capacity: " + maxDroneCapacity);
105
106             // 100.000 replications
107             performReplications(warmupPeriod, dataColPeriod, issuesOrDemand,
                    ↪ timeIntervalStart, timeIntervalEnd, multiplicationFactor,
                    ↪ seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel,
                    ↪ maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
108         }
109         System.out.println("");
110         maxDronesPerMonth = standardMaxDronesPerMonth;
111         maxDronesPerWeek = standardMaxDronesPerWeek;
112         criticalInvLevel = standardCriticalInvLevel;
113
114         // Iterate over drone capacity with weekly and monthly restrictions
115         List<Double> droneCapacityList = new ArrayList<>(Arrays.asList(38.0, 48.0,
                ↪ 58.0, 68.0, 78.0, 88.0, 98.0));
116         for (int p = 0; p < maxDronesPerWeekList.size(); p++)
117         {
118             maxDroneCapacity = droneCapacityList.get(p);
119             System.out.println("Seasonality factor: " + seasonalityFactor + ", Mean
                    ↪ secondary lead-time: " + meanSecLeadTime + ", Max drones per month:
                    ↪ " + maxDronesPerMonth + ", Max drones per week: " + maxDronesPerWeek
                    ↪  + ", Critical inventory level: " + criticalInvLevel + ", Maximum
                    ↪ drone capacity: " + maxDroneCapacity);
120
121             // 100.000 replications
122             performReplications(warmupPeriod, dataColPeriod, issuesOrDemand,
                    ↪ timeIntervalStart, timeIntervalEnd, multiplicationFactor,
```

```
                ↪ seasonalityFactor, meanSecLeadTime, useDrones, criticalInvLevel,
                ↪ maxDroneCapacity, maxDronesPerMonth, maxDronesPerWeek);
123         }
124         maxDronesPerMonth = standardMaxDronesPerMonth;
125         maxDronesPerWeek = standardMaxDronesPerWeek;
126     }
127
128     /**
129      * Performs a set number of replications with the given parameters and computes and
            ↪ prints statistical results.
130      * @param warmupPeriod, period in years used to eliminate effects of the initial
            ↪ state of the simulation system.
131      * @param dataColPeriod, period in years after the warmupPeriod from which data is
            ↪ collected.
132      * @param issuesOrDemand, true if historical issues are used in computing the
            ↪ replenishment target, false if demand is used.
133      * @param timeIntervalStart, start of the interval from which historical values are
            ↪ used in computing the replenishment target
134      * @param timeIntervalEnd, end of the interval from which historical values are
            ↪ used in computing the replenishment target
135      * @param multiplicationFactor, factor by which the average of historical values is
            ↪ multiplied to compute the replenishment target
136      * @param seasonalityFactor, measure of fluctuation in demand values.
137      * @param meanSecLeadTime, mean secondary lead time.
138      * @param useDrones, true if the use of drones is allowed, false otherwise.
139      * @param criticalInvLevel, level from below which drone orders can be placed.
140      * @param maxDroneCapacity, maximum cargo of a drone
141      * @param maxDronesPerMonth, maximum number of drone orders that can be placed per
            ↪ month for a health clinic.
142      * @param maxDronesPerWeek, maximum number of drone orders that can be placed per
            ↪ week for a health clinic.
143      */
144     public static void performReplications(int warmupPeriod, int dataColPeriod, boolean
        ↪ issuesOrDemand, int timeIntervalStart, int timeIntervalEnd, int
        ↪ multiplicationFactor, double seasonalityFactor, double meanSecLeadTime,
        ↪ boolean useDrones, double criticalInvLevel, double maxDroneCapacity, int
        ↪ maxDronesPerMonth, int maxDronesPerWeek)
145     {
146         List<Double> serviceLevels = new ArrayList<Double>();
147         List<Double> avgInventoryLevels = new ArrayList<Double>();
148         List<Double> maxInventoryLevels = new ArrayList<Double>();
149         List<Double> avgDemandPerMonth = new ArrayList<Double>();
150         List<Double> avgLeadTimes = new ArrayList<Double>();
151         List<Double> avgNrsDroneDeliveriesYear = new ArrayList<Double>();
152         List<Double> avgNrsProductsDeliveredDroneYear = new ArrayList<Double>();
153         for (int replNr = 1; replNr <= 100000; replNr++)
154         {
155             int startYear = 2009; // startyear of simulation
156             int startWeek = 21; // start week of year of simulation
157             List<Double> perfMeasures = new ArrayList<Double>();
158             perfMeasures = runSimulation(warmupPeriod, dataColPeriod, startYear,
                ↪ startWeek, issuesOrDemand, timeIntervalStart, timeIntervalEnd,
                ↪ multiplicationFactor, seasonalityFactor, meanSecLeadTime, useDrones,
```

```
                ↪  criticalInvLevel, maxDroneCapacity, maxDronesPerMonth,
                ↪ maxDronesPerWeek);
159             serviceLevels.add(perfMeasures.get(0)/perfMeasures.get(1));
160             avgInventoryLevels.add(perfMeasures.get(2)/48.0);
161             maxInventoryLevels.add(perfMeasures.get(3));
162             avgDemandPerMonth.add(perfMeasures.get(4));
163             avgLeadTimes.add(perfMeasures.get(5));
164             avgNrsDroneDeliveriesYear.add(perfMeasures.get(6));
165             avgNrsProductsDeliveredDroneYear.add(perfMeasures.get(7));
166         }
167
168         // compute statistical results
169         double serviceLevel = computeAverage(serviceLevels);
170         double avgInventoryLevel = computeAverage(avgInventoryLevels);
171         double maxInventoryLevel = computeAverage(maxInventoryLevels);
172         double demandPerMonth = computeAverage(avgDemandPerMonth);
173         double avgLeadtime = computeAverage(avgLeadTimes);
174         double avgNrDroneDeliveriesYear = computeAverage(avgNrsDroneDeliveriesYear);
175         double avgNrProductsDeliveredDroneYear = computeAverage(
                ↪ avgNrsProductsDeliveredDroneYear);
176
177         // print results
178         System.out.println("Service level: " + serviceLevel);
179         System.out.println("Average inventory level: " + avgInventoryLevel);
180         System.out.println("Maximum inventory level: " + maxInventoryLevel);
181         System.out.println("Average demand per month: " + demandPerMonth);
182         System.out.println("Average lead time: " + avgLeadtime);
183         System.out.println("Average number of drone deliveries per year: " +
                ↪ avgNrDroneDeliveriesYear);
184         System.out.println("Average number of products delivered by drone per year: " +
                ↪  avgNrProductsDeliveredDroneYear);
185     }
186
187     /**
188      * Simulates a single health clinic with the given parameters
189      * @param warmupPeriod, period in years used to eliminate effects of the initial
                ↪ state of the simulation system.
190      * @param dataColPeriod, period in years after the warmupPeriod from which data is
                ↪ collected.
191      * @param startYear, year in which the simulation starts.
192      * @param startWeek, week of the year in which the simulation starts.
193      * @param issuesOrDemandI, true if historical issues are used in computing the
                ↪ replenishment target, false if demand is used.
194      * @param timeIntervalStartI, start of the interval from which historical values
                ↪ are used in computing the replenishment target
195      * @param timeIntervalEndI, end of the interval from which historical values are
                ↪ used in computing the replenishment target
196      * @param multiplicationFactorI, factor by which the average of historical values
                ↪ is multiplied to compute the replenishment target
197      * @param seasonalityFactorI, measure of fluctuation in demand values.
198      * @param meanSecLeadTimeI, mean secondary lead time.
199      * @param useDronesI, true if the use of drones is allowed, false otherwise.
200      * @param criticalInvLevelI, level from below which drone orders can be placed.
```

```
201         * @param maxDroneCapacityI, maximum cargo of a drone
202         * @param maxDronesPerMonthI, maximum number of drone orders that can be placed per
                ↪ month for a health clinic.
203         * @param maxDronesPerWeekI, maximum number of drone orders that can be placed per
                ↪ week for a health clinic.
204         * @return a list of performance measures obtained from the dataColPeriod.
205         */
206        public static List<Double> runSimulation(int warmupPeriod, int dataColPeriod, int
               ↪ startYear, int startWeek, boolean issuesOrDemandI, int timeIntervalStartI,
               ↪ int timeIntervalEndI, int multiplicationFactorI, double seasonalityFactorI,
               ↪ double meanSecLeadTimeI, boolean useDronesI, double criticalInvLevelI,
               ↪ double maxDroneCapacityI, int maxDronesPerMonthI, int maxDronesPerWeekI)
207        {
208            Clinic simulationClinic = new Clinic();
209            simulationClinic.setPolicy(issuesOrDemandI, timeIntervalStartI,
                   ↪ timeIntervalEndI, multiplicationFactorI, seasonalityFactorI,
                   ↪ meanSecLeadTimeI, startWeek, useDronesI, criticalInvLevelI,
                   ↪ maxDroneCapacityI, maxDronesPerMonthI, maxDronesPerWeekI);
210            int currentSimulationWeek = 1;
211            int currentWeek = startWeek;
212            int currentYearFromStartYear = 1;
213            int currentYear = startYear;
214
215            // simulate warm-up period
216            while (currentSimulationWeek <= warmupPeriod*48)
217            {
218                int currentMonth = (int) Math.ceil(currentWeek/4.0);
219
220                // deliver replenishment order if scheduled to be delivered this week
221                simulationClinic.receiveOrder(currentSimulationWeek);
222
223                // in first week of calendar month place order
224                if(currentWeek%4 == 1)
225                {
226                    simulationClinic.placeOrder(currentSimulationWeek, currentMonth,
                           ↪ currentYearFromStartYear);
227                }
228
229                // generate demand
230                simulationClinic.generateDemand(currentWeek, currentMonth,
                       ↪ currentYearFromStartYear, currentSimulationWeek);
231
232                // update simulation to next week
233                currentWeek += 1;
234                currentSimulationWeek += 1;
235                if (currentWeek%48 == 1)
236                {
237                    currentYear += 1;
238                    currentWeek = 1;
239                    currentYearFromStartYear += 1;
240                }
241            }
242
```

```
243         // simulate data-collection period
244         simulationClinic.resetPerformanceMeasures();
245         while (currentSimulationWeek <= (warmupPeriod+dataColPeriod)*48)
246         {
247             int currentMonth = (int) Math.ceil(currentWeek/4.0);
248 
249             // deliver replenishment order if scheduled to be delivered this week
250             simulationClinic.receiveOrder(currentSimulationWeek);
251 
252             // in first week of calendar month place order
253             if(currentWeek%4 == 1)
254             {
255                 simulationClinic.placeOrder(currentSimulationWeek, currentMonth,
                        ↪ currentYearFromStartYear);
256             }
257 
258             // generate demand
259             simulationClinic.generateDemand(currentWeek, currentMonth,
                    ↪ currentYearFromStartYear, currentSimulationWeek);
260 
261             // update simulation to next week
262             currentWeek += 1;
263             currentSimulationWeek += 1;
264             if (currentWeek%48 == 1)
265             {
266                 currentYear += 1;
267                 currentWeek = 1;
268                 currentYearFromStartYear += 1;
269             }
270         }
271 
272         // update performance measures
273         double satisfiedDemand = simulationClinic.getIssues(warmupPeriod, dataColPeriod
                ↪ );
274         double totalDemand = simulationClinic.getDemand(warmupPeriod, dataColPeriod);
275         double sumInventory = simulationClinic.getInventory(dataColPeriod);
276         double maxInventory = simulationClinic.getMaxInventory(warmupPeriod,
                ↪ dataColPeriod);
277         double avgDemandMonth = simulationClinic.getAverageDemandPerMonth(warmupPeriod,
                ↪  dataColPeriod);
278         double avgLeadTime = simulationClinic.getAverageLeadTimes();
279         double avgNrDroneDeliveriesYear = simulationClinic.getAvgNrDroneDeliveries(
                ↪ dataColPeriod);
280         double avgNrProductsDeliveredDroneYear = simulationClinic.
                ↪ getAvgNrProductsDeliveredDrone(dataColPeriod);
281 
282         List<Double> perfMeasures = new ArrayList<Double>();
283         perfMeasures.add(satisfiedDemand);
284         perfMeasures.add(totalDemand);
285         perfMeasures.add(sumInventory);
286         perfMeasures.add(maxInventory);
287         perfMeasures.add(avgDemandMonth);
288         perfMeasures.add(avgLeadTime);
```

```
289             perfMeasures.add(avgNrDroneDeliveriesYear);
290             perfMeasures.add(avgNrProductsDeliveredDroneYear);
291 
292             return perfMeasures;
293         }
294 
295         /**
296          * Computes the average of a list of statistical results
297          * @param list, a list of values from which the average must be computed.
298          * @return the average of the values in list.
299          */
300         public static double computeAverage(List<Double> list)
301         {
302             double sum = 0;
303             double size = list.size();
304             for (int j = 0; j < size; j++)
305             {
306                 sum += list.get(j);
307             }
308 
309             double result = sum/size;
310             return result;
311         }
312 }
```

B.2 Clinic

```
1  import java.util.ArrayList;
2  import java.util.Arrays;
3  import java.util.List;
4  import java.util.Random;
5  
6  /**
7   * The Clinic class allows to keep track of the state of the health clinic.
8   * @author Ingrid Pool
9   */
10 public class Clinic
11 {
12     private double inventory = 0; // number of products in stock
13     private double outstandingInventory = 0; // number of products ordered, but not yet
       ↪ delivered
14     private List<Order> outstandingOrders = new ArrayList<Order>();
15     private List<Double> satisfiedDemand = new ArrayList<Double>();
16     private List<Double> totalDemand = new ArrayList<Double>();
17     private List<Double> totalUnmetDemand = new ArrayList<Double>();
18     private double sumInventory = 0; // sum of inventorylevels
19     private double deliveries = 0; // counter for the number of orders delivered
20     private double leadtimes = 0; // sum of the lead times of orders delivered.
21     private List<Double> maxInventory = new ArrayList<Double>();
22     private boolean issuesOrDemand; // issues = true, demand = false
23     private int timeIntervalStart; // first month of historical interval for
       ↪ calculating averages
```

```
24      private int timeIntervalEnd; // next to last month of historical interval for
            ↪ calculating averages
25      private double multiplicationFactor; // multiple of average to calculate
            ↪ replenishment target
26      private double seasonalityFactor; // ratio of peak demand mean to average demand
            ↪ mean
27      private double meanSecLeadTime; // average number of weeks it takes to deliver from
            ↪  district center to local clinic
28      private List<Double> historicalDemand = new ArrayList<Double>(Arrays.asList(186.8,
            ↪ 223.1, 216.0, 182.3, 139.4, 116.8, 118.9, 131.0, 117.3, 104.5, 116.0, 147.9)
            ↪ );
29      private List<Double> unmetDemand = new ArrayList<Double>();
30      private List<Double> historicalIssues = new ArrayList<Double>(Arrays.asList(186.8,
            ↪ 223.1, 216.0, 182.3, 139.4, 116.8, 118.9, 131.0, 117.3, 104.5, 116.0, 147.9)
            ↪ );
31      private List<Double> delayProbabilities = new ArrayList<Double>(Arrays.asList
            ↪ (0.779, 0.765, 0.781, 0.846, 0.927, 0.969, 0.974, 0.984, 0.986, 0.99, 0.946,
            ↪  0.848));
32      private List<Double> demandWeek = new ArrayList<Double>(Arrays.asList(42.0, 45.0,
            ↪ 48.4, 51.4, 54.0, 55.9, 56.6, 56.6, 56.3, 55.1, 53.3, 51.4, 49.5, 46.9,
            ↪ 44.3, 41.6, 38.6, 35.7, 33.4, 31.7, 30.3, 29.4, 28.7, 28.3, 28.4, 29.0,
            ↪ 30.1, 31.5, 32.7, 33.2, 33.0, 32.1, 31.0, 29.8, 28.8, 27.8, 26.8, 26.1,
            ↪ 25.8, 25.9, 26.6, 27.9, 29.7, 31.8, 34.1, 36.2, 37.9, 39.8));
33      private boolean useDrones;
34      private double criticalInvLevel;
35      private double maxDroneCapacity;
36      private int maxDronesPerMonth;
37      private int dronesThisMonth = 0;
38      private int maxDronesPerWeek;
39      private int nrDroneDeliveries = 0;
40      private int nrProductsDeliveredDrone = 0;
41
42      /**
43       * Initializes the clinic
44       * @param issuesOrDemandI, compute replenishment target based of historical issues
             ↪ (true) or demand (false)
45       * @param timeIntervalStartI, start of interval from which to use historical values
46       * @param timeIntervalEndI, end of interval from which to use historical values
47       * @param multiplicationFactorI, multiple of times the average of historical values
             ↪  has to be replenished to
48       * @param seasonalityFactorI, factor representing the variability of the demand
             ↪ values over the year
49       * @param meanSecLeadTimeI, average amount of weeks to deliver from district to
             ↪ health clinic
50       * @param startWeek, week of the year in which the simulation starts.
51       * @param useDronesI, true if the use of drones is allowed, false otherwise.
52       * @param criticalInvLevelI, level from below which drone orders can be placed.
53       * @param maxDroneCapacityI, maximum cargo of a drone
54       * @param maxDronesPerMonthI, maximum number of drone orders that can be placed per
             ↪  month for a health clinic.
55       * @param maxDronesPerWeekI, maximum number of drone orders that can be placed per
             ↪ week for a health clinic.
56       */
```

```
57      public void setPolicy(boolean issuesOrDemandI, int timeIntervalStartI, int
            ↪ timeIntervalEndI, int multiplicationFactorI, double seasonalityFactorI,
            ↪ double meanSecLeadTimeI, int startWeek, boolean useDronesI, double
            ↪ criticalInvLevelI, double maxDroneCapacityI, int maxDronesPerMonthI, int
            ↪ maxDronesPerWeekI)
58      {
59          // Set the settings for the policy
60          issuesOrDemand = issuesOrDemandI;
61          timeIntervalStart = timeIntervalStartI;
62          timeIntervalEnd = timeIntervalEndI;
63          multiplicationFactor = multiplicationFactorI;
64          seasonalityFactor = seasonalityFactorI;
65          meanSecLeadTime = meanSecLeadTimeI;
66          useDrones = useDronesI;
67          criticalInvLevel = criticalInvLevelI;
68          maxDroneCapacity = maxDroneCapacityI;
69          maxDronesPerMonth = maxDronesPerMonthI;
70          maxDronesPerWeek = maxDronesPerWeekI;
71
72          // fill unmetDemand array
73          for (int index = 0; index < historicalDemand.size(); index++)
74          {
75              unmetDemand.add(historicalDemand.get(index) - historicalIssues.get(index));
76          }
77
78          // Adjust the demand values according to the seasonality factor
79          boolean week = true;
80          boolean month = false;
81          double sum = 0;
82          double sumWeek = 0;
83          double sumMonth = 0;
84          double maxDemand = 0;
85          double maxDemandWeek = 0;
86          double maxDemandMonth = 0;
87          double divider = 0.0;
88          for (int index = 0; index < demandWeek.size(); index++)
89          {
90              double demandValue = demandWeek.get(index);
91              sumWeek += demandValue;
92              if (demandValue > maxDemandWeek)
93              {
94                  maxDemandWeek = demandValue;
95              }
96          }
97          for (int index = 0; index < historicalDemand.size(); index++)
98          {
99              double demandValue = historicalDemand.get(index);
100             sumMonth += demandValue;
101             if (demandValue > maxDemandMonth)
102             {
103                 maxDemandMonth = demandValue;
104             }
105         }
```

```
106         if (week == true)
107         {
108             divider = 48.0;
109             sum = sumWeek;
110             maxDemand = maxDemandWeek;
111         }
112         if (month == true)
113         {
114             divider = 12.0;
115             sum = sumMonth;
116             maxDemand = maxDemandMonth;
117         }
118 
119         double averageDemand = sum/divider;
120         double maxDemandTarget = averageDemand*seasonalityFactor;
121         double gapAvgTarget = maxDemandTarget - averageDemand;
122         double gapAvgMax = maxDemand - averageDemand;
123         double neededMultiplication = gapAvgTarget/gapAvgMax;
124         double averageDemandMonth = sumMonth/12.0;
125         for (int i = 0; i < historicalDemand.size(); i++)
126         {
127             double demandValue = historicalDemand.get(i);
128             double gap = demandValue - averageDemandMonth;
129             historicalDemand.set(i, averageDemandMonth + gap*neededMultiplication);
130             historicalIssues.set(i, averageDemandMonth + gap*neededMultiplication);
131         }
132 
133         double averageDemandWeek = sumWeek/48.0;
134         for (int j = 0; j < demandWeek.size(); j++)
135         {
136             double demandValue = demandWeek.get(j);
137             double gap = demandValue - averageDemandWeek;
138             demandWeek.set(j, averageDemandWeek + gap*neededMultiplication);
139         }
140 
141         // if startweek is not at beginning of year, adjust historical demand
142         int duplicateMonths = (int) Math.floor(startWeek/4);
143         for (int i = 12; i < duplicateMonths+12; i++)
144         {
145             historicalDemand.add(historicalDemand.get(i-12));
146             historicalIssues.add(historicalIssues.get(i-12));
147         }
148     }
149 
150     /**
151      * Finds the order that is delivered next from the list of outstanding orders
152      * @return the next order to be delivered
153      */
154     private Order getNextOrder()
155     {
156         Order nextOrder = null;
157         for (int i=0; i<outstandingOrders.size();i++)
158         {
```

```
159             Order thisOrder = outstandingOrders.get(i);
160             if (nextOrder == null || thisOrder.deliveryTime < nextOrder.deliveryTime)
161             {
162                 nextOrder = thisOrder;
163             }
164         }
165         return nextOrder;
166     }
167 
168     /**
169      * Receives the orders that are delivered at week t and update the inventory
         ↪ statistics
170      * @param t current week of the simulation
171      */
172     public void receiveOrder(int t)
173     {
174         // update performance measures inventory level
175         sumInventory += inventory;
176         double currentMaxInventory = 0;
177         if (maxInventory.size() == Math.ceil(t/48.0))
178         {
179             currentMaxInventory = maxInventory.get((int) Math.ceil(t/48.0) - 1);
180         }
181         else
182         {
183             maxInventory.add(0.0);
184         }
185         if (inventory > currentMaxInventory)
186         {
187             maxInventory.set((int) Math.ceil(t/48.0)-1, inventory);
188         }
189 
190         // receive order
191         while(getNextOrder() != null && getNextOrder().deliveryTime == t)
192         {
193             Order nextOrder = getNextOrder();
194             inventory += nextOrder.quantity;
195             outstandingInventory -= nextOrder.quantity;
196             outstandingOrders.remove(nextOrder);
197         }
198     }
199 
200     /**
201      * Places a new order with quantity based of replenishment target, inventory and
         ↪ outstanding inventory
202      * @param currentSimulationWeek, current week of the simulation
203      * @param currentMonth, current month of the year
204      * @param currentYearFromStartYear, current year of the simulation
205      */
206     public void placeOrder(int currentSimulationWeek, int currentMonth, int
        ↪ currentYearFromStartYear)
207     {
208         // compute order quantity
```

```
209         double replenishmentTarget = 0.0;
210         List<Double> historicalValues = new ArrayList<Double>();
211         if (issuesOrDemand == true)
212         {
213             historicalValues = historicalIssues;
214         }
215         else if (issuesOrDemand == false)
216         {
217             historicalValues = historicalDemand;
218         }
219         for (int month = timeIntervalStart; month < timeIntervalEnd; month++)
220         {
221             replenishmentTarget += historicalValues.get(month+currentMonth-1+12*
                ↪ currentYearFromStartYear);
222         }
223         replenishmentTarget = replenishmentTarget/(timeIntervalEnd-timeIntervalStart)*
            ↪ multiplicationFactor;
224         double orderQuantity = Math.round(replenishmentTarget - inventory -
            ↪ outstandingInventory);
225 
226         // place order
227         if (orderQuantity > 0)
228         {
229             Order order = new Order();
230             order.setQuantity(orderQuantity);
231             double delayProbability = delayProbabilities.get(currentMonth-1);
232             order.setDeliveryTime(currentSimulationWeek, delayProbability,
                ↪ meanSecLeadTime);
233             outstandingInventory += orderQuantity;
234 
235             // update outstandingOrders
236             outstandingOrders.add(order);
237             deliveries += 1;
238             leadtimes += order.deliveryTime-currentSimulationWeek;
239         }
240     }
241 
242     /**
243      * Generates random demand for week currentSimulationWeek, fulfills this demand if
            ↪ possible and updates the statistics
244      * @param currentWeekOfYear, current week of the year
245      * @param currentMonth, current month of the year
246      * @param currentYearFromStartYear, current year of the simulation
247      * @param currentSimulationWeek, current week of the simulation
248      */
249     public void generateDemand(int currentWeekOfYear, int currentMonth, int
        ↪ currentYearFromStartYear, int currentSimulationWeek)
250     {
251         // Generate demand with lognormal distribution
252         double generatedDemand;
253         double meanDemand = demandWeek.get(currentWeekOfYear-1);
254         double stdDevDemand = 0.5*meanDemand;
255         Random r = new Random();
```

```
256             double stdNorm = r.nextGaussian();
257             double mu = Math.log(Math.pow(meanDemand,2.0)/Math.sqrt(Math.pow(stdDevDemand
                    ↪ ,2.0) + Math.pow(meanDemand,2.0)));
258             double sigmaSqrt = Math.log(1+Math.pow(stdDevDemand, 2.0)/Math.pow(meanDemand,
                    ↪ 2.0));
259             generatedDemand = Math.round((Math.exp(stdNorm*Math.sqrt(sigmaSqrt)+mu)));
260
261             // fulfill demand from inventory
262             double issues = Math.min(generatedDemand, inventory);
263             inventory = inventory - issues;
264
265             // update historical issues, historical demand and unmet demand
266             updateHistoricalValues(historicalDemand, currentMonth, currentYearFromStartYear
                    ↪ , generatedDemand);
267             updateHistoricalValues(historicalIssues, currentMonth, currentYearFromStartYear
                    ↪ , issues);
268             updateHistoricalValues(unmetDemand, currentMonth, currentYearFromStartYear,
                    ↪ generatedDemand-issues);
269
270             // update performance measures
271             double oldSatisfiedDemand = 0;
272             double oldTotalDemand = 0;
273             double oldTotalUnmetDemand= 0;
274             if (satisfiedDemand.size() == Math.ceil(currentSimulationWeek/48.0))
275             {
276                 oldSatisfiedDemand = satisfiedDemand.get((int) Math.ceil(
                        ↪ currentSimulationWeek/48.0) - 1);
277                 oldTotalDemand = totalDemand.get((int) Math.ceil(currentSimulationWeek
                        ↪ /48.0) - 1);
278                 oldTotalUnmetDemand = totalUnmetDemand.get((int) Math.ceil(
                        ↪ currentSimulationWeek/48.0) - 1);
279             }
280             else
281             {
282                 satisfiedDemand.add(0.0);
283                 totalDemand.add(0.0);
284                 totalUnmetDemand.add(0.0);
285             }
286             satisfiedDemand.set((int) Math.ceil(currentSimulationWeek/48.0)-1,
                    ↪ oldSatisfiedDemand + issues);
287             totalDemand.set((int) Math.ceil(currentSimulationWeek/48.0)-1, oldTotalDemand +
                    ↪  generatedDemand);
288             totalUnmetDemand.set((int) Math.ceil(currentSimulationWeek/48.0)-1,
                    ↪ oldTotalUnmetDemand + generatedDemand-issues);
289
290             // if critical inventory level is reached, if possible place and receive drone
                    ↪ delivery
291             int dronesThisWeek = 0;
292             if (currentWeekOfYear%4==1)
293             {
294                 dronesThisMonth = 0;
295             }
```

```
296         if (useDrones == true && inventory <= criticalInvLevel && dronesThisMonth <
                ↪ maxDronesPerMonth)
297         {
298             int nextWeek = (currentWeekOfYear+1)%48;
299             if (nextWeek == 0)
300             {
301                 nextWeek = 48;
302             }
303             int nextNextWeek = (currentWeekOfYear+2)%48;
304             if (nextNextWeek == 0)
305             {
306                 nextNextWeek = 48;
307             }
308             double expectedDemandNextTwoWeeks = demandWeek.get(nextWeek-1) + demandWeek
                    ↪ .get(nextNextWeek-1);
309             double extraInventoryNeeded = expectedDemandNextTwoWeeks-inventory;
310             while (extraInventoryNeeded > 0 && dronesThisMonth < maxDronesPerMonth &&
                    ↪ dronesThisWeek < maxDronesPerWeek)
311             {
312                 double orderQuantity = Math.min(extraInventoryNeeded, maxDroneCapacity)
                        ↪ ;
313                 inventory += orderQuantity;
314                 extraInventoryNeeded -= orderQuantity;
315                 dronesThisMonth += 1;
316                 dronesThisWeek += 1;
317                 nrDroneDeliveries += 1;
318                 nrProductsDeliveredDrone += orderQuantity;
319             }
320         }
321     }
322
323     /**
324      * Updates the list with historical values per month by adding addValue to the
            ↪ value of the appropriate month
325      * @param List, the list to be updated
326      * @param currentMonth, current month of the year
327      * @param currentYearFromStartYear, current year of the simulation
328      * @param addValue, value to add to the current value of the month
329      */
330     public void updateHistoricalValues(List<Double> List, int currentMonth, int
            ↪ currentYearFromStartYear, double addValue)
331     {
332         if (List.size() == currentMonth+12*currentYearFromStartYear)
333         {
334             double oldValue = List.get(currentMonth-1+12*currentYearFromStartYear);
335             List.set(currentMonth-1+12*currentYearFromStartYear, (oldValue + addValue))
                    ↪ ;
336         }
337         else
338         {
339             List.add(addValue);
340         }
341     }
```

```
342
343     /**
344      * Resets the performance measures to 0.
345      */
346     public void resetPerformanceMeasures()
347     {
348         sumInventory = 0;
349         deliveries = 0;
350         leadtimes = 0;
351         nrDroneDeliveries = 0;
352         nrProductsDeliveredDrone = 0;
353     }
354
355     /**
356      * Computes the average amount of issues per year for dataColPeriod
357      * @param warmupPeriod, period in years the simulation runs before collecting data
358      * @param dataColPeriod, period in years over which data is collected from the
         ↪ simulation
359      * @return average amount of issues per year over the data collection period
360      */
361     public double getIssues(int warmupPeriod, int dataColPeriod)
362     {
363         double sumSatisfiedDemand = 0;
364         double totalYears = dataColPeriod;
365         for (int i = warmupPeriod; i < satisfiedDemand.size(); i++)
366         {
367             sumSatisfiedDemand += satisfiedDemand.get(i);
368         }
369         return sumSatisfiedDemand/totalYears;
370     }
371
372     /**
373      * Computes the average amount of demand per year for dataColPeriod
374      * @param warmupPeriod, period in years the simulation runs before collecting data
375      * @param dataColPeriod, period in years over which data is collected from the
         ↪ simulation
376      * @return average amount of demand per year over the data collection period
377      */
378     public double getDemand(int warmupPeriod, int dataColPeriod)
379     {
380         double sumTotalDemand = 0;
381         double totalYears = dataColPeriod;
382         for (int i = warmupPeriod; i < totalDemand.size(); i++)
383         {
384             sumTotalDemand += totalDemand.get(i);
385         }
386         return sumTotalDemand/totalYears;
387     }
388
389     /**
390      * Computes the average amount of inventory per year for dataColPeriod
391      * @param warmupPeriod, period in years the simulation runs before collecting data
```

```
392         * @param dataColPeriod, period in years over which data is collected from the
                ↪ simulation
393         * @return average amount of inventory per year over the data collection period
394         */
395        public double getInventory(int dataColPeriod)
396        {
397            double totalYears = dataColPeriod;
398            return sumInventory/totalYears;
399        }
400
401        /**
402         * Computes the average of the maximum inventory per year for dataColPeriod
403         * @param warmupPeriod, period in years the simulation runs before collecting data
404         * @param dataColPeriod, period in years over which data is collected from the
                ↪ simulation
405         * @return average maximum inventory per year over the data collection period
406         */
407        public double getMaxInventory(int warmupPeriod, int dataColPeriod)
408        {
409            double sumMaxInventory = 0;
410            double totalYears = dataColPeriod;
411            for (int i = warmupPeriod; i < maxInventory.size(); i++)
412            {
413                sumMaxInventory += maxInventory.get(i);
414            }
415            return sumMaxInventory/totalYears;
416        }
417
418        /**
419         * Computes the average demand per month for dataColPeriod
420         * @param warmupPeriod, period in years the simulation runs before collecting data
421         * @param dataColPeriod, period in years over which data is collected from the
                ↪ simulation
422         * @return average demand per month over the data collection period
423         */
424        public double getAverageDemandPerMonth(int warmupPeriod, int dataColPeriod)
425        {
426            double sumDemand = getDemand(warmupPeriod, dataColPeriod)*dataColPeriod;
427            return sumDemand/(dataColPeriod*12);
428        }
429
430        /**
431         * Computes the average lead time for dataColPeriod
432         * @param warmupPeriod, period in years the simulation runs before collecting data
433         * @param dataColPeriod, period in years over which data is collected from the
                ↪ simulation
434         * @return average lead time over the data collection period
435         */
436        public double getAverageLeadTimes()
437        {
438            double average = leadtimes/deliveries;
439            return average;
440        }
```

```
441
442     /**
443      * computes the average number of drone deliveries per year
444      * @param dataColPeriod, period in years over which data is collected from the
             ↪ simulation
445      * @return the average number of drone deliveries per year
446      */
447     public double getAvgNrDroneDeliveries(int dataColPeriod)
448     {
449         double totalYears = dataColPeriod;
450         double average = nrDroneDeliveries/totalYears;
451         return average;
452     }
453
454     /**
455      * Computes the average number of products delivered by drone per year
456      * @param dataColPeriod, period in years over which data is collected from the
             ↪ simulation
457      * @return the average number of productes delivered by drone per year
458      */
459     public double getAvgNrProductsDeliveredDrone(int dataColPeriod)
460     {
461         double totalYears = dataColPeriod;
462         double average = nrProductsDeliveredDrone/totalYears;
463         return average;
464     }
465 }
```

B.3 Order

```
1  /**
2   * The Order class keeps track of the quantity and delivery time of an order.
3   * @author Ingrid Pool
4   */
5  public class Order
6  {
7      double quantity;
8      int deliveryTime;
9
10     /**
11      * Sets the quantity of the order equal to orderQuantity
12      * @param orderQuantity, quantity of the order
13      */
14     public void setQuantity(double orderQuantity)
15     {
16         quantity = orderQuantity;
17     }
18
19     /**
20      * Computes the lead time of the order (weeks) and sets the week of delivery
21      * @param currentSimulationWeek, current week of the simulation
22      * @param delayProbability, probability that in this month the order is NOT delayed
```

```
23          * @param meanSecLeadTimeI, average lead time (weeks) of the secondary stage of
                ↪ delivery
24          */
25         public void setDeliveryTime(int currentSimulationWeek, double delayProbability,
               ↪ double meanSecLeadTimeI)
26         {
27             int primaryLeadTimeWeeks = 2;
28
29             // generate secondary lead time in weeks without access problems with geometric
                   ↪ distribution
30             int secondaryLeadTimeWeeks = 1;
31             double meanSecLeadTimeWeeks = meanSecLeadTimeI;
32             double probability = 1.0/meanSecLeadTimeWeeks;
33             boolean stop = false;
34             while(!stop)
35             {
36                 double rand = Math.random();
37                 if (rand <= probability)
38                 {
39                     stop = true;
40                     break;
41                 }
42                 else
43                 {
44                     secondaryLeadTimeWeeks += 1;
45                 }
46             }
47
48             // generate seasonal access problems for secondary lead time with bernoulli
                   ↪ distribution
49             int secondaryLeadTimeWeeksAccessProblems = 0;
50             double rand2 = Math.random();
51             if (rand2 >= delayProbability)
52             {
53                 secondaryLeadTimeWeeksAccessProblems += 1;
54             }
55
56             deliveryTime = currentSimulationWeek + primaryLeadTimeWeeks +
                   ↪ secondaryLeadTimeWeeks + secondaryLeadTimeWeeksAccessProblems;
57         }
58 }
```