coordinated\_ordering9

GroupB

28/07/2020

Table of Contents

# Data preparation

library(readxl)  
  
product\_data <- read\_excel("Data\_ordering.xlsx",sheet = "product data")  
box\_data <- read\_excel("Data\_ordering.xlsx",sheet = "box data")  
#rack data  
Total\_racks=8  
levels\_per\_rack=4  
rack\_length= 6000  
rack\_width=1750  
rack\_height=300

## Convert to Boxes

Convert demand , quantity and unit price all from part to box which is the capacity of a box for part .

Demand for box with part .

Price for box with material .

value of box

product\_data$demand\_per\_year= ceiling((product\_data$`demand per day` \*262)/ product\_data$`pieces/box`)   
  
product\_data$box\_cost= product\_data$`pieces/box` \* product\_data$price  
  
product\_data

## # A tibble: 62 x 7  
## `material ID` `demand per day` `box ID` `pieces/box` price demand\_per\_year  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 7305667+74 12 6203060 45 1.51 70  
## 2 7305669+77 30 6203059 15 7.62 524  
## 3 7305670+77 30 6203059 15 1.62 524  
## 4 7305673+76 30 6203059 16 1.51 492  
## 5 7305674+76 30 6203059 16 1.51 492  
## 6 7305817+74 60 6203060 16 4.55 983  
## 7 7305819+79 30 6203059 6 11.1 1310  
## 8 7305820+79 30 6203059 6 2.86 1310  
## 9 7305823+73 30 6203059 30 4.05 262  
## 10 7305824+73 30 6203059 30 2.56 262  
## # ... with 52 more rows, and 1 more variable: box\_cost <dbl>

product\_data <- merge(product\_data, box\_data[,c("box ID", "ordering cost (€)")], by = "box ID")  
  
colnames(product\_data)[8] <- "ordering\_cost"  
product\_data

## box ID material ID demand per day pieces/box price demand\_per\_year box\_cost  
## 1 3103147 7348808+73 30 18 2.58 437 46.44  
## 2 3103147 7346592+72 360 64 1.99 1474 127.36  
## 3 3104147 7455691+71 24 20 4.39 315 87.80  
## 4 3104147 7455692+71 24 20 4.39 315 87.80  
## 5 3106410 7305865+77 30 14 3.15 562 44.10  
## 6 3108210 7306005+73 6 20 1.50 79 30.00  
## 7 3108210 7308013+79 30 13 19.46 605 252.98  
## 8 3108210 7308014+79 30 13 1.58 605 20.54  
## 9 3108210 7455688+72 24 24 10.70 262 256.80  
## 10 3108210 7455633+73 24 8 24.57 786 196.56  
## 11 3108210 7455634+73 24 8 1.60 786 12.80  
## 12 3108210 7306006+73 6 20 3.15 79 63.00  
## 13 3108210 7455687+72 24 24 3.99 262 95.76  
## 14 6203059 7305834+73 30 12 4.27 655 51.24  
## 15 6203059 7305819+79 30 6 11.13 1310 66.78  
## 16 6203059 7305670+77 30 15 1.62 524 24.30  
## 17 6203059 7305673+76 30 16 1.51 492 24.16  
## 18 6203059 7305863+75 60 16 1.52 983 24.32  
## 19 6203059 7305669+77 30 15 7.62 524 114.30  
## 20 6203059 7343137+72 30 32 6.74 246 215.68  
## 21 6203059 7305820+79 30 6 2.86 1310 17.16  
## 22 6203059 7305674+76 30 16 1.51 492 24.16  
## 23 6203059 7305824+73 30 30 2.56 262 76.80  
## 24 6203059 7326393+74 30 12 1.75 655 21.00  
## 25 6203059 7305833+73 30 12 2.02 655 24.24  
## 26 6203059 7306017+79 6 9 1.51 175 13.59  
## 27 6203059 7306018+79 6 9 6.62 175 59.58  
## 28 6203059 7313982+31 6 12 9.67 131 116.04  
## 29 6203059 7305867+76 60 48 1.58 328 75.84  
## 30 6203059 7305971+75 6 15 3.32 105 49.80  
## 31 6203059 7305972+75 6 15 3.13 105 46.95  
## 32 6203059 7343138+74 30 32 1.96 246 62.72  
## 33 6203059 7305823+73 30 30 4.05 262 121.50  
## 34 6203059 7326394+74 30 12 2.96 655 35.52  
## 35 6203059 7313981+31 6 12 5.68 131 68.16  
## 36 6203060 7305667+74 12 45 1.51 70 67.95  
## 37 6203060 7306023+78 30 15 2.13 524 31.95  
## 38 6203060 7306024+78 30 15 2.15 524 32.25  
## 39 6203060 7326409+74 30 11 1.87 715 20.57  
## 40 6203060 7326825+72 30 18 1.68 437 30.24  
## 41 6203060 7305817+74 60 16 4.55 983 72.80  
## 42 6203060 7306013+73 6 16 1.53 99 24.48  
## 43 6203060 7306037+79 6 20 3.82 79 76.40  
## 44 6203060 7306009+75 6 13 3.62 121 47.06  
## 45 6203060 7306010+75 6 13 2.13 121 27.69  
## 46 6203060 7326826+72 30 18 1.69 437 30.42  
## 47 6203060 7306014+73 6 16 1.70 99 27.20  
## 48 6203060 7306038+30 6 20 2.28 79 45.60  
## 49 6203061 7306034+77 6 30 2.11 53 63.30  
## 50 6203061 7328209+73 30 16 1.87 492 29.92  
## 51 6203061 7305977+74 24 36 1.54 175 55.44  
## 52 6203061 7305832+73 30 30 1.50 262 45.00  
## 53 6203061 7306026+75 6 20 1.62 79 32.40  
## 54 6203061 7306033+77 6 30 2.53 53 75.90  
## 55 6203061 7328210+73 30 16 1.76 492 28.16  
## 56 6203061 7306025+75 6 20 1.85 79 37.00  
## 57 6203061 7355006+72 30 36 3.08 219 110.88  
## 58 6203061 7346813+73 60 30 2.45 524 73.50  
## 59 6203061 7313474+73 30 8 2.82 983 22.56  
## 60 6203061 7355005+72 30 36 1.67 219 60.12  
## 61 6203062 7332421+76 30 5 3.45 1572 17.25  
## 62 6203062 7332422+76 30 5 1.57 1572 7.85  
## ordering\_cost  
## 1 50  
## 2 50  
## 3 45  
## 4 45  
## 5 60  
## 6 65  
## 7 65  
## 8 65  
## 9 65  
## 10 65  
## 11 65  
## 12 65  
## 13 65  
## 14 80  
## 15 80  
## 16 80  
## 17 80  
## 18 80  
## 19 80  
## 20 80  
## 21 80  
## 22 80  
## 23 80  
## 24 80  
## 25 80  
## 26 80  
## 27 80  
## 28 80  
## 29 80  
## 30 80  
## 31 80  
## 32 80  
## 33 80  
## 34 80  
## 35 80  
## 36 75  
## 37 75  
## 38 75  
## 39 75  
## 40 75  
## 41 75  
## 42 75  
## 43 75  
## 44 75  
## 45 75  
## 46 75  
## 47 75  
## 48 75  
## 49 50  
## 50 50  
## 51 50  
## 52 50  
## 53 50  
## 54 50  
## 55 50  
## 56 50  
## 57 50  
## 58 50  
## 59 50  
## 60 50  
## 61 70  
## 62 70

# Separate Ordering (SO)

ordering cost for part

stock holding cost rate based on unit price and interest rate therefore

calculate EOQ for each part i:

$$ q\_i^\* = \sqrt \frac{2 \cdot y\_i \cdot (c\_i^{or}+c^{-or})}{pr\_i \cdot h}$$

Min EOQ, used in starting feasible solution.

$$ q\_i^\* = \sqrt \frac{2 \cdot y\_i \cdot (c\_i^{or})} {pr\_i \cdot h}$$

## EOQ modelling

#dei= demand\_per\_year, cori= ordering cost for box i  
#cord= ordering cost, pri= box cost, h= interest rate  
  
# directly vectorized  
so\_eoq\_fun <- Vectorize(function(dei,cori,cord,pri,h){  
 eoq<- sqrt((2\*dei\*(cori+cord))/(pri\*h))  
 return(eoq)  
})  
  
# maximum EOQ provided there is no coordination of ordering cycles (i.e., cord are not shared among parallely ordered items)  
vec\_so\_eoq\_fun.max <- so\_eoq\_fun(dei = product\_data$demand\_per\_year,cori=product\_data$ordering\_cost,cord=1500,pri=product\_data$box\_cost,h=0.10)  
# minimum EOQ disregarding common ordering cost cord  
vec\_so\_eoq\_fun.min <- so\_eoq\_fun(dei = product\_data$demand\_per\_year,cori=product\_data$ordering\_cost,cord=0,pri=product\_data$box\_cost,h=0.10)  
  
product\_data$eoq.min <- round(vec\_so\_eoq\_fun.min)  
product\_data$eoq.max <- round(vec\_so\_eoq\_fun.max)  
product\_data

## box ID material ID demand per day pieces/box price demand\_per\_year box\_cost  
## 1 3103147 7348808+73 30 18 2.58 437 46.44  
## 2 3103147 7346592+72 360 64 1.99 1474 127.36  
## 3 3104147 7455691+71 24 20 4.39 315 87.80  
## 4 3104147 7455692+71 24 20 4.39 315 87.80  
## 5 3106410 7305865+77 30 14 3.15 562 44.10  
## 6 3108210 7306005+73 6 20 1.50 79 30.00  
## 7 3108210 7308013+79 30 13 19.46 605 252.98  
## 8 3108210 7308014+79 30 13 1.58 605 20.54  
## 9 3108210 7455688+72 24 24 10.70 262 256.80  
## 10 3108210 7455633+73 24 8 24.57 786 196.56  
## 11 3108210 7455634+73 24 8 1.60 786 12.80  
## 12 3108210 7306006+73 6 20 3.15 79 63.00  
## 13 3108210 7455687+72 24 24 3.99 262 95.76  
## 14 6203059 7305834+73 30 12 4.27 655 51.24  
## 15 6203059 7305819+79 30 6 11.13 1310 66.78  
## 16 6203059 7305670+77 30 15 1.62 524 24.30  
## 17 6203059 7305673+76 30 16 1.51 492 24.16  
## 18 6203059 7305863+75 60 16 1.52 983 24.32  
## 19 6203059 7305669+77 30 15 7.62 524 114.30  
## 20 6203059 7343137+72 30 32 6.74 246 215.68  
## 21 6203059 7305820+79 30 6 2.86 1310 17.16  
## 22 6203059 7305674+76 30 16 1.51 492 24.16  
## 23 6203059 7305824+73 30 30 2.56 262 76.80  
## 24 6203059 7326393+74 30 12 1.75 655 21.00  
## 25 6203059 7305833+73 30 12 2.02 655 24.24  
## 26 6203059 7306017+79 6 9 1.51 175 13.59  
## 27 6203059 7306018+79 6 9 6.62 175 59.58  
## 28 6203059 7313982+31 6 12 9.67 131 116.04  
## 29 6203059 7305867+76 60 48 1.58 328 75.84  
## 30 6203059 7305971+75 6 15 3.32 105 49.80  
## 31 6203059 7305972+75 6 15 3.13 105 46.95  
## 32 6203059 7343138+74 30 32 1.96 246 62.72  
## 33 6203059 7305823+73 30 30 4.05 262 121.50  
## 34 6203059 7326394+74 30 12 2.96 655 35.52  
## 35 6203059 7313981+31 6 12 5.68 131 68.16  
## 36 6203060 7305667+74 12 45 1.51 70 67.95  
## 37 6203060 7306023+78 30 15 2.13 524 31.95  
## 38 6203060 7306024+78 30 15 2.15 524 32.25  
## 39 6203060 7326409+74 30 11 1.87 715 20.57  
## 40 6203060 7326825+72 30 18 1.68 437 30.24  
## 41 6203060 7305817+74 60 16 4.55 983 72.80  
## 42 6203060 7306013+73 6 16 1.53 99 24.48  
## 43 6203060 7306037+79 6 20 3.82 79 76.40  
## 44 6203060 7306009+75 6 13 3.62 121 47.06  
## 45 6203060 7306010+75 6 13 2.13 121 27.69  
## 46 6203060 7326826+72 30 18 1.69 437 30.42  
## 47 6203060 7306014+73 6 16 1.70 99 27.20  
## 48 6203060 7306038+30 6 20 2.28 79 45.60  
## 49 6203061 7306034+77 6 30 2.11 53 63.30  
## 50 6203061 7328209+73 30 16 1.87 492 29.92  
## 51 6203061 7305977+74 24 36 1.54 175 55.44  
## 52 6203061 7305832+73 30 30 1.50 262 45.00  
## 53 6203061 7306026+75 6 20 1.62 79 32.40  
## 54 6203061 7306033+77 6 30 2.53 53 75.90  
## 55 6203061 7328210+73 30 16 1.76 492 28.16  
## 56 6203061 7306025+75 6 20 1.85 79 37.00  
## 57 6203061 7355006+72 30 36 3.08 219 110.88  
## 58 6203061 7346813+73 60 30 2.45 524 73.50  
## 59 6203061 7313474+73 30 8 2.82 983 22.56  
## 60 6203061 7355005+72 30 36 1.67 219 60.12  
## 61 6203062 7332421+76 30 5 3.45 1572 17.25  
## 62 6203062 7332422+76 30 5 1.57 1572 7.85  
## ordering\_cost eoq.min eoq.max  
## 1 50 97 540  
## 2 50 108 599  
## 3 45 57 333  
## 4 45 57 333  
## 5 60 124 631  
## 6 65 59 287  
## 7 65 56 274  
## 8 65 196 960  
## 9 65 36 179  
## 10 65 72 354  
## 11 65 283 1386  
## 12 65 40 198  
## 13 65 60 293  
## 14 80 143 636  
## 15 80 177 787  
## 16 80 186 825  
## 17 80 181 802  
## 18 80 254 1130  
## 19 80 86 381  
## 20 80 43 190  
## 21 80 349 1553  
## 22 80 181 802  
## 23 80 74 328  
## 24 80 223 993  
## 25 80 208 924  
## 26 80 144 638  
## 27 80 69 305  
## 28 80 43 189  
## 29 80 83 370  
## 30 80 58 258  
## 31 80 60 266  
## 32 80 79 352  
## 33 80 59 261  
## 34 80 172 763  
## 35 80 55 246  
## 36 75 39 180  
## 37 75 157 719  
## 38 75 156 715  
## 39 75 228 1046  
## 40 75 147 675  
## 41 75 142 652  
## 42 75 78 357  
## 43 75 39 180  
## 44 75 62 285  
## 45 75 81 371  
## 46 75 147 673  
## 47 75 74 339  
## 48 75 51 234  
## 49 50 29 161  
## 50 50 128 714  
## 51 50 56 313  
## 52 50 76 425  
## 53 50 49 275  
## 54 50 26 147  
## 55 50 132 736  
## 56 50 46 257  
## 57 50 44 247  
## 58 50 84 470  
## 59 50 209 1162  
## 60 50 60 336  
## 61 70 357 1692  
## 62 70 529 2508

## Lane Occupation

sorting column lenght or width.This determines the lane width

if sorted by length then the value is width and vice versa, used for rack lenght.

Add and to product\_data table

# constraints #####################################  
# this can be formulated more elegantly, but it works and that suffices  
b\_sorting <- double(length(product\_data$`box ID`))   
b\_not\_sorting <-double(length(product\_data$`box ID`))  
  
for(j in 1: length(box\_data$`box ID`)){  
 for (k in 1:length(product\_data$`box ID`)) {  
 if(box\_data$`box ID`[j]==product\_data$`box ID`[k]){  
   
 if(box\_data$sorting[j]=="width"){  
 b\_sorting[k] <- box\_data$width[j]  
 b\_not\_sorting[k]<- box\_data$length[j]  
   
 }else{  
 b\_sorting[k] <- box\_data$length[j]  
 b\_not\_sorting[k]<- box\_data$width[j]  
 }  
 }  
 }  
   
}  
  
product\_data$b\_sorting <- b\_sorting   
product\_data$b\_not\_sorting <- b\_not\_sorting  
product\_data

## box ID material ID demand per day pieces/box price demand\_per\_year box\_cost  
## 1 3103147 7348808+73 30 18 2.58 437 46.44  
## 2 3103147 7346592+72 360 64 1.99 1474 127.36  
## 3 3104147 7455691+71 24 20 4.39 315 87.80  
## 4 3104147 7455692+71 24 20 4.39 315 87.80  
## 5 3106410 7305865+77 30 14 3.15 562 44.10  
## 6 3108210 7306005+73 6 20 1.50 79 30.00  
## 7 3108210 7308013+79 30 13 19.46 605 252.98  
## 8 3108210 7308014+79 30 13 1.58 605 20.54  
## 9 3108210 7455688+72 24 24 10.70 262 256.80  
## 10 3108210 7455633+73 24 8 24.57 786 196.56  
## 11 3108210 7455634+73 24 8 1.60 786 12.80  
## 12 3108210 7306006+73 6 20 3.15 79 63.00  
## 13 3108210 7455687+72 24 24 3.99 262 95.76  
## 14 6203059 7305834+73 30 12 4.27 655 51.24  
## 15 6203059 7305819+79 30 6 11.13 1310 66.78  
## 16 6203059 7305670+77 30 15 1.62 524 24.30  
## 17 6203059 7305673+76 30 16 1.51 492 24.16  
## 18 6203059 7305863+75 60 16 1.52 983 24.32  
## 19 6203059 7305669+77 30 15 7.62 524 114.30  
## 20 6203059 7343137+72 30 32 6.74 246 215.68  
## 21 6203059 7305820+79 30 6 2.86 1310 17.16  
## 22 6203059 7305674+76 30 16 1.51 492 24.16  
## 23 6203059 7305824+73 30 30 2.56 262 76.80  
## 24 6203059 7326393+74 30 12 1.75 655 21.00  
## 25 6203059 7305833+73 30 12 2.02 655 24.24  
## 26 6203059 7306017+79 6 9 1.51 175 13.59  
## 27 6203059 7306018+79 6 9 6.62 175 59.58  
## 28 6203059 7313982+31 6 12 9.67 131 116.04  
## 29 6203059 7305867+76 60 48 1.58 328 75.84  
## 30 6203059 7305971+75 6 15 3.32 105 49.80  
## 31 6203059 7305972+75 6 15 3.13 105 46.95  
## 32 6203059 7343138+74 30 32 1.96 246 62.72  
## 33 6203059 7305823+73 30 30 4.05 262 121.50  
## 34 6203059 7326394+74 30 12 2.96 655 35.52  
## 35 6203059 7313981+31 6 12 5.68 131 68.16  
## 36 6203060 7305667+74 12 45 1.51 70 67.95  
## 37 6203060 7306023+78 30 15 2.13 524 31.95  
## 38 6203060 7306024+78 30 15 2.15 524 32.25  
## 39 6203060 7326409+74 30 11 1.87 715 20.57  
## 40 6203060 7326825+72 30 18 1.68 437 30.24  
## 41 6203060 7305817+74 60 16 4.55 983 72.80  
## 42 6203060 7306013+73 6 16 1.53 99 24.48  
## 43 6203060 7306037+79 6 20 3.82 79 76.40  
## 44 6203060 7306009+75 6 13 3.62 121 47.06  
## 45 6203060 7306010+75 6 13 2.13 121 27.69  
## 46 6203060 7326826+72 30 18 1.69 437 30.42  
## 47 6203060 7306014+73 6 16 1.70 99 27.20  
## 48 6203060 7306038+30 6 20 2.28 79 45.60  
## 49 6203061 7306034+77 6 30 2.11 53 63.30  
## 50 6203061 7328209+73 30 16 1.87 492 29.92  
## 51 6203061 7305977+74 24 36 1.54 175 55.44  
## 52 6203061 7305832+73 30 30 1.50 262 45.00  
## 53 6203061 7306026+75 6 20 1.62 79 32.40  
## 54 6203061 7306033+77 6 30 2.53 53 75.90  
## 55 6203061 7328210+73 30 16 1.76 492 28.16  
## 56 6203061 7306025+75 6 20 1.85 79 37.00  
## 57 6203061 7355006+72 30 36 3.08 219 110.88  
## 58 6203061 7346813+73 60 30 2.45 524 73.50  
## 59 6203061 7313474+73 30 8 2.82 983 22.56  
## 60 6203061 7355005+72 30 36 1.67 219 60.12  
## 61 6203062 7332421+76 30 5 3.45 1572 17.25  
## 62 6203062 7332422+76 30 5 1.57 1572 7.85  
## ordering\_cost eoq.min eoq.max b\_sorting b\_not\_sorting  
## 1 50 97 540 198 297  
## 2 50 108 599 198 297  
## 3 45 57 333 396 297  
## 4 45 57 333 396 297  
## 5 60 124 631 594 396  
## 6 65 59 287 596 794  
## 7 65 56 274 596 794  
## 8 65 196 960 596 794  
## 9 65 36 179 596 794  
## 10 65 72 354 596 794  
## 11 65 283 1386 596 794  
## 12 65 40 198 596 794  
## 13 65 60 293 596 794  
## 14 80 143 636 594 396  
## 15 80 177 787 594 396  
## 16 80 186 825 594 396  
## 17 80 181 802 594 396  
## 18 80 254 1130 594 396  
## 19 80 86 381 594 396  
## 20 80 43 190 594 396  
## 21 80 349 1553 594 396  
## 22 80 181 802 594 396  
## 23 80 74 328 594 396  
## 24 80 223 993 594 396  
## 25 80 208 924 594 396  
## 26 80 144 638 594 396  
## 27 80 69 305 594 396  
## 28 80 43 189 594 396  
## 29 80 83 370 594 396  
## 30 80 58 258 594 396  
## 31 80 60 266 594 396  
## 32 80 79 352 594 396  
## 33 80 59 261 594 396  
## 34 80 172 763 594 396  
## 35 80 55 246 594 396  
## 36 75 39 180 396 297  
## 37 75 157 719 396 297  
## 38 75 156 715 396 297  
## 39 75 228 1046 396 297  
## 40 75 147 675 396 297  
## 41 75 142 652 396 297  
## 42 75 78 357 396 297  
## 43 75 39 180 396 297  
## 44 75 62 285 396 297  
## 45 75 81 371 396 297  
## 46 75 147 673 396 297  
## 47 75 74 339 396 297  
## 48 75 51 234 396 297  
## 49 50 29 161 198 297  
## 50 50 128 714 198 297  
## 51 50 56 313 198 297  
## 52 50 76 425 198 297  
## 53 50 49 275 198 297  
## 54 50 26 147 198 297  
## 55 50 132 736 198 297  
## 56 50 46 257 198 297  
## 57 50 44 247 198 297  
## 58 50 84 470 198 297  
## 59 50 209 1162 198 297  
## 60 50 60 336 198 297  
## 61 70 357 1692 594 396  
## 62 70 529 2508 594 396

E.g Using box\_ID 6203060, sorted by length , Also for material ID 7305667+74 with therefore,

*1.* Number of boxes in a lane:

*2.* How many lanes part will occupy if you order some number of boxes

$$lane\_i=q\_i \cdot \frac {b\_i^{-1}}{rack\_{length}}=214 \cdot \frac {297}{6000}=10.593= 11 \space lanes $$

lane <- ceiling(product\_data$eoq \* (product\_data$b\_not\_sorting/rack\_length))  
lane

## numeric(0)

1. Do we meet the capacity contraint?

*3.* Total number of lanes constraints

Collapsing Rack 4 levels in a rack and joining the 8 racks to become 1 level.

Total rack width available:

rack\_total\_width <-51200 #rack\_width \* 4 \* 8  
rack\_total\_width #56000

## [1] 51200

lane.min <- ceiling(product\_data$eoq.min \* product\_data$b\_not\_sorting / rack\_length)  
lane.max <- ceiling(product\_data$eoq.max \* product\_data$b\_not\_sorting / rack\_length)  
  
# total rack width   
  
rack\_total\_width <- 51200 # rack\_width \* 4 \* 8

Overflow in mm

relative overflow in %

# overflow in mm  
sum(lane.min\*product\_data$b\_sorting) - rack\_total\_width

## [1] 218694

sum(lane.max\*product\_data$b\_sorting) - rack\_total\_width

## [1] 1173092

# relative overflow in %  
(sum(lane.min\*product\_data$b\_sorting) - rack\_total\_width)/rack\_total\_width\*100

## [1] 427.1367

(sum(lane.max\*product\_data$b\_sorting) - rack\_total\_width)/rack\_total\_width\*100

## [1] 2291.195

Please note the coefficients are are different for every ith item except racklength which is the same for all items.Also the left and right hand side of the equation need to be in mm.

#get the with of the lanes in mm  
rack\_width\_occupied=sum(lane.min\*product\_data$b\_sorting)  
rack\_width\_occupied

## [1] 269894

if(rack\_width\_occupied <= rack\_total\_width){  
 print(paste0("Capacity constraint fulfilled: ", rack\_width\_occupied, "<=",rack\_total\_width))  
   
}else {  
 violated <- rack\_width\_occupied - rack\_total\_width  
 print(paste0("Capacity constraint violated by: ",violated ))  
}

## [1] "Capacity constraint violated by: 218694"

As seen above contraint was voilated by , this means that we ordered too much, therefore we need to optimize

Using our example to get number of the total width i.e (summation of lanes) in : Summation of lanes can simply be:

Using equation (1) to prove this concept.

$$214 \cdot \frac {396 \cdot 297}{6000}=4194.828 \\ b\_i= \frac {4194.828}{10.593}=396$$

1. Adjust Q by reducing it:

no of orders for part .

average inventory for part

## Constrained EOQ optimization

Subject to:

library(ROI)

## ROI: R Optimization Infrastructure

## Registered solver plugins: nlminb, alabama, deoptimr, deoptim, glpk, quadprog, symphony.

## Default solver: auto.

library(ROI.plugin.alabama)  
  
n <- length(product\_data$`material ID`) #number of materials  
cori <- product\_data$ordering\_cost #ordering cost for each items  
cord <- 1500 #ordering cost whenever there is an order  
dei <- product\_data$demand\_per\_year  
h<-0.10  
box\_cost <- product\_data$box\_cost #pri  
  
# objective function --> I dropped the 1500\*62 as it is not decision relevant  
obj.fun <- function(q, d= dei, c.or = cori, c.h = h\*box\_cost ) (sum((dei/q)\*c.or)+ sum(c.h\*(q/2)))  
# benchmarks  
obj.fun(product\_data$eoq.max)

## [1] 79279.71

obj.fun(product\_data$eoq.min)

## [1] 31501.53

# constraint function --> also contains the ceiling of lanes and a sum was missing  
const.fun <- function(q, bns = product\_data$b\_not\_sorting, bs = product\_data$b\_sorting, rl = rack\_length) {  
 sum(bs \* ceiling( bns \* q / rl))  
 }  
  
const.fun(product\_data$eoq.max)

## [1] 1224292

const.fun(product\_data$eoq.min)

## [1] 269894

# try to figure out a freasible starting solution  
const.fun(product\_data$eoq.min/10) - rack\_total\_width

## [1] -10386

qopt <- OP(  
 objective = F\_objective(F=obj.fun ,n=n),  
 types = rep("C",n),  
 bounds = V\_bound(ub= product\_data$eoq.min , lb= rep(1, n)),  
 constraints = F\_constraint(F=const.fun,  
 dir="<=",  
 rhs = rack\_total\_width)  
)  
  
#This shows that minimum EOQ is too big and therefore will not meet the rack space constraint.  
const.fun(min(product\_data$eoq.max))#366272

## [1] 313556

const.fun(min(product\_data$eoq.max)/8)# 61428 still > 56000

## [1] 52716

const.fun(min(product\_data$eoq.max)/8.7)#52716 < 56000

## [1] 52716

const.fun(min(product\_data$eoq.max)/10)# 33098 < 56000 is much lower but this can still be the starting value

## [1] 33098

round(min(product\_data$eoq.max)/10) #15

## [1] 15

copt\_sol <- ROI\_solve(qopt, start = rep(min(product\_data$eoq.max)/10,n), solver = "alabama" )  
# always check whether the algorithm converged  
copt\_sol# The objective value is: 101625.9

## Optimal solution found.  
## The objective value is: 1.092799e+05

# solution  
copt\_sol$solution #vector of optimal Quantity that meets the space constraints and minimizes the Obj function.

## [1] 19.21277 19.98082 18.10408 18.10408 20.07066 16.50759 19.87768 20.22267  
## [9] 18.02317 20.07752 20.34807 16.36283 18.50350 20.28845 15.09718 20.37517  
## [17] 20.21815 19.93476 20.25461 18.49918 15.51115 20.21815 19.03311 20.33883  
## [25] 20.33375 18.30819 18.16358 17.33315 19.59133 17.14746 17.15822 18.92858  
## [33] 18.91119 20.31552 17.49555 16.37103 20.20202 20.20152 20.19630 20.09364  
## [41] 19.85988 17.04693 16.50265 17.29302 17.36417 20.09317 17.03609 16.63235  
## [49] 15.59037 19.53891 17.19712 18.05721 16.17151 15.52704 19.54321 16.15005  
## [57] 17.44414 19.59295 20.38314 17.61382 14.94486 14.83263

round(copt\_sol$solution)

## [1] 19 20 18 18 20 17 20 20 18 20 20 16 19 20 15 20 20 20 20 18 16 20 19 20 20  
## [26] 18 18 17 20 17 17 19 19 20 17 16 20 20 20 20 20 17 17 17 17 20 17 17 16 20  
## [51] 17 18 16 16 20 16 17 20 20 18 15 15

copt\_sol$objval #101625.9

## [1] 109279.9

const.fun(copt\_sol$solution)#55884

## [1] 51132

const.fun(floor(copt\_sol$solution))#55092

## [1] 50340

obj.fun(copt\_sol$solution)#101625.9

## [1] 109279.9

obj.fun(floor(copt\_sol$solution))#104639.7 rounded q values

## [1] 112079.8

removing waste of 150 for every 32 levels

## No constraint

rack\_total\_width <- 51200  
qopt1 <- OP(  
 objective = F\_objective(F=obj.fun ,n=n),  
 types = rep("C",n),  
 bounds = V\_bound(ub= product\_data$eoq.min , lb= rep(1, n)),  
)  
  
copt\_sol1 <- ROI\_solve(qopt1, start = rep(min(product\_data$eoq.max)/10,n), solver = "alabama" )  
# always check whether the algorithm converged  
copt\_sol1$objval

## [1] 31505.11

## Compare results

Remember, was dropped from the objective function as it is not decision relevant. \* *Total costs using EOQ* This has no constraint.

obj.fun <- function(q, d= dei, c.or = cori, c.h = h\*box\_cost ) (1500\*62)+(sum((dei/q)\*c.or)+ sum(c.h\*(q/2)))  
obj.fun(product\_data$eoq.max)#172279.7

## [1] 172279.7

* *Total costs with Capacity constraint Optimized*

const.fun(copt\_sol$solution)#51132

## [1] 51132

const.fun(floor(copt\_sol$solution))#50340 we rounded down to avoid exceeding capacity

## [1] 50340

obj.fun(copt\_sol$solution)#202279.9

## [1] 202279.9

obj.fun(floor(copt\_sol$solution))#205079.8 rounded q values

## [1] 205079.8

* *Total costs with no Capacity constraint Optimized* This is using EOQ.min

obj.fun(copt\_sol1$solution)#124505.1

## [1] 124505.1

obj.fun(floor(copt\_sol1$solution))#124508.6 rounded q values

## [1] 124508.6

Optimum costs with no capacity constraint using EOQ.max considers is €172279.7, with no capacity constraint optimized using EOQ.min is €124508.6, with capacity constraint, it costs €205079.8 This cost is expected to be higher when capacity constraint is included, as limited capacity will not allow to take advantage of cost savings.

# demand vs quantity  
  
product\_data$`demand per day`

## [1] 30 360 24 24 30 6 30 30 24 24 24 6 24 30 30 30 30 60 30  
## [20] 30 30 30 30 30 30 6 6 6 60 6 6 30 30 30 6 12 30 30  
## [39] 30 30 60 6 6 6 6 30 6 6 6 30 24 30 6 6 30 6 30  
## [58] 60 30 30 30 30

product\_data$demand\_per\_year #

## [1] 437 1474 315 315 562 79 605 605 262 786 786 79 262 655 1310  
## [16] 524 492 983 524 246 1310 492 262 655 655 175 175 131 328 105  
## [31] 105 246 262 655 131 70 524 524 715 437 983 99 79 121 121  
## [46] 437 99 79 53 492 175 262 79 53 492 79 219 524 983 219  
## [61] 1572 1572

floor(copt\_sol1$solution)# optimum quantities with no capacity constraint same as eoq.min

## [1] 96 107 56 56 123 58 55 195 35 71 282 39 59 142 175 185 180 253 85  
## [20] 42 348 180 73 222 207 143 68 42 82 57 59 78 58 171 54 38 156 155  
## [39] 227 146 141 77 38 61 80 146 73 50 28 127 55 75 48 25 131 45 43  
## [58] 83 208 59 356 465

floor(copt\_sol$solution) # optimum quantities with capacity constraint

## [1] 19 19 18 18 20 16 19 20 18 20 20 16 18 20 15 20 20 19 20 18 15 20 19 20 20  
## [26] 18 18 17 19 17 17 18 18 20 17 16 20 20 20 20 19 17 16 17 17 20 17 16 15 19  
## [51] 17 18 16 15 19 16 17 19 20 17 14 14

Plot EOQ cost vs cost

cost function per item

#EOQ cost  
  
#with no constraint  
paste("with no capacity constraint, cost is €", round(obj.fun(product\_data$eoq.max)))#172280

## [1] "with no capacity constraint, cost is \200 172280"

#with no capacity constraint optimized.  
#paste("with no capacity constraint optimized, cost is €",round(obj.fun(copt\_sol1$solution)))# eoq.min  
  
#with constraint  
paste("with capacity constraint, cost is €",round(obj.fun(copt\_sol$solution))) #202280

## [1] "with capacity constraint, cost is \200 202280"

paste("It costs €",round(obj.fun(copt\_sol$solution))-round(obj.fun(product\_data$eoq.max))," extra due to lack of capacity")

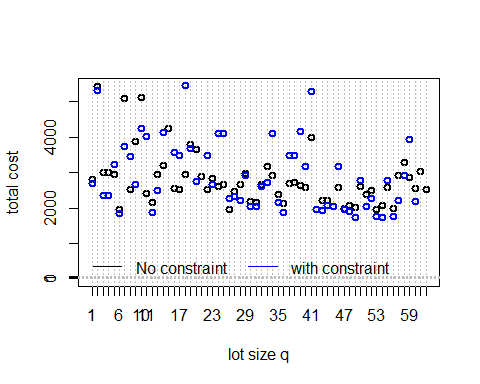
## [1] "It costs \200 30000 extra due to lack of capacity"

The above result shows that If there is no capacity constraint, the cost will be lower.

x.val <- seq.int(1,62, length.out = 62)  
  
cost.fun <- function(q, d= dei, c.or = cori, c.h = h\*box\_cost ){  
 (1500 +(dei/q)\*c.or)+ (c.h\*(q/2))  
}   
  
y.vec.no.const <- cost.fun(q=product\_data$eoq.max)  
y.vec.yes.const<- cost.fun(copt\_sol$solution)  
  
paste(length(which((round(y.vec.yes.const)>=round(y.vec.no.const))==TRUE))," Items costs more when capacity becomes a problem")

## [1] "26 Items costs more when capacity becomes a problem"

#par(mfrow=c(1,1))  
{   
 plot(x.val,round(y.vec.no.const), xlab ="lot size q", ylab = "total cost", type = "p", lwd =2, ylim=c(12,max(y.vec.no.const)), xaxp=c(1, 10, 1) )  
 axis(1, 1:62)  
axis(2, 1:62)  
abline(h=1:62, v=1:62, col="gray", lty=3)  
  
points(x=x.val,y=round(y.vec.yes.const),lwd=2, col="blue")  
  
legend("bottomleft",lty = c(1,1),col = c("black","blue"), legend = c("No constraint","with constraint"), bty="n" , horiz = T)  
}



* Notice that 26 blue dots which represents the more expensive items when capacity is accounted for.
* Also the notice , for most items where the blue is larger than black (non constraint items)(the variance is much more) the difference is much larger. the reverse is the case when black is larger than blue.

# Joint Ordering (JO)

Using Joint replenishment problem

Assumptions:

* One supplier with outbound storage.
* products
* Demand rates:
* Stock\_holding cost rates:
* Specific setup costs:
* General setup costs:
* Cycle time of product :

n <- length(product\_data$`box ID`)  
c.or0 <- cord  
c.or <- product\_data$ordering\_cost  
#H.vec <- 0.5 \* dei \* c.sh  
# based on the previous definition, I think this should be the vector of holding cost multipliers  
H.vec <- 0.1 \* product\_data$box\_cost \* dei \* 0.5

## Objective Function with no capacity constraint

* basic cycle time
* Holding cost multiplier are $H\_0 \space and \space H\_i$

$$C(m\_i B)=\sum\_{i=0}^n (\frac {c\_i^{or}}{m\_i \cdot B}+H\_i \cdot m\_i \cdot B) == \frac {c\_o^{or}}{B}+ \sum\_{i=1}^n (\frac {c\_i^{or}}{m\_i \cdot B}+H\_i \cdot m\_i \cdot B) \\ subject \space to: \quad \quad \quad \quad \quad \quad \quad \quad \quad \\ m\_i \ge m\_0 \quad \forall i >0 \\m\_i \in \mathbb N \qquad \forall i $$

–> an important side note is that the JRP is basically a quite standard EOQ problem which uses the substitution , i.e., . Then, you substitute ans result in the JRP (adding the common ordering cost).

jrp.obj.fun <- function(m , B, cor, Hvec, cor0) cor0/B + sum(cor/B/m) + sum(Hvec\*B\*m)

* *Cycle Time*

we determine and

# calculate cycle times  
T.vec <- sqrt(c.or/H.vec)  
# order products  
reo.id <- order(T.vec)  
T.vec <- T.vec[reo.id]  
c.or <- c.or[reo.id]  
H.vec <- H.vec[reo.id]  
  
cost\_cycle.mat <- t(cbind(c.or,H.vec,T.vec))  
costs.cycle <- data.frame(cost\_cycle.mat)  
colnames(costs.cycle) <- reo.id   
rownames(costs.cycle) <- c( "$c\_i^{or}$" ,"$H\_i$","$T\_i$" )  
kable(costs.cycle ,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 14 | 5 | 1 | 33 | 61 | 13 | 29 | 18 | 50 | 34 | 21 | 55 | 60 | 23 | 52 | 38 | 37 | 25 | 39 | 51 | 32 | 8 | 28 | 46 | 62 | 40 | 24 | 16 | 11 | 17 | 22 | 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 26 |
|  | 50.0000000 | 65.0000000 | 65.0000000 | 80.0000000 | 65.0000000 | 75.0000000 | 50.0000000 | 80.0000000 | 80.0000000 | 45.0000000 | 45.0000000 | 50.0000000 | 50.0000000 | 80.0000000 | 60.0000000 | 50.0000000 | 80.0000000 | 70.0000000 | 65.0000000 | 80.0000000 | 80.000000 | 50.0000000 | 80.0000000 | 80.0000000 | 50.0000000 | 50.0000000 | 80.0000000 | 50.0000000 | 75.0000000 | 75.0000000 | 80.0000000 | 75.0000000 | 50.0000000 | 80.0000000 | 65.00000 | 80.0000000 | 75.0000000 | 70.0000000 | 75.0000000 | 80.0000000 | 80.0000000 | 65.000000 | 80.0000000 | 80.0000000 | 80.0000000 | 80.000000 | 75.0000000 | 50.0000000 | 65.0000000 | 75.0000000 | 50.0000000 | 80.0000000 | 75.0000000 | 80.0000000 | 50.0000000 | 50.0000000 | 75.0000000 | 75.0000000 | 65.0000000 | 75.0000000 | 75.0000000 | 80.0000000 |
|  | 9386.4320000 | 7724.8080000 | 7652.6450000 | 4374.0900000 | 3364.0800000 | 3578.1200000 | 1925.7000000 | 2994.6600000 | 2652.8640000 | 1382.8500000 | 1382.8500000 | 1214.1360000 | 1108.8240000 | 1678.1100000 | 1239.2100000 | 1014.7140000 | 1591.6500000 | 1355.8500000 | 1254.4560000 | 1243.7760000 | 1195.328000 | 736.0320000 | 1163.2800000 | 1123.9800000 | 692.7360000 | 658.3140000 | 1006.0800000 | 589.5000000 | 844.9500000 | 837.0900000 | 793.8600000 | 735.3775000 | 485.1000000 | 771.4560000 | 621.33500 | 760.0620000 | 664.6770000 | 617.0100000 | 660.7440000 | 687.7500000 | 636.6600000 | 503.040000 | 594.3360000 | 594.3360000 | 521.3250000 | 446.448000 | 301.7800000 | 201.1350000 | 248.8500000 | 284.7130000 | 167.7450000 | 261.4500000 | 237.8250000 | 246.4875000 | 146.1500000 | 127.9800000 | 180.1200000 | 167.5245000 | 118.5000000 | 134.6400000 | 121.1760000 | 118.9125000 |
|  | 0.0729852 | 0.0917303 | 0.0921618 | 0.1352387 | 0.1390028 | 0.1447782 | 0.1611353 | 0.1634448 | 0.1736551 | 0.1803926 | 0.1803926 | 0.2029324 | 0.2123507 | 0.2183407 | 0.2200408 | 0.2219797 | 0.2241926 | 0.2272182 | 0.2276297 | 0.2536144 | 0.258703 | 0.2606374 | 0.2622424 | 0.2667876 | 0.2686588 | 0.2755932 | 0.2819868 | 0.2912347 | 0.2979306 | 0.2993261 | 0.3174483 | 0.3193563 | 0.3210476 | 0.3220249 | 0.32344 | 0.3244296 | 0.3359118 | 0.3368239 | 0.3369101 | 0.3410591 | 0.3544796 | 0.359464 | 0.3668842 | 0.3668842 | 0.3917335 | 0.423311 | 0.4985232 | 0.4985873 | 0.5110788 | 0.5132477 | 0.5459592 | 0.5531599 | 0.5615674 | 0.5697017 | 0.5849053 | 0.6250488 | 0.6452822 | 0.6691007 | 0.7406235 | 0.7463518 | 0.7867239 | 0.8202217 |

library(kableExtra)  
# calculate T^2 and cumu. cost shares  
res.mat <- t(cbind(c.or/H.vec,(c.or0 + cumsum(c.or))/cumsum(H.vec)))  
df <- data.frame(res.mat)  
colnames(df) <- reo.id   
rownames(df) <- c( "$T\_i^2$" ,"$T^C$" )  
kable(df,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 14 | 5 | 1 | 33 | 61 | 13 | 29 | 18 | 50 | 34 | 21 | 55 | 60 | 23 | 52 | 38 | 37 | 25 | 39 | 51 | 32 | 8 | 28 | 46 | 62 | 40 | 24 | 16 | 11 | 17 | 22 | 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 26 |
|  | 0.0053268 | 0.0084144 | 0.0084938 | 0.0182895 | 0.0193218 | 0.0209607 | 0.0259646 | 0.0267142 | 0.0301561 | 0.0325415 | 0.0325415 | 0.0411815 | 0.0450928 | 0.0476727 | 0.0484179 | 0.0492750 | 0.0502623 | 0.0516281 | 0.0518153 | 0.0643203 | 0.0669272 | 0.0679318 | 0.0687711 | 0.0711756 | 0.0721776 | 0.0759516 | 0.0795165 | 0.0848176 | 0.0887626 | 0.0895961 | 0.1007734 | 0.1019884 | 0.1030715 | 0.1037000 | 0.1046135 | 0.1052546 | 0.1128368 | 0.1134503 | 0.1135084 | 0.1163213 | 0.1256558 | 0.1292144 | 0.1346040 | 0.1346040 | 0.1534551 | 0.1791922 | 0.2485254 | 0.2485893 | 0.2612015 | 0.2634232 | 0.2980715 | 0.3059858 | 0.3153579 | 0.3245601 | 0.3421143 | 0.3906860 | 0.4163891 | 0.4476957 | 0.5485232 | 0.5570410 | 0.6189344 | 0.6727636 |
|  | 0.1651320 | 0.0943824 | 0.0678407 | 0.0604023 | 0.0561503 | 0.0526605 | 0.0513079 | 0.0495115 | 0.0483353 | 0.0478503 | 0.0473943 | 0.0472359 | 0.0471872 | 0.0472033 | 0.0472325 | 0.0472718 | 0.0473595 | 0.0474636 | 0.0475596 | 0.0479182 | 0.0483013 | 0.0485419 | 0.0489263 | 0.0493275 | 0.0495786 | 0.0498512 | 0.0503126 | 0.0506241 | 0.0511114 | 0.0515925 | 0.0521687 | 0.0527035 | 0.0530577 | 0.0536178 | 0.0540681 | 0.0546150 | 0.0551540 | 0.0556507 | 0.0561738 | 0.0567346 | 0.0573244 | 0.0578072 | 0.0584117 | 0.0590069 | 0.0596495 | 0.0603421 | 0.0610762 | 0.0615624 | 0.0622009 | 0.0629345 | 0.0634384 | 0.0642460 | 0.0650042 | 0.0658139 | 0.0663241 | 0.0668477 | 0.0676400 | 0.0684395 | 0.0691528 | 0.0699751 | 0.0708066 | 0.0716999 |

$$B=\sqrt \frac {\sum\_{i=0}^n \frac {c\_i^{or}}{m\_i}}{\sum\_{i=0}^n m\_i \cdot H\_i} $$

# identify break  
which(res.mat[1,] > res.mat[2,])# break occured after 13

## [1] 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38  
## [26] 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62

id.comb <- min(which(res.mat[1,] > res.mat[2,])) - 1 #3  
id.comb

## [1] 13

# calculate B  
B <- min(T.vec) #0.01955164  
# solution with m - integers #######################  
m.vec.int <- round(T.vec/B,0)  
  
m.vec.int[1:id.comb] <- 1   
# re-optimize B for fixed m.vec  
B.int <- sqrt(sum(c.or/m.vec.int)/sum(m.vec.int\*H.vec))#0.02014868  
# total cost   
c.cost.int <- jrp.obj.fun( m = m.vec.int, B=B.int, cor = c.or,Hvec=H.vec, cor0 = c.or0)#192606  
df <- data.frame(rbind(round(T.vec/B,2), round(T.vec/B), m.vec.int))  
colnames(df) <- reo.id   
rownames(df) <- c("$m\_i=\\frac{T\_i}{B}$" ,"$[m\_i]$", "$[\\tilde{m}\_i]$" )  
kable(df,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 14 | 5 | 1 | 33 | 61 | 13 | 29 | 18 | 50 | 34 | 21 | 55 | 60 | 23 | 52 | 38 | 37 | 25 | 39 | 51 | 32 | 8 | 28 | 46 | 62 | 40 | 24 | 16 | 11 | 17 | 22 | 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 26 |
|  | 1 | 1.26 | 1.26 | 1.85 | 1.9 | 1.98 | 2.21 | 2.24 | 2.38 | 2.47 | 2.47 | 2.78 | 2.91 | 2.99 | 3.01 | 3.04 | 3.07 | 3.11 | 3.12 | 3.47 | 3.54 | 3.57 | 3.59 | 3.66 | 3.68 | 3.78 | 3.86 | 3.99 | 4.08 | 4.1 | 4.35 | 4.38 | 4.4 | 4.41 | 4.43 | 4.45 | 4.6 | 4.61 | 4.62 | 4.67 | 4.86 | 4.93 | 5.03 | 5.03 | 5.37 | 5.8 | 6.83 | 6.83 | 7 | 7.03 | 7.48 | 7.58 | 7.69 | 7.81 | 8.01 | 8.56 | 8.84 | 9.17 | 10.15 | 10.23 | 10.78 | 11.24 |
|  | 1 | 1.00 | 1.00 | 2.00 | 2.0 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.0 | 4.00 | 4.00 | 4.0 | 4.00 | 4.00 | 4.00 | 5.0 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 6.0 | 7.00 | 7.00 | 7 | 7.00 | 7.00 | 8.00 | 8.00 | 8.00 | 8.00 | 9.00 | 9.00 | 9.00 | 10.00 | 10.00 | 11.00 | 11.00 |
|  | 1 | 1.00 | 1.00 | 1.00 | 1.0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.0 | 4.00 | 4.00 | 4.0 | 4.00 | 4.00 | 4.00 | 5.0 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 6.0 | 7.00 | 7.00 | 7 | 7.00 | 7.00 | 8.00 | 8.00 | 8.00 | 8.00 | 9.00 | 9.00 | 9.00 | 10.00 | 10.00 | 11.00 | 11.00 |

Reoptimizing basic cycle time yields and the total cost are 4.99433410^{4}

The order quantities are given by multiplying the cycle times with demand rates , i.e.  such that

dei <- dei[reo.id]  
df <- data.frame(rbind( m.vec.int, round(m.vec.int\*B, 2), round(m.vec.int\*B\*dei, 2) ))  
colnames(df) <- reo.id   
rownames(df) <- c("$[\\tilde{m}\_i]$", "$T\_i$", "$q\_i$" )  
kable(df,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 14 | 5 | 1 | 33 | 61 | 13 | 29 | 18 | 50 | 34 | 21 | 55 | 60 | 23 | 52 | 38 | 37 | 25 | 39 | 51 | 32 | 8 | 28 | 46 | 62 | 40 | 24 | 16 | 11 | 17 | 22 | 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 26 |
|  | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 4.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 5.00 | 6.00 | 7.00 | 7.00 | 7.00 | 7.00 | 7.00 | 8.00 | 8.00 | 8.00 | 8.00 | 9.00 | 9.00 | 9.00 | 10.00 | 10.00 | 11.00 | 11.0 |
|  | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.07 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.29 | 0.36 | 0.36 | 0.36 | 0.36 | 0.36 | 0.36 | 0.36 | 0.36 | 0.36 | 0.44 | 0.51 | 0.51 | 0.51 | 0.51 | 0.51 | 0.58 | 0.58 | 0.58 | 0.58 | 0.66 | 0.66 | 0.66 | 0.73 | 0.73 | 0.80 | 0.8 |
|  | 107.58 | 57.37 | 44.16 | 95.61 | 19.12 | 71.74 | 38.24 | 38.24 | 17.95 | 22.99 | 22.99 | 15.98 | 71.74 | 143.42 | 123.05 | 95.68 | 57.37 | 344.20 | 57.37 | 71.82 | 286.98 | 143.63 | 191.22 | 382.44 | 143.63 | 63.94 | 76.49 | 76.49 | 152.98 | 152.98 | 191.22 | 208.74 | 51.09 | 71.82 | 176.62 | 38.24 | 159.47 | 573.66 | 159.47 | 239.03 | 191.22 | 286.83 | 179.54 | 179.54 | 63.86 | 57.37 | 40.36 | 27.08 | 40.36 | 61.82 | 27.08 | 61.31 | 40.87 | 61.31 | 46.13 | 51.89 | 51.89 | 79.48 | 57.66 | 72.26 | 79.48 | 140.5 |

–> Now the question is: Is this feasible? –> no:

q.vec <- m.vec.int\*B\*dei  
paste(round(c.cost.int)," is total cost with no capacity constraint")# 49943

## [1] "49943 is total cost with no capacity constraint"

const.fun(q.vec) > rack\_total\_width

## [1] TRUE

When you try to optimize the JRP directly, without introducing the substitution you need to update the JRP function as follows

library(ROI)  
jrp.obj.fun2 <- function(Tvec, cor, Hvec, cor0) sum(cor0/Tvec) + sum(cor/Tvec) + sum(Hvec\*Tvec)  
# you need to initialize the parameters in the objective function --> Tvec is to be optimized over, the rest is given  
jrp.obj.fun2 <- function(Tvec, cor=c.or, Hvec = H.vec, cor0 = c.or0) sum(cor0/Tvec) + sum(cor/Tvec) + sum(Hvec\*Tvec)  
  
qopt2 <- OP(  
 objective = F\_objective(F=jrp.obj.fun2 , n=62),  
 types = rep("C",n),  
 bounds = V\_bound(ub= rep(30, 62), lb= rep(.001,62))  
)   
  
jrp.obj.fun2(m.vec.int\*B)

## [1] 511643.4

jrp.obj.fun2(rep(.00001,62))

## [1] 9724500001

jrp.obj.fun2(rep(15,62))

## [1] 1208367

# you don't need Alabama here as there is no constraint. Basically you can also optimize this problem by taking derivatives  
copt\_sol2 <- ROI\_solve(qopt2, start = m.vec.int\*B)  
copt\_sol2

## Optimal solution found.  
## The objective value is: 1.519523e+05

copt\_sol2$solution

## [1] 0.4063670 0.4501050 0.4522240 0.6010157 0.6820617 0.6634549 0.8971623  
## [8] 0.7263596 0.7717417 1.0570060 1.0570072 1.1298829 1.1823300 0.9703281  
## [15] 1.1219884 1.2359369 0.9963324 1.0760803 1.1169429 1.1270896 1.1497041  
## [22] 1.4511717 1.1654254 1.1856357 1.4958263 1.5344552 1.2531816 1.6215258  
## [29] 1.3652893 1.3716838 1.4107755 1.4634696 1.7875233 1.4311173 1.5870533  
## [36] 1.4418001 1.5393302 1.5951506 1.5439084 1.5157012 1.5753462 1.7638244  
## [43] 1.6304591 1.6304814 1.7409141 1.8812131 2.2845095 2.7760107 2.5077432  
## [50] 2.3519777 3.0397243 2.4582625 2.5734214 2.5317848 3.2565722 3.4800606  
## [57] 2.9570114 3.0661930 3.6340240 3.4201775 3.6051845 3.6450557

m.vec.int\*B\*dei

## [1] 107.58017 57.36636 44.15604 95.61060 19.12212 71.74444 38.24424  
## [8] 38.24424 17.95436 22.99033 22.99033 15.98376 71.74444 143.41590  
## [15] 123.05303 95.68358 57.36636 344.19815 57.36636 71.81743 286.97776  
## [22] 143.63485 191.22119 382.44239 143.63485 63.93502 76.48848 76.48848  
## [29] 152.97695 152.97695 191.22119 208.73764 51.08963 71.81743 176.62416  
## [36] 38.24424 159.47264 573.66358 159.47264 239.02649 191.22119 286.83179  
## [43] 179.54356 179.54356 63.86204 57.36636 40.36081 27.07750 40.36081  
## [50] 61.81845 27.07750 61.30756 40.87171 61.30756 46.12664 51.89247  
## [57] 51.89247 79.48087 57.65830 72.25534 79.48087 140.49649

paste(round(copt\_sol2$objval), "Total cost with no Constraint for T")

## [1] "151952 Total cost with no Constraint for T"

## Including capacity constraint

–> You should integrate the shelf capacity constraint –> I also recommmend to stick with the basic period approach

This is formally defined as

Thus, with the substitution above, the left-hand side changes to

Thus, you can change the constraint function quite easily

# we reinitialize the vectors  
n <- length(product\_data$`box ID`)  
c.or0 <- cord  
c.or <- product\_data$ordering\_cost  
dei <- product\_data$demand\_per\_year  
H.vec <- 0.1 \* product\_data$box\_cost \* dei \* 0.5  
  
  
bnotsort <- product\_data$b\_not\_sorting  
bsort <- product\_data$b\_sorting  
copt\_sol.solution <- copt\_sol$solution  
  
  
  
# calculate cycle times  
T.vec <- sqrt(c.or/H.vec)  
# order products  
reo.id <- order(T.vec)  
T.vec <- T.vec[reo.id]  
c.or <- c.or[reo.id]  
H.vec <- H.vec[reo.id]  
bnotsort <- bnotsort[reo.id]  
bsort <- bsort[reo.id]  
copt\_sol.solution <- copt\_sol.solution[reo.id]  
dei <- dei[reo.id]  
  
const.fun2 <- function(m, B = B.start, y= dei, bns = bnotsort, bs = bsort, rl = rack\_length) {  
 sum(bs \* ceiling( bns \* m\*B\*y / rl))  
 }

Then, you can use the original JRP function and update the model by including the capacity constraint. Therefore I recommend that you fix to some appropriate value based on the feasible solution above

#rack\_total\_width<-51200  
  
#library(ROI.plugin.deoptim)  
  
T.feas <- copt\_sol.solution/dei  
B.start <- min(T.feas)  
  
jrp.obj.fun <- function(m , B = B.start, cor = c.or, Hvec = H.vec, cor0 = c.or0) cor0/B + sum(cor/B/m) + sum(Hvec\*B\*m)  
jrp.obj.fun(m= rep(1, 62))

## [1] 609625.9

const.fun2(m= rep(1, 62))

## [1] 28330

qopt3 <- OP(  
 objective = F\_objective(F=jrp.obj.fun ,n=n),  
 # now integer decision variables  
 types = rep("C",n),  
 bounds = V\_bound(li = 1:n, ui = 1:n, ub= rep(50, n) , lb= rep(1, n)),  
 constraints = F\_constraint(F=const.fun2,  
 dir="<=",  
 rhs = rack\_total\_width)  
)  
# Good starting point essential --> m = T.feas/B.start  
copt\_sol3 <- ROI\_solve(qopt3, start = T.feas/B.start , solver = "alabama") # "deoptimr"  
  
copt\_sol3

## Optimal solution found.  
## The objective value is: 2.682538e+05

copt\_sol3$solution

## [1] 1.436648 2.707211 3.482128 1.221403 7.290618 2.141202 3.962808  
## [8] 4.096633 7.969883 6.091166 6.091166 8.441896 2.197617 3.282782  
## [15] 3.784948 4.659539 7.649835 1.007566 7.484917 6.330304 2.149274  
## [22] 4.208911 3.287162 1.254894 4.209837 8.524009 7.699154 7.304386  
## [29] 4.085895 4.085996 3.290112 2.993645 10.414829 8.154875 3.542564  
## [36] 14.022989 4.873057 1.000000 4.873171 3.290934 4.121016 2.743690  
## [43] 4.355227 4.355227 11.000134 14.154378 22.139151 31.048969 21.951577  
## [50] 15.146764 31.175616 17.307923 24.786343 17.318781 21.666120 21.694921  
## [57] 22.313149 15.209087 22.145777 18.237660 18.249267 11.087707

jrp.obj.fun(m= copt\_sol3$solution)

## [1] 268253.8

const.fun2(m= copt\_sol3$solution) <= rack\_total\_width

## [1] TRUE

# not feasible  
const.fun2(round(copt\_sol3$solution)) <= rack\_total\_width

## [1] FALSE

# feasible  
const.fun2(floor(copt\_sol3$solution)) <= rack\_total\_width

## [1] TRUE

# potential starting solution  
m.start <- floor(copt\_sol3$solution) # multiplier m  
q.start <- ceiling(m.start \* B.start \* dei) # order quantity q (in boxes, rounded up)  
q.start

## [1] 14 15 18 13 18 19 15 20 17 18 18 17 19 19 16 17 18 15 18 19 19 19 19 13 19  
## [26] 17 18 18 20 20 19 14 17 19 18 18 17 15 17 19 20 15 19 19 19 18 17 16 16 18  
## [51] 16 17 16 17 16 16 17 18 17 17 17 19

#Order according to m in ascending order  
#reo.id <- order(m.start)  
#dei <- dei[reo.id]  
#m.start <- m.start[reo.id]  
  
df2 <- data.frame(rbind( m.start, round(m.start\*B.start, 2), ceiling(m.start\*B.start\*dei) ))  
colnames(df2) <- reo.id   
rownames(df2) <- c("$[\\tilde{m}\_i]$", "$T\_i$", "$q\_i$" )  
kable(df2,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 14 | 5 | 1 | 33 | 61 | 13 | 29 | 18 | 50 | 34 | 21 | 55 | 60 | 23 | 52 | 38 | 37 | 25 | 39 | 51 | 32 | 8 | 28 | 46 | 62 | 40 | 24 | 16 | 11 | 17 | 22 | 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 26 |
|  | 1.00 | 2.00 | 3.00 | 1.00 | 7.00 | 2.00 | 3.00 | 4.00 | 7.00 | 6.00 | 6.00 | 8.00 | 2.00 | 3.00 | 3.00 | 4.00 | 7.00 | 1.00 | 7.00 | 6.00 | 2.00 | 4.00 | 3.00 | 1.00 | 4.00 | 8.00 | 7.00 | 7.00 | 4.00 | 4.00 | 3.00 | 2.00 | 10.00 | 8.00 | 3.00 | 14.00 | 4.00 | 1.00 | 4.00 | 3.00 | 4.00 | 2.00 | 4.00 | 4.00 | 11.0 | 14.00 | 22.00 | 31.00 | 21.0 | 15.00 | 31.00 | 17.00 | 24.00 | 17.00 | 21.0 | 21.0 | 22.00 | 15.00 | 22.00 | 18.00 | 18.00 | 11.0 |
|  | 0.01 | 0.02 | 0.03 | 0.01 | 0.07 | 0.02 | 0.03 | 0.04 | 0.07 | 0.06 | 0.06 | 0.08 | 0.02 | 0.03 | 0.03 | 0.04 | 0.07 | 0.01 | 0.07 | 0.06 | 0.02 | 0.04 | 0.03 | 0.01 | 0.04 | 0.08 | 0.07 | 0.07 | 0.04 | 0.04 | 0.03 | 0.02 | 0.09 | 0.08 | 0.03 | 0.13 | 0.04 | 0.01 | 0.04 | 0.03 | 0.04 | 0.02 | 0.04 | 0.04 | 0.1 | 0.13 | 0.21 | 0.29 | 0.2 | 0.14 | 0.29 | 0.16 | 0.23 | 0.16 | 0.2 | 0.2 | 0.21 | 0.14 | 0.21 | 0.17 | 0.17 | 0.1 |
|  | 14.00 | 15.00 | 18.00 | 13.00 | 18.00 | 19.00 | 15.00 | 20.00 | 17.00 | 18.00 | 18.00 | 17.00 | 19.00 | 19.00 | 16.00 | 17.00 | 18.00 | 15.00 | 18.00 | 19.00 | 19.00 | 19.00 | 19.00 | 13.00 | 19.00 | 17.00 | 18.00 | 18.00 | 20.00 | 20.00 | 19.00 | 14.00 | 17.00 | 19.00 | 18.00 | 18.00 | 17.00 | 15.00 | 17.00 | 19.00 | 20.00 | 15.00 | 19.00 | 19.00 | 19.0 | 18.00 | 17.00 | 16.00 | 16.0 | 18.00 | 16.00 | 17.00 | 16.00 | 17.00 | 16.0 | 16.0 | 17.00 | 18.00 | 17.00 | 17.00 | 17.00 | 19.0 |

library(gtools)  
df2<-df2[mixedorder(colnames(df2))]  
df2

## 1 2 3 4 5 6 7 8 9 10  
## $[\\tilde{m}\_i]$ 4.00 1.00 6.00 6.00 3.00 22.00 3.00 3.00 7.00 2.00  
## $T\_i$ 0.04 0.01 0.06 0.06 0.03 0.21 0.03 0.03 0.07 0.02  
## $q\_i$ 17.00 14.00 18.00 18.00 16.00 17.00 18.00 18.00 18.00 15.00  
## 11 12 13 14 15 16 17 18 19 20  
## $[\\tilde{m}\_i]$ 2.00 21.0 7.00 3.00 1.00 4.00 4.00 2.00 4.00 7.00  
## $T\_i$ 0.02 0.2 0.07 0.03 0.01 0.04 0.04 0.02 0.04 0.07  
## $q\_i$ 15.00 16.0 18.00 19.00 13.00 20.00 19.00 19.00 20.00 17.00  
## 21 22 23 24 25 26 27 28 29 30  
## $[\\tilde{m}\_i]$ 1.00 4.00 7.00 3.00 3.00 11.0 11.0 14.00 6.00 17.00  
## $T\_i$ 0.01 0.04 0.07 0.03 0.03 0.1 0.1 0.13 0.06 0.16  
## $q\_i$ 13.00 19.00 18.00 19.00 19.00 19.0 19.0 18.00 19.00 17.00  
## 31 32 33 34 35 36 37 38 39 40  
## $[\\tilde{m}\_i]$ 17.00 8.00 7.00 3.00 14.00 24.00 4.00 4.00 2.00 4.00  
## $T\_i$ 0.16 0.08 0.07 0.03 0.13 0.23 0.04 0.04 0.02 0.04  
## $q\_i$ 17.00 19.00 18.00 19.00 18.00 16.00 20.00 20.00 14.00 17.00  
## 41 42 43 44 45 46 47 48 49 50  
## $[\\tilde{m}\_i]$ 2.00 18.00 22.00 15.00 15.00 4.00 18.00 22.00 31.00 4.00  
## $T\_i$ 0.02 0.17 0.21 0.14 0.14 0.04 0.17 0.21 0.29 0.04  
## $q\_i$ 19.00 17.00 17.00 18.00 18.00 17.00 17.00 17.00 16.00 19.00  
## 51 52 53 54 55 56 57 58 59 60  
## $[\\tilde{m}\_i]$ 10.00 7.00 21.0 31.00 4.00 21.0 8.00 3.00 2.00 8.00  
## $T\_i$ 0.09 0.07 0.2 0.29 0.04 0.2 0.08 0.03 0.02 0.08  
## $q\_i$ 17.00 18.00 16.0 16.00 19.00 16.0 17.00 15.00 19.00 17.00  
## 61 62  
## $[\\tilde{m}\_i]$ 1.00 1.00  
## $T\_i$ 0.01 0.01  
## $q\_i$ 15.00 15.00

q.start<- as.vector(df2[3,1:62])  
q.start

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24  
## $q\_i$ 17 14 18 18 16 17 18 18 18 15 15 16 18 19 13 20 19 19 20 17 13 19 18 19  
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48  
## $q\_i$ 19 19 19 18 19 17 17 19 18 19 18 16 20 20 14 17 19 17 17 18 18 17 17 17  
## 49 50 51 52 53 54 55 56 57 58 59 60 61 62  
## $q\_i$ 16 19 17 18 16 16 19 16 17 15 19 17 15 15

Afterwards, you have to find a layout scheme such that a precise shelf layout results.

As you rightly point out you need to calculate the number of lanes required for each product given a certain order quantity (integer number of boxes):

whereby is the number of boxes per lane dedicated to product . I.e.,

. Thus, for the solution of the JRP we have:

l.start <- ceiling(q.start/floor(rack\_length/product\_data$b\_not\_sorting))  
rownames(l.start) <- "lanes"  
l.start

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27  
## lanes 1 1 1 1 2 3 3 3 3 3 3 3 3 2 1 2 2 2 2 2 1 2 2 2 2 2 2  
## 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51  
## lanes 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## 52 53 54 55 56 57 58 59 60 61 62  
## lanes 1 1 1 1 1 1 1 1 1 1 1

Now, these lanes have to be assigned to the levels of the shelves. Therefore, we first need to determine the types of lanes required. The lanes are just described by their width. Due to the safety margins we should round the lane width to full centimeter. I.e., we need to assign lanes with the following widths

unique(round(product\_data$b\_sorting/100)\*100)

## [1] 200 400 600

Now comes the tricky part: We have to decide how many lanes of a certain width should be assigned for each level. Luckily, there is only a small number of useful patterns of lanes per level. To be precise there are 9 efficient patterns to arrange these 3 lane types (assuming we use each level exhaustively):

patterns <- rbind(  
c(8,0,0),  
c(0,4,0),  
c(0,1,2),  
c(2,0,2),  
c(6,1,0),  
c(4,2,0),  
c(2,3,0),  
c(3,1,1),  
c(1,2,1)  
)  
colnames(patterns) <- c("200","400","600")  
  
patterns

## 200 400 600  
## [1,] 8 0 0  
## [2,] 0 4 0  
## [3,] 0 1 2  
## [4,] 2 0 2  
## [5,] 6 1 0  
## [6,] 4 2 0  
## [7,] 2 3 0  
## [8,] 3 1 1  
## [9,] 1 2 1

Let indicate the number of lanes of type associated to pattern . Now we need to assess the number of lanes per required of each type. As outlined above, we know the number of lanes per product , thus we can deduce the demand for lane type () by summing up the for each lane type, i.e., :

# assign lane width as names  
names(l.start) <- round(product\_data$b\_sorting/100)\*100   
  
# number of items with lane types 200, 400, 600 that is demand for each lane type  
ld <- c(sum(l.start[names(l.start) == "200"]),  
sum(l.start[names(l.start) == "400"]),  
sum(l.start[names(l.start) == "600"]))  
ld

## [1] 14 15 70

Need to reduce 70 units to 64

## Illustration

#m.start  
  
product\_data$Order\_frequency\_M<- unlist(df2[1,1:62])  
product\_data$Lot\_size\_q <- unlist(q.start)  
product\_data

## box ID material ID demand per day pieces/box price demand\_per\_year box\_cost  
## 1 3103147 7348808+73 30 18 2.58 437 46.44  
## 2 3103147 7346592+72 360 64 1.99 1474 127.36  
## 3 3104147 7455691+71 24 20 4.39 315 87.80  
## 4 3104147 7455692+71 24 20 4.39 315 87.80  
## 5 3106410 7305865+77 30 14 3.15 562 44.10  
## 6 3108210 7306005+73 6 20 1.50 79 30.00  
## 7 3108210 7308013+79 30 13 19.46 605 252.98  
## 8 3108210 7308014+79 30 13 1.58 605 20.54  
## 9 3108210 7455688+72 24 24 10.70 262 256.80  
## 10 3108210 7455633+73 24 8 24.57 786 196.56  
## 11 3108210 7455634+73 24 8 1.60 786 12.80  
## 12 3108210 7306006+73 6 20 3.15 79 63.00  
## 13 3108210 7455687+72 24 24 3.99 262 95.76  
## 14 6203059 7305834+73 30 12 4.27 655 51.24  
## 15 6203059 7305819+79 30 6 11.13 1310 66.78  
## 16 6203059 7305670+77 30 15 1.62 524 24.30  
## 17 6203059 7305673+76 30 16 1.51 492 24.16  
## 18 6203059 7305863+75 60 16 1.52 983 24.32  
## 19 6203059 7305669+77 30 15 7.62 524 114.30  
## 20 6203059 7343137+72 30 32 6.74 246 215.68  
## 21 6203059 7305820+79 30 6 2.86 1310 17.16  
## 22 6203059 7305674+76 30 16 1.51 492 24.16  
## 23 6203059 7305824+73 30 30 2.56 262 76.80  
## 24 6203059 7326393+74 30 12 1.75 655 21.00  
## 25 6203059 7305833+73 30 12 2.02 655 24.24  
## 26 6203059 7306017+79 6 9 1.51 175 13.59  
## 27 6203059 7306018+79 6 9 6.62 175 59.58  
## 28 6203059 7313982+31 6 12 9.67 131 116.04  
## 29 6203059 7305867+76 60 48 1.58 328 75.84  
## 30 6203059 7305971+75 6 15 3.32 105 49.80  
## 31 6203059 7305972+75 6 15 3.13 105 46.95  
## 32 6203059 7343138+74 30 32 1.96 246 62.72  
## 33 6203059 7305823+73 30 30 4.05 262 121.50  
## 34 6203059 7326394+74 30 12 2.96 655 35.52  
## 35 6203059 7313981+31 6 12 5.68 131 68.16  
## 36 6203060 7305667+74 12 45 1.51 70 67.95  
## 37 6203060 7306023+78 30 15 2.13 524 31.95  
## 38 6203060 7306024+78 30 15 2.15 524 32.25  
## 39 6203060 7326409+74 30 11 1.87 715 20.57  
## 40 6203060 7326825+72 30 18 1.68 437 30.24  
## 41 6203060 7305817+74 60 16 4.55 983 72.80  
## 42 6203060 7306013+73 6 16 1.53 99 24.48  
## 43 6203060 7306037+79 6 20 3.82 79 76.40  
## 44 6203060 7306009+75 6 13 3.62 121 47.06  
## 45 6203060 7306010+75 6 13 2.13 121 27.69  
## 46 6203060 7326826+72 30 18 1.69 437 30.42  
## 47 6203060 7306014+73 6 16 1.70 99 27.20  
## 48 6203060 7306038+30 6 20 2.28 79 45.60  
## 49 6203061 7306034+77 6 30 2.11 53 63.30  
## 50 6203061 7328209+73 30 16 1.87 492 29.92  
## 51 6203061 7305977+74 24 36 1.54 175 55.44  
## 52 6203061 7305832+73 30 30 1.50 262 45.00  
## 53 6203061 7306026+75 6 20 1.62 79 32.40  
## 54 6203061 7306033+77 6 30 2.53 53 75.90  
## 55 6203061 7328210+73 30 16 1.76 492 28.16  
## 56 6203061 7306025+75 6 20 1.85 79 37.00  
## 57 6203061 7355006+72 30 36 3.08 219 110.88  
## 58 6203061 7346813+73 60 30 2.45 524 73.50  
## 59 6203061 7313474+73 30 8 2.82 983 22.56  
## 60 6203061 7355005+72 30 36 1.67 219 60.12  
## 61 6203062 7332421+76 30 5 3.45 1572 17.25  
## 62 6203062 7332422+76 30 5 1.57 1572 7.85  
## ordering\_cost eoq.min eoq.max b\_sorting b\_not\_sorting Order\_frequency\_M  
## 1 50 97 540 198 297 4  
## 2 50 108 599 198 297 1  
## 3 45 57 333 396 297 6  
## 4 45 57 333 396 297 6  
## 5 60 124 631 594 396 3  
## 6 65 59 287 596 794 22  
## 7 65 56 274 596 794 3  
## 8 65 196 960 596 794 3  
## 9 65 36 179 596 794 7  
## 10 65 72 354 596 794 2  
## 11 65 283 1386 596 794 2  
## 12 65 40 198 596 794 21  
## 13 65 60 293 596 794 7  
## 14 80 143 636 594 396 3  
## 15 80 177 787 594 396 1  
## 16 80 186 825 594 396 4  
## 17 80 181 802 594 396 4  
## 18 80 254 1130 594 396 2  
## 19 80 86 381 594 396 4  
## 20 80 43 190 594 396 7  
## 21 80 349 1553 594 396 1  
## 22 80 181 802 594 396 4  
## 23 80 74 328 594 396 7  
## 24 80 223 993 594 396 3  
## 25 80 208 924 594 396 3  
## 26 80 144 638 594 396 11  
## 27 80 69 305 594 396 11  
## 28 80 43 189 594 396 14  
## 29 80 83 370 594 396 6  
## 30 80 58 258 594 396 17  
## 31 80 60 266 594 396 17  
## 32 80 79 352 594 396 8  
## 33 80 59 261 594 396 7  
## 34 80 172 763 594 396 3  
## 35 80 55 246 594 396 14  
## 36 75 39 180 396 297 24  
## 37 75 157 719 396 297 4  
## 38 75 156 715 396 297 4  
## 39 75 228 1046 396 297 2  
## 40 75 147 675 396 297 4  
## 41 75 142 652 396 297 2  
## 42 75 78 357 396 297 18  
## 43 75 39 180 396 297 22  
## 44 75 62 285 396 297 15  
## 45 75 81 371 396 297 15  
## 46 75 147 673 396 297 4  
## 47 75 74 339 396 297 18  
## 48 75 51 234 396 297 22  
## 49 50 29 161 198 297 31  
## 50 50 128 714 198 297 4  
## 51 50 56 313 198 297 10  
## 52 50 76 425 198 297 7  
## 53 50 49 275 198 297 21  
## 54 50 26 147 198 297 31  
## 55 50 132 736 198 297 4  
## 56 50 46 257 198 297 21  
## 57 50 44 247 198 297 8  
## 58 50 84 470 198 297 3  
## 59 50 209 1162 198 297 2  
## 60 50 60 336 198 297 8  
## 61 70 357 1692 594 396 1  
## 62 70 529 2508 594 396 1  
## Lot\_size\_q  
## 1 17  
## 2 14  
## 3 18  
## 4 18  
## 5 16  
## 6 17  
## 7 18  
## 8 18  
## 9 18  
## 10 15  
## 11 15  
## 12 16  
## 13 18  
## 14 19  
## 15 13  
## 16 20  
## 17 19  
## 18 19  
## 19 20  
## 20 17  
## 21 13  
## 22 19  
## 23 18  
## 24 19  
## 25 19  
## 26 19  
## 27 19  
## 28 18  
## 29 19  
## 30 17  
## 31 17  
## 32 19  
## 33 18  
## 34 19  
## 35 18  
## 36 16  
## 37 20  
## 38 20  
## 39 14  
## 40 17  
## 41 19  
## 42 17  
## 43 17  
## 44 18  
## 45 18  
## 46 17  
## 47 17  
## 48 17  
## 49 16  
## 50 19  
## 51 17  
## 52 18  
## 53 16  
## 54 16  
## 55 19  
## 56 16  
## 57 17  
## 58 15  
## 59 19  
## 60 17  
## 61 15  
## 62 15

#m.start <- sort(unique(product\_data$Order\_frequency\_M))  
m.start <- product\_data$Order\_frequency\_M  
sort(unique(m.start)) # 21 unique values meaning 22 columns

## [1] 1 2 3 4 6 7 8 10 11 14 15 17 18 21 22 24 31

max(m.start)

## [1] 31

name.m<- paste("Mi=",sort(unique(m.start)), sep = "")  
  
cycle.mat <- matrix(0, nrow = max(m.start)+1,ncol = length(unique(m.start)))  
  
colnames(cycle.mat)<-name.m  
  
#cycle.mat  
# for first row  
i<- NULL  
j<-1  
  
  
 for(i in sort(unique(m.start))){  
   
 tmp <- which(m.start==i)  
 tmp<- paste(tmp,collapse = ",")  
   
 cycle.mat[1,name.m[j]] <- tmp  
 j<- j+1  
   
 }  
   
  
  
for (k in 1:max(m.start)+1) {  
 j<- 1  
 for(i in sort(unique(m.start))){  
 if((k-1)%%i==0){  
 tmp <- which(m.start==i)  
 tmp<- paste(tmp,collapse = ",")  
   
 #print(tmp)  
   
 cycle.mat[k,name.m[j]] <- tmp  
 j<- j+1   
 }else {  
 j<- j+1   
 }  
   
 }  
   
 }  
  
cycle.df <- as.data.frame(cycle.mat)   
cycle.df

## Mi=1 Mi=2 Mi=3  
## 1 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 2 2,15,21,61,62 0 0  
## 3 2,15,21,61,62 10,11,18,39,41,59 0  
## 4 2,15,21,61,62 0 5,7,8,14,24,25,34,58  
## 5 2,15,21,61,62 10,11,18,39,41,59 0  
## 6 2,15,21,61,62 0 0  
## 7 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 8 2,15,21,61,62 0 0  
## 9 2,15,21,61,62 10,11,18,39,41,59 0  
## 10 2,15,21,61,62 0 5,7,8,14,24,25,34,58  
## 11 2,15,21,61,62 10,11,18,39,41,59 0  
## 12 2,15,21,61,62 0 0  
## 13 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 14 2,15,21,61,62 0 0  
## 15 2,15,21,61,62 10,11,18,39,41,59 0  
## 16 2,15,21,61,62 0 5,7,8,14,24,25,34,58  
## 17 2,15,21,61,62 10,11,18,39,41,59 0  
## 18 2,15,21,61,62 0 0  
## 19 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 20 2,15,21,61,62 0 0  
## 21 2,15,21,61,62 10,11,18,39,41,59 0  
## 22 2,15,21,61,62 0 5,7,8,14,24,25,34,58  
## 23 2,15,21,61,62 10,11,18,39,41,59 0  
## 24 2,15,21,61,62 0 0  
## 25 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 26 2,15,21,61,62 0 0  
## 27 2,15,21,61,62 10,11,18,39,41,59 0  
## 28 2,15,21,61,62 0 5,7,8,14,24,25,34,58  
## 29 2,15,21,61,62 10,11,18,39,41,59 0  
## 30 2,15,21,61,62 0 0  
## 31 2,15,21,61,62 10,11,18,39,41,59 5,7,8,14,24,25,34,58  
## 32 2,15,21,61,62 0 0  
## Mi=4 Mi=6 Mi=7 Mi=8 Mi=10 Mi=11  
## 1 1,16,17,19,22,37,38,40,46,50,55 3,4,29 9,13,20,23,33,52 32,57,60 51 26,27  
## 2 0 0 0 0 0 0  
## 3 0 0 0 0 0 0  
## 4 0 0 0 0 0 0  
## 5 1,16,17,19,22,37,38,40,46,50,55 0 0 0 0 0  
## 6 0 0 0 0 0 0  
## 7 0 3,4,29 0 0 0 0  
## 8 0 0 9,13,20,23,33,52 0 0 0  
## 9 1,16,17,19,22,37,38,40,46,50,55 0 0 32,57,60 0 0  
## 10 0 0 0 0 0 0  
## 11 0 0 0 0 51 0  
## 12 0 0 0 0 0 26,27  
## 13 1,16,17,19,22,37,38,40,46,50,55 3,4,29 0 0 0 0  
## 14 0 0 0 0 0 0  
## 15 0 0 9,13,20,23,33,52 0 0 0  
## 16 0 0 0 0 0 0  
## 17 1,16,17,19,22,37,38,40,46,50,55 0 0 32,57,60 0 0  
## 18 0 0 0 0 0 0  
## 19 0 3,4,29 0 0 0 0  
## 20 0 0 0 0 0 0  
## 21 1,16,17,19,22,37,38,40,46,50,55 0 0 0 51 0  
## 22 0 0 9,13,20,23,33,52 0 0 0  
## 23 0 0 0 0 0 26,27  
## 24 0 0 0 0 0 0  
## 25 1,16,17,19,22,37,38,40,46,50,55 3,4,29 0 32,57,60 0 0  
## 26 0 0 0 0 0 0  
## 27 0 0 0 0 0 0  
## 28 0 0 0 0 0 0  
## 29 1,16,17,19,22,37,38,40,46,50,55 0 9,13,20,23,33,52 0 0 0  
## 30 0 0 0 0 0 0  
## 31 0 3,4,29 0 0 51 0  
## 32 0 0 0 0 0 0  
## Mi=14 Mi=15 Mi=17 Mi=18 Mi=21 Mi=22 Mi=24 Mi=31  
## 1 28,35 44,45 30,31 42,47 12,53,56 6,43,48 36 49,54  
## 2 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 0 0 0  
## 5 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0  
## 7 0 0 0 0 0 0 0 0  
## 8 0 0 0 0 0 0 0 0  
## 9 0 0 0 0 0 0 0 0  
## 10 0 0 0 0 0 0 0 0  
## 11 0 0 0 0 0 0 0 0  
## 12 0 0 0 0 0 0 0 0  
## 13 0 0 0 0 0 0 0 0  
## 14 0 0 0 0 0 0 0 0  
## 15 28,35 0 0 0 0 0 0 0  
## 16 0 44,45 0 0 0 0 0 0  
## 17 0 0 0 0 0 0 0 0  
## 18 0 0 30,31 0 0 0 0 0  
## 19 0 0 0 42,47 0 0 0 0  
## 20 0 0 0 0 0 0 0 0  
## 21 0 0 0 0 0 0 0 0  
## 22 0 0 0 0 12,53,56 0 0 0  
## 23 0 0 0 0 0 6,43,48 0 0  
## 24 0 0 0 0 0 0 0 0  
## 25 0 0 0 0 0 0 36 0  
## 26 0 0 0 0 0 0 0 0  
## 27 0 0 0 0 0 0 0 0  
## 28 0 0 0 0 0 0 0 0  
## 29 28,35 0 0 0 0 0 0 0  
## 30 0 0 0 0 0 0 0 0  
## 31 0 44,45 0 0 0 0 0 0  
## 32 0 0 0 0 0 0 0 49,54

rownames(cycle.df)<-1:nrow(cycle.df)  
  
kable(cycle.df,"html", row.names = T, escape = F) %>%  
 kable\_styling("striped")

Mi=1

Mi=2

Mi=3

Mi=4

Mi=6

Mi=7

Mi=8

Mi=10

Mi=11

Mi=14

Mi=15

Mi=17

Mi=18

Mi=21

Mi=22

Mi=24

Mi=31

1

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

1,16,17,19,22,37,38,40,46,50,55

3,4,29

9,13,20,23,33,52

32,57,60

51

26,27

28,35

44,45

30,31

42,47

12,53,56

6,43,48

36

49,54

2

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

3

2,15,21,61,62

10,11,18,39,41,59

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

4

2,15,21,61,62

0

5,7,8,14,24,25,34,58

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

2,15,21,61,62

10,11,18,39,41,59

0

1,16,17,19,22,37,38,40,46,50,55

0

0

0

0

0

0

0

0

0

0

0

0

0

6

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

7

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

0

3,4,29

0

0

0

0

0

0

0

0

0

0

0

0

8

2,15,21,61,62

0

0

0

0

9,13,20,23,33,52

0

0

0

0

0

0

0

0

0

0

0

9

2,15,21,61,62

10,11,18,39,41,59

0

1,16,17,19,22,37,38,40,46,50,55

0

0

32,57,60

0

0

0

0

0

0

0

0

0

0

10

2,15,21,61,62

0

5,7,8,14,24,25,34,58

0

0

0

0

0

0

0

0

0

0

0

0

0

0

11

2,15,21,61,62

10,11,18,39,41,59

0

0

0

0

0

51

0

0

0

0

0

0

0

0

0

12

2,15,21,61,62

0

0

0

0

0

0

0

26,27

0

0

0

0

0

0

0

0

13

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

1,16,17,19,22,37,38,40,46,50,55

3,4,29

0

0

0

0

0

0

0

0

0

0

0

0

14

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

15

2,15,21,61,62

10,11,18,39,41,59

0

0

0

9,13,20,23,33,52

0

0

0

28,35

0

0

0

0

0

0

0

16

2,15,21,61,62

0

5,7,8,14,24,25,34,58

0

0

0

0

0

0

0

44,45

0

0

0

0

0

0

17

2,15,21,61,62

10,11,18,39,41,59

0

1,16,17,19,22,37,38,40,46,50,55

0

0

32,57,60

0

0

0

0

0

0

0

0

0

0

18

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

30,31

0

0

0

0

0

19

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

0

3,4,29

0

0

0

0

0

0

0

42,47

0

0

0

0

20

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

21

2,15,21,61,62

10,11,18,39,41,59

0

1,16,17,19,22,37,38,40,46,50,55

0

0

0

51

0

0

0

0

0

0

0

0

0

22

2,15,21,61,62

0

5,7,8,14,24,25,34,58

0

0

9,13,20,23,33,52

0

0

0

0

0

0

0

12,53,56

0

0

0

23

2,15,21,61,62

10,11,18,39,41,59

0

0

0

0

0

0

26,27

0

0

0

0

0

6,43,48

0

0

24

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

25

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

1,16,17,19,22,37,38,40,46,50,55

3,4,29

0

32,57,60

0

0

0

0

0

0

0

0

36

0

26

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

27

2,15,21,61,62

10,11,18,39,41,59

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

28

2,15,21,61,62

0

5,7,8,14,24,25,34,58

0

0

0

0

0

0

0

0

0

0

0

0

0

0

29

2,15,21,61,62

10,11,18,39,41,59

0

1,16,17,19,22,37,38,40,46,50,55

0

9,13,20,23,33,52

0

0

0

28,35

0

0

0

0

0

0

0

30

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

31

2,15,21,61,62

10,11,18,39,41,59

5,7,8,14,24,25,34,58

0

3,4,29

0

0

51

0

0

44,45

0

0

0

0

0

0

32

2,15,21,61,62

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

49,54

* *Demand vs Quantity*

which is the capacity of a box for part . Demand in Years for item Demand per day in boxes for item . Demand per day in items for item $yb\_i= $

cycle time is number of days it takes before a new order.

E.g Item 4 has lot size quantity of 18 boxes, and daily demand in boxes is 1.2

$$ct= \frac {q}{yb\_i}= \frac {m\_i \cdot B \cdot y\_i}{yb\_i} \\ ct= \frac {18}{1.2}=15days$$

The example above shows that for item 4, it takes about 15 day to make another order.

another example, assuming we order 17 items and demand pay day in boxes is 1.6667 this mean it will take about 10 day to make another other.

# Demand per day in boxes  
#its still okay to leave the values in fraction as we are not ordering now.  
product\_data$deman\_per\_day\_boxes <- product\_data$`demand per day`/product\_data$`pieces/box`  
  
#This is number of days it takes before a new order "product\_data$Lot\_size\_q" is placed  
product\_data$cycle\_time\_in\_days <- (m.start\*B.start\*product\_data$demand\_per\_year)/product\_data$deman\_per\_day\_boxes

* *Convert Cycles to days*

#min(product\_data$cycle\_time\_in\_days)#2.472104 days is equivalent to M=1  
#sort(sort(unique(product\_data$Order\_frequency\_M)))  
#sort(unique(product\_data$Order\_frequency\_M))\*min(product\_data$cycle\_time\_in\_days)  
  
  
  
cycle.time <- data.frame(rbind(sort(unique(product\_data$Order\_frequency\_M))\*min(product\_data$cycle\_time\_in\_days) ))  
  
colnames(cycle.time) <- sort(unique(product\_data$Order\_frequency\_M))  
rownames(cycle.time) <- c("Days M." )  
  
kable(cycle.time,"html", row.names = T, escape = F) %>%  
 kable\_styling("striped")

1

2

3

4

6

7

8

10

11

14

15

17

18

21

22

24

31

Days M.

2.472104

4.944209

7.416314

9.888418

14.83263

17.30473

19.77684

24.72105

27.19315

34.60946

37.08157

42.02578

44.49788

51.91419

54.3863

59.33051

76.63524

* *Calculate demand after first cycle*

Assuming that demand All Quantity has been supplied for all items in the first cycle

reduce lane assignment while fulfilling demand

#cycle\_demand== Lot\_size\_q  
#cycle\_demand <- product\_data$cycle\_time\_in\_days\*product\_data$deman\_per\_day\_boxes  
  
#l.start2 <- (cycle\_demand \*product\_data$b\_not\_sorting)/rack\_length  
l.start2 <- ceiling(l.start)  
l.start2[5]<-1  
l.start2[10]<- 2  
l.start2[11]<-2  
l.start2[12]<- 2  
l.start2[20]<- 1  
l.start2[30]<-1  
  
#l.start2 <- ceiling(l.start)  
vec<-unlist(l.start2)  
  
product\_data$number\_of\_lanes <- vec  
  
l.start2

## 200 200 400 400 600 600 600 600 600 600 600 600 600 600 600 600 600 600  
## lanes 1 1 1 1 1 3 3 3 3 2 2 2 3 2 1 2 2 2  
## 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 400  
## lanes 2 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1  
## 400 400 400 400 400 400 400 400 400 400 400 400 200 200 200 200 200 200  
## lanes 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## 200 200 200 200 200 200 600 600  
## lanes 1 1 1 1 1 1 1 1

# assign lane width as names  
names(l.start2) <- round(product\_data$b\_sorting/100)\*100   
  
# number of items with lane types 200, 400, 600 that is demand for each lane type  
ld <- c(sum(l.start2[names(l.start2) == "200"]),  
sum(l.start2[names(l.start2) == "400"]),  
sum(l.start2[names(l.start2) == "600"]))  
ld

## [1] 14 15 64

* *Prove that demand is always met*

Using (T,S) policy

We are using periodic review as order intervals can be derived from order frequency for each of the items.

* Assuming demand is uniformly distributed thus the same quantity of demand repeats each time.
* Initial stock level for the 62 items are given based on lanes assigned
* Each Item has some form of order frequency which we derived cycle time from, going over a time period of 262 days. This represents order interval.
* Lead time is zero.
* At each order interval, we order up to which is the stock level based on lane assigned for each item meaning, lanes are filled.
* No Backorder
* Stock is filled to the capacity in period 1.

*Notations and Formulars*

On-hand stock S(t)

Outstanding orders O(t)

Backorders B(t)

Inventory level available units

for item 30, we changed the cycle time in days to 39 instead of 42.

#item 39  
product\_data$cycle\_time\_in\_days[30]<- 39

In the code below, Shows (T,S) Policy, how demand is met over 262 time period. Reason for TS policy is that we can not exceed capacity, thus we order upto Lane capacity of each item. Also each item has its cycle time in days which was derived from order frequency M.

#install.packages("writexl")  
#library("writexl")  
  
#write\_xlsx(product\_data,"C:\\Users\\...\\product\_data.xlsx")  
  
stock.level <- (product\_data$number\_of\_lanes\*rack\_length)/product\_data$b\_not\_sorting  
  
#product\_data$number\_of\_lanes  
floor(stock.level)

## [1] 20 20 20 20 15 22 22 22 22 15 15 15 22 30 15 30 30 30 30 15 15 30 30 30 30  
## [26] 30 30 30 30 15 30 30 30 30 30 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  
## [51] 20 20 20 20 20 20 20 20 20 20 15 15

Periodic.inventory <-function(item){  
   
 #Yt demand  
dem <- product\_data$deman\_per\_day\_boxes  
dem[item]  
  
#qt quantity  
quant <- product\_data$Lot\_size\_q  
quant[item]  
  
#Initial Stock level  
stock.level <- floor((product\_data$number\_of\_lanes\*rack\_length)/product\_data$b\_not\_sorting)  
stock.level[item]  
  
#inventory level  
#inventory.level <- inventory.level[t-1]- dem[t]  
  
period.mat <- matrix(0.00000, nrow = 3,ncol = 262)  
  
colnames(period.mat)<-1:262  
m <- product\_data$cycle\_time\_in\_days  
  
#period.mat[3,0]<- stock.level[1]  
t<- 1  
#for (t in 1:262) {  
while(t<=262){  
 if(item==12){  
   
 if(t==1){  
 period.mat[1,t]<-dem[item]  
 #period.mat[2,t]<- quant[item]  
 period.mat[3,t]<- stock.level[item] #stock.level[1]  
   
 }else {  
 period.mat[1,t]<-dem[item]  
 if(t %% floor(m[item])==0){  
 #b<-t+1  
 #print(t+1)  
 period.mat[2,t]<- quant[item]  
   
 # fill the capacity  
 period.mat[3,t]<- stock.level[item] #quant[item]+period.mat[3,t-1]-dem[item]  
 }else{  
 period.mat[2,t]<- 0  
 period.mat[3,t]<- period.mat[3,t-1]-dem[item]  
 }  
 }  
   
   
 }else if(item== 30){  
   
   
 if(t==1){  
 period.mat[1,t]<-dem[item]  
 #period.mat[2,t]<- quant[item]  
 period.mat[3,t]<- stock.level[item] #stock.level[1]  
   
 }else {  
 period.mat[1,t]<-dem[item]  
 if(t %% floor(m[item])==0){  
 #b<-t+1  
 #print(t+1)  
 period.mat[2,t]<- quant[item]  
   
 # fill the capacity  
 period.mat[3,t]<- stock.level[item] #quant[item]+period.mat[3,t-1]-dem[item]  
 }else{  
 period.mat[2,t]<- 0  
 period.mat[3,t]<- period.mat[3,t-1]-dem[item]  
 }  
 }  
   
   
 }else {  
   
   
 if(t==1){  
 period.mat[1,t]<-dem[item]  
 #period.mat[2,t]<- quant[item]  
 period.mat[3,t]<- stock.level[item] #stock.level[1]  
   
 }else {  
 period.mat[1,t]<-dem[item]  
 if(t %% floor(m[item])==1){  
 #b<-t+1  
 #print(t+1)  
 period.mat[2,t]<- quant[item]  
   
 # fill the capacity  
 period.mat[3,t]<- stock.level[item] #quant[item]+period.mat[3,t-1]-dem[item]  
 }else{  
 period.mat[2,t]<- 0  
 period.mat[3,t]<- period.mat[3,t-1]-dem[item]  
 }  
 }  
   
   
}  
  
 t=t+1  
   
 }  
   
   
 return (period.mat)  
   
}  
  
period.df <- as.data.frame(Periodic.inventory(2))  
row.names(period.df)<- c("yt","qt","lt")  
period.df

## 1 2 3 4 5 6 7 8 9 10 11  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 12 13 14 15 16 17 18 19 20 21 22  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 23 24 25 26 27 28 29 30 31 32 33  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 34 35 36 37 38 39 40 41 42 43 44  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 45 46 47 48 49 50 51 52 53 54 55  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 56 57 58 59 60 61 62 63 64 65 66  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 67 68 69 70 71 72 73 74 75 76 77  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 78 79 80 81 82 83 84 85 86 87 88  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 89 90 91 92 93 94 95 96 97 98 99  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 100 101 102 103 104 105 106 107 108 109 110  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 111 112 113 114 115 116 117 118 119 120 121  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 122 123 124 125 126 127 128 129 130 131 132  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 133 134 135 136 137 138 139 140 141 142 143  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 144 145 146 147 148 149 150 151 152 153 154  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 155 156 157 158 159 160 161 162 163 164 165  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 166 167 168 169 170 171 172 173 174 175 176  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 177 178 179 180 181 182 183 184 185 186 187  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 188 189 190 191 192 193 194 195 196 197 198  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 199 200 201 202 203 204 205 206 207 208 209  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 210 211 212 213 214 215 216 217 218 219 220  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 221 222 223 224 225 226 227 228 229 230 231  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 232 233 234 235 236 237 238 239 240 241 242  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375  
## 243 244 245 246 247 248 249 250 251 252 253  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000  
## lt 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000  
## 254 255 256 257 258 259 260 261 262  
## yt 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625 5.625  
## qt 0.000 14.000 0.000 14.000 0.000 14.000 0.000 14.000 0.000  
## lt 14.375 20.000 14.375 20.000 14.375 20.000 14.375 20.000 14.375

Ave.phy.inv <-c(vector(),1:62)  
alpha.sl <- c(vector(),1:62)  
for (i in 1:62) {  
 period.df <- as.data.frame(Periodic.inventory(i))  
 row.names(period.df)<- c("yt","qt","lt")  
 Ave.phy.inv[i]<- sum(period.df[3,][period.df[3,]>0])/length(period.df[3,])  
 alpha.sl[i] <- (length(period.df[3,][period.df[3,]>0])/length(period.df[3,])) \*100  
}  
  
ts.summary <- data.frame(rbind(Ave.phy.inv,alpha.sl))  
  
colnames(ts.summary)<- 1:62  
rownames(ts.summary)<-c("Average Inventory","Alpha Service Level")  
ts.summary

## 1 2 3 4 5 6  
## Average Inventory 13.35878 17.1875 12.2916 12.2916 8.620502 14.26069  
## Alpha Service Level 100.00000 100.0000 100.0000 100.0000 100.000000 100.00000  
## 7 8 9 10 11 12  
## Average Inventory 15.12977 15.12977 14.13359 10.5229 10.5229 7.467939  
## Alpha Service Level 100.00000 100.00000 100.00000 100.0000 100.0000 96.564885  
## 13 14 15 16 17 18  
## Average Inventory 14.13359 22.55725 12.5 22.03053 22.52863 24.40363  
## Alpha Service Level 100.00000 100.00000 100.0 100.00000 100.00000 100.00000  
## 19 20 21 22 23 24  
## Average Inventory 22.03053 7.625239 12.5 22.52863 22.13359 22.55725  
## Alpha Service Level 100.00000 94.274809 100.0 100.00000 100.00000 100.00000  
## 25 26 27 28 29 30  
## Average Inventory 22.55725 21.52672 21.52672 21.97901 21.97042 7.654198  
## Alpha Service Level 100.00000 100.00000 100.00000 100.00000 100.00000 98.091603  
## 31 32 33 34 35 36  
## Average Inventory 22.04427 21.66985 22.13359 22.55725 21.97901 12.70331  
## Alpha Service Level 100.00000 100.00000 100.00000 100.00000 100.00000 100.00000  
## 37 38 39 40 41 42  
## Average Inventory 12.03053 12.03053 15.92991 13.35878 14.40363 11.99761  
## Alpha Service Level 100.00000 100.00000 100.00000 100.00000 100.00000 100.00000  
## 43 44 45 46 47 48  
## Average Inventory 12.26069 11.78215 11.78215 13.35878 11.99761 12.26069  
## Alpha Service Level 100.00000 100.00000 100.00000 100.00000 100.00000 100.00000  
## 49 50 51 52 53 54  
## Average Inventory 12.94427 12.52863 12.38931 12.13359 12.40725 12.94427  
## Alpha Service Level 100.00000 100.00000 100.00000 100.00000 100.00000 100.00000  
## 55 56 57 58 59 60  
## Average Inventory 12.52863 12.40725 12.59542 14.0458 14.40363 12.59542  
## Alpha Service Level 100.00000 100.00000 100.00000 100.0000 100.00000 100.00000  
## 61 62  
## Average Inventory 12 12  
## Alpha Service Level 100 100

As seen from the result, only Items 12 and 30 has an service level of 98% of the result has 100% service level.

## Changes made on q

1. We assumed that there are 262 working days for the year 2020 according to <https://hr.uiowa.edu/pay/payroll-services/payroll-calendars/working-day-payroll-calendar-2020>
2. rack\_total\_width which was 56,000 mm, looking at the 9 patterns all of which have 150 mm in waste, meaning all 32 levels will have 150 mm waste each. Therefor thereby changing rack total width 51,200.
3. These changes then affects the values of q for all the 62 items involved.

To fulfill lane demands from the items are 15 lanes with with 200, 15 lanes with width 400 and 64 lanes with width 600. To fulfull this demand, only 2 patterns are needed.

patterns[3:4,]

## 200 400 600  
## [1,] 0 1 2  
## [2,] 2 0 2

Assign Patterns

fitted.pattern <- patterns[3:4,]  
# loop through 32 levels to assign patterns  
  
levels.pattern.mat <- matrix(0, nrow = 32,ncol = 3)  
  
for (i in 1:32) {  
 for (j in 1:3) {  
   
 if(i<= 16){  
 levels.pattern.mat[i,j] <- t(fitted.pattern[1,j])  
 } else {  
 levels.pattern.mat[i,j] <- t(fitted.pattern[2,j])  
 }  
 }  
}  
  
  
levels.pattern.mat

## [,1] [,2] [,3]  
## [1,] 0 1 2  
## [2,] 0 1 2  
## [3,] 0 1 2  
## [4,] 0 1 2  
## [5,] 0 1 2  
## [6,] 0 1 2  
## [7,] 0 1 2  
## [8,] 0 1 2  
## [9,] 0 1 2  
## [10,] 0 1 2  
## [11,] 0 1 2  
## [12,] 0 1 2  
## [13,] 0 1 2  
## [14,] 0 1 2  
## [15,] 0 1 2  
## [16,] 0 1 2  
## [17,] 2 0 2  
## [18,] 2 0 2  
## [19,] 2 0 2  
## [20,] 2 0 2  
## [21,] 2 0 2  
## [22,] 2 0 2  
## [23,] 2 0 2  
## [24,] 2 0 2  
## [25,] 2 0 2  
## [26,] 2 0 2  
## [27,] 2 0 2  
## [28,] 2 0 2  
## [29,] 2 0 2  
## [30,] 2 0 2  
## [31,] 2 0 2  
## [32,] 2 0 2

# confirm if lane demand is met.  
  
sum(levels.pattern.mat[1:32,1]) >= sum(ld[1]) #TRUE for 200

## [1] TRUE

sum(levels.pattern.mat[1:32,2]) >= sum(ld[2]) #TRUE for 400

## [1] TRUE

sum(levels.pattern.mat[1:32,3]) >= sum(ld[3]) #TRUE for 600

## [1] TRUE

# Analysis

## ABC Analysis

Procedure: consumption and price of material , then 1. Calculate value share of material

#remember box cost is demand times price  
value.share <- product\_data$box\_cost/sum(product\_data$box\_cost)  
#as.data.frame(value.share)

1. Order materials descendingly according to value share: $v\_1^´ v\_2^´ …v\_{|I|}^´ $

#cycle.time <- data.frame(rbind(sort(unique(product\_data$Order\_frequency\_M))\*min(product\_data$cycle\_time\_in\_days) ))  
  
#colnames(cycle.time) <- sort(unique(product\_data$Order\_frequency\_M))  
#rownames(cycle.time) <- c("Days M." )  
  
#kable(cycle.time,"html", row.names = T, escape = F) %>%  
# kable\_styling("striped")  
  
  
  
#ord.share<- data.frame(rbind(sort(value.share,decreasing=TRUE) ))   
#reoder.item <- order(value.share,decreasing = TRUE)  
#colnames(ord.share)<- reoder.item   
#ord.share  
#as.data.frame(ord.share)  
  
id.vec <- 1:62  
  
#id.vec.ord contains the indexes of materials in decreasing order.  
id.vec.ord <- id.vec[order(value.share, decreasing = T)]  
  
#val.vec.ord contains values of the materials in decreasing order  
val.vec.ord <- sort(value.share,decreasing = T)

1. Calculate cumulative ordered value share of each material:

cum.val.vec.ord <-cumsum(val.vec.ord)  
rel.cum.val.vec.ord <- cum.val.vec.ord / sum(val.vec.ord)  
  
rel.cum.val.vec.ord

## [1] 0.06670702 0.13242174 0.18844732 0.23950624 0.27258960 0.30415075  
## [7] 0.33429359 0.36398445 0.39278691 0.41766177 0.44046892 0.46327607  
## [13] 0.48322583 0.50307169 0.52278767 0.54248806 0.56158060 0.58049131  
## [19] 0.59819673 0.61584759 0.63319453 0.64963750 0.66600254 0.68229485  
## [25] 0.69791177 0.71338842 0.72778965 0.74109989 0.75403606 0.76626048  
## [31] 0.77845634 0.79051971 0.80236488 0.81405419 0.82550972 0.83512093  
## [37] 0.84434770 0.85276400 0.86114134 0.86944076 0.87734273 0.88519795  
## [43] 0.89299083 0.90076292 0.90807784 0.91527066 0.92233620 0.92869519  
## [49] 0.93501261 0.94132484 0.94762148 0.95389735 0.96017321 0.96603345  
## [55] 0.97148847 0.97683178 0.98216730 0.98664821 0.99110573 0.99463590  
## [61] 0.99796086 1.00000000

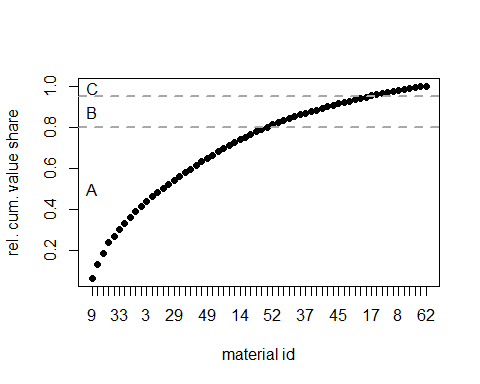
1. Categorize materials according to into classes A,B and C by class limits e.g 80,95,100

class.vec <- rep("A", n)  
class.vec[rel.cum.val.vec.ord > 0.8 & rel.cum.val.vec.ord <= 0.95 ] <- "B"  
class.vec[rel.cum.val.vec.ord > 0.95 ] <- "C"  
  
tab <- data.frame("ord.material.id" = id.vec.ord, "price (ord.)" = product\_data$box\_cost[order(value.share, decreasing = T)] , "demand (ord.)" = product\_data$deman\_per\_day\_boxes[order(value.share, decreasing = T)], "mat.values" = val.vec.ord, "cum.mat.values" = cum.val.vec.ord, "rel.cum.value.shares" = round(rel.cum.val.vec.ord \* 100, 1), "class" = class.vec )  
  
kable(tab, digits = c(0,2,0,2,2,1,0), caption = "ABC analysis results (values classified by boxes)", format = "pandoc")

ABC analysis results (values classified by boxes)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ord.material.id | price..ord.. | demand..ord.. | mat.values | cum.mat.values | rel.cum.value.shares | class |
| 9 | 256.80 | 1 | 0.07 | 0.07 | 6.7 | A |
| 7 | 252.98 | 2 | 0.07 | 0.13 | 13.2 | A |
| 20 | 215.68 | 1 | 0.06 | 0.19 | 18.8 | A |
| 10 | 196.56 | 3 | 0.05 | 0.24 | 24.0 | A |
| 2 | 127.36 | 6 | 0.03 | 0.27 | 27.3 | A |
| 33 | 121.50 | 1 | 0.03 | 0.30 | 30.4 | A |
| 28 | 116.04 | 0 | 0.03 | 0.33 | 33.4 | A |
| 19 | 114.30 | 2 | 0.03 | 0.36 | 36.4 | A |
| 57 | 110.88 | 1 | 0.03 | 0.39 | 39.3 | A |
| 13 | 95.76 | 1 | 0.02 | 0.42 | 41.8 | A |
| 3 | 87.80 | 1 | 0.02 | 0.44 | 44.0 | A |
| 4 | 87.80 | 1 | 0.02 | 0.46 | 46.3 | A |
| 23 | 76.80 | 1 | 0.02 | 0.48 | 48.3 | A |
| 43 | 76.40 | 0 | 0.02 | 0.50 | 50.3 | A |
| 54 | 75.90 | 0 | 0.02 | 0.52 | 52.3 | A |
| 29 | 75.84 | 1 | 0.02 | 0.54 | 54.2 | A |
| 58 | 73.50 | 2 | 0.02 | 0.56 | 56.2 | A |
| 41 | 72.80 | 4 | 0.02 | 0.58 | 58.0 | A |
| 35 | 68.16 | 0 | 0.02 | 0.60 | 59.8 | A |
| 36 | 67.95 | 0 | 0.02 | 0.62 | 61.6 | A |
| 15 | 66.78 | 5 | 0.02 | 0.63 | 63.3 | A |
| 49 | 63.30 | 0 | 0.02 | 0.65 | 65.0 | A |
| 12 | 63.00 | 0 | 0.02 | 0.67 | 66.6 | A |
| 32 | 62.72 | 1 | 0.02 | 0.68 | 68.2 | A |
| 60 | 60.12 | 1 | 0.02 | 0.70 | 69.8 | A |
| 27 | 59.58 | 1 | 0.02 | 0.71 | 71.3 | A |
| 51 | 55.44 | 1 | 0.01 | 0.73 | 72.8 | A |
| 14 | 51.24 | 2 | 0.01 | 0.74 | 74.1 | A |
| 30 | 49.80 | 0 | 0.01 | 0.75 | 75.4 | A |
| 44 | 47.06 | 0 | 0.01 | 0.77 | 76.6 | A |
| 31 | 46.95 | 0 | 0.01 | 0.78 | 77.8 | A |
| 1 | 46.44 | 2 | 0.01 | 0.79 | 79.1 | A |
| 48 | 45.60 | 0 | 0.01 | 0.80 | 80.2 | B |
| 52 | 45.00 | 1 | 0.01 | 0.81 | 81.4 | B |
| 5 | 44.10 | 2 | 0.01 | 0.83 | 82.6 | B |
| 56 | 37.00 | 0 | 0.01 | 0.84 | 83.5 | B |
| 34 | 35.52 | 2 | 0.01 | 0.84 | 84.4 | B |
| 53 | 32.40 | 0 | 0.01 | 0.85 | 85.3 | B |
| 38 | 32.25 | 2 | 0.01 | 0.86 | 86.1 | B |
| 37 | 31.95 | 2 | 0.01 | 0.87 | 86.9 | B |
| 46 | 30.42 | 2 | 0.01 | 0.88 | 87.7 | B |
| 40 | 30.24 | 2 | 0.01 | 0.89 | 88.5 | B |
| 6 | 30.00 | 0 | 0.01 | 0.89 | 89.3 | B |
| 50 | 29.92 | 2 | 0.01 | 0.90 | 90.1 | B |
| 55 | 28.16 | 2 | 0.01 | 0.91 | 90.8 | B |
| 45 | 27.69 | 0 | 0.01 | 0.92 | 91.5 | B |
| 47 | 27.20 | 0 | 0.01 | 0.92 | 92.2 | B |
| 42 | 24.48 | 0 | 0.01 | 0.93 | 92.9 | B |
| 18 | 24.32 | 4 | 0.01 | 0.94 | 93.5 | B |
| 16 | 24.30 | 2 | 0.01 | 0.94 | 94.1 | B |
| 25 | 24.24 | 2 | 0.01 | 0.95 | 94.8 | B |
| 17 | 24.16 | 2 | 0.01 | 0.95 | 95.4 | C |
| 22 | 24.16 | 2 | 0.01 | 0.96 | 96.0 | C |
| 59 | 22.56 | 4 | 0.01 | 0.97 | 96.6 | C |
| 24 | 21.00 | 2 | 0.01 | 0.97 | 97.1 | C |
| 39 | 20.57 | 3 | 0.01 | 0.98 | 97.7 | C |
| 8 | 20.54 | 2 | 0.01 | 0.98 | 98.2 | C |
| 61 | 17.25 | 6 | 0.00 | 0.99 | 98.7 | C |
| 21 | 17.16 | 5 | 0.00 | 0.99 | 99.1 | C |
| 26 | 13.59 | 1 | 0.00 | 0.99 | 99.5 | C |
| 11 | 12.80 | 3 | 0.00 | 1.00 | 99.8 | C |
| 62 | 7.85 | 6 | 0.00 | 1.00 | 100.0 | C |

# a plot  
plot(1:n, rel.cum.val.vec.ord, type="b", xaxt = "n", pch =16, xlab = "material id", ylab = "rel. cum. value share")  
axis(1, at = 1:n, labels = id.vec.ord)  
abline(h = c(.8,.95), lwd=2, lty=2, col="darkgrey")  
text(x = rep(1,3) , y = c(.5,.875,.99), labels = LETTERS[1:3] )



## IQR Analysis

Categorization by life cycle - Inventory quality ratio

Aim is to minimize the risk of obselensce.

* Life cycle of product is the change of demand over time.
* Variation of material with respect to it’s importance. Typically associated with ABC analysis.
* The higher the value of materials, the potential risk of obscelence situation. This risk increase if this are compared to it’s demand overstocked.

There are 262 days working in consideration, 5 working days in a week, $=52.4 $ that is 52 weeks in a year. So we assume that current stock level is equivelent to stocks that can fit lane assignments for each item and

* estimated demand value per period (week).
* The current stock value.
* The accepted turnover time of material then.

We will use these thresholds : A…2 per., B..4 per. and C..6 per.

1. Calculate active inventory
2. Inventory quatlity ratio $IQR\_i= $ the closer to 1, the better.

n <- 62  
set.seed(123456)  
ave.stock.vec <- floor(stock.level)  
  
o.vec <- as.factor(class.vec)  
levels(o.vec) <- c(2,4,6)  
o.vec <- as.numeric(as.character(o.vec))  
names(o.vec) <- id.vec.ord  
  
s.vec <- product\_data$box\_cost \*ave.stock.vec  
#paste("Stock ",s.vec)  
  
toval.vec <- o.vec[as.character(1:n)]\*(product\_data$deman\_per\_day\_boxes)\* product\_data$box\_cost  
#paste("active Inve ",toval.vec)  
  
a.vec <- apply(cbind(s.vec, toval.vec),1 , min)  
a.vec

## 1 2 3 4 5 6 7 8 9 10   
## 154.80 1432.80 210.72 210.72 378.00 36.00 1167.60 284.40 513.60 1179.36   
## 11 12 13 14 15 16 17 18 19 20   
## 192.00 37.80 191.52 256.20 667.80 194.40 271.80 364.80 457.20 404.40   
## 21 22 23 24 25 26 27 28 29 30   
## 257.40 271.80 153.60 315.00 242.40 54.36 79.44 116.04 189.60 39.84   
## 31 32 33 34 35 36 37 38 39 40   
## 37.56 117.60 243.00 355.20 68.16 36.24 255.60 258.00 336.60 201.60   
## 41 42 43 44 45 46 47 48 49 50   
## 546.00 36.72 45.84 43.44 51.12 202.80 40.80 54.72 25.32 224.40   
## 51 52 53 54 55 56 57 58 59 60   
## 73.92 180.00 38.88 30.36 211.20 44.40 184.80 294.00 451.20 100.20   
## 61 62   
## 258.75 117.75

iqr.vec2 <- toval.vec/s.vec  
e.vec <- s.vec - a.vec  
e.vec

## 1 2 3 4 5 6 7 8 9 10   
## 774.00 1114.40 1545.28 1545.28 283.50 624.00 4397.96 167.48 5136.00 1769.04   
## 11 12 13 14 15 16 17 18 19 20   
## 0.00 907.20 1915.20 1281.00 333.90 534.60 453.00 364.80 2971.80 2830.80   
## 21 22 23 24 25 26 27 28 29 30   
## 0.00 453.00 2150.40 315.00 484.80 353.34 1707.96 3365.16 2085.60 707.16   
## 31 32 33 34 35 36 37 38 39 40   
## 1370.94 1764.00 3402.00 710.40 1976.64 1322.76 383.40 387.00 74.80 403.20   
## 41 42 43 44 45 46 47 48 49 50   
## 910.00 452.88 1482.16 897.76 502.68 405.60 503.20 857.28 1240.68 374.00   
## 51 52 53 54 55 56 57 58 59 60   
## 1034.88 720.00 609.12 1487.64 352.00 695.60 2032.80 1176.00 0.00 1102.20   
## 61 62   
## 0.00 0.00

e.vec2 <- s.vec - toval.vec  
e.vec2

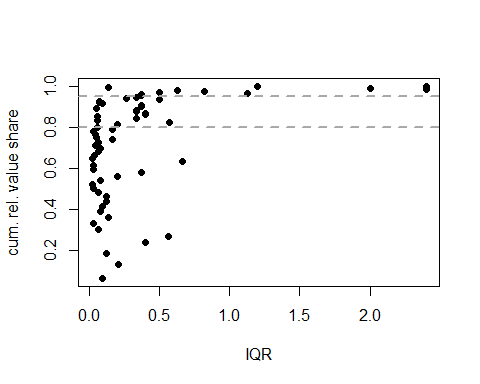
## 1 2 3 4 5 6 7 8 9 10   
## 774.00 1114.40 1545.28 1545.28 283.50 624.00 4397.96 167.48 5136.00 1769.04   
## 11 12 13 14 15 16 17 18 19 20   
## -38.40 907.20 1915.20 1281.00 333.90 534.60 453.00 364.80 2971.80 2830.80   
## 21 22 23 24 25 26 27 28 29 30   
## -257.40 453.00 2150.40 315.00 484.80 353.34 1707.96 3365.16 2085.60 707.16   
## 31 32 33 34 35 36 37 38 39 40   
## 1370.94 1764.00 3402.00 710.40 1976.64 1322.76 383.40 387.00 74.80 403.20   
## 41 42 43 44 45 46 47 48 49 50   
## 910.00 452.88 1482.16 897.76 502.68 405.60 503.20 857.28 1240.68 374.00   
## 51 52 53 54 55 56 57 58 59 60   
## 1034.88 720.00 609.12 1487.64 352.00 695.60 2032.80 1176.00 -56.40 1102.20   
## 61 62   
## -362.25 -164.85

tab <- data.frame("material.id" = 1:n, "stock.value" = round(s.vec,1), "to.value/act.inv" = round(toval.vec,1), "iqr" = round(iqr.vec2,1), "excess.inv" = round(e.vec2,1))  
  
kable(tab, digits = c(0,1,1,1,1), caption = "Life-cycle analysis for materials 1-62", format = "pandoc")

Life-cycle analysis for materials 1-62

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| material.id | stock.value | to.value.act.inv | iqr | excess.inv |
| 1 | 928.8 | 154.8 | 0.2 | 774.0 |
| 2 | 2547.2 | 1432.8 | 0.6 | 1114.4 |
| 3 | 1756.0 | 210.7 | 0.1 | 1545.3 |
| 4 | 1756.0 | 210.7 | 0.1 | 1545.3 |
| 5 | 661.5 | 378.0 | 0.6 | 283.5 |
| 6 | 660.0 | 36.0 | 0.1 | 624.0 |
| 7 | 5565.6 | 1167.6 | 0.2 | 4398.0 |
| 8 | 451.9 | 284.4 | 0.6 | 167.5 |
| 9 | 5649.6 | 513.6 | 0.1 | 5136.0 |
| 10 | 2948.4 | 1179.4 | 0.4 | 1769.0 |
| 11 | 192.0 | 230.4 | 1.2 | -38.4 |
| 12 | 945.0 | 37.8 | 0.0 | 907.2 |
| 13 | 2106.7 | 191.5 | 0.1 | 1915.2 |
| 14 | 1537.2 | 256.2 | 0.2 | 1281.0 |
| 15 | 1001.7 | 667.8 | 0.7 | 333.9 |
| 16 | 729.0 | 194.4 | 0.3 | 534.6 |
| 17 | 724.8 | 271.8 | 0.4 | 453.0 |
| 18 | 729.6 | 364.8 | 0.5 | 364.8 |
| 19 | 3429.0 | 457.2 | 0.1 | 2971.8 |
| 20 | 3235.2 | 404.4 | 0.1 | 2830.8 |
| 21 | 257.4 | 514.8 | 2.0 | -257.4 |
| 22 | 724.8 | 271.8 | 0.4 | 453.0 |
| 23 | 2304.0 | 153.6 | 0.1 | 2150.4 |
| 24 | 630.0 | 315.0 | 0.5 | 315.0 |
| 25 | 727.2 | 242.4 | 0.3 | 484.8 |
| 26 | 407.7 | 54.4 | 0.1 | 353.3 |
| 27 | 1787.4 | 79.4 | 0.0 | 1708.0 |
| 28 | 3481.2 | 116.0 | 0.0 | 3365.2 |
| 29 | 2275.2 | 189.6 | 0.1 | 2085.6 |
| 30 | 747.0 | 39.8 | 0.1 | 707.2 |
| 31 | 1408.5 | 37.6 | 0.0 | 1370.9 |
| 32 | 1881.6 | 117.6 | 0.1 | 1764.0 |
| 33 | 3645.0 | 243.0 | 0.1 | 3402.0 |
| 34 | 1065.6 | 355.2 | 0.3 | 710.4 |
| 35 | 2044.8 | 68.2 | 0.0 | 1976.6 |
| 36 | 1359.0 | 36.2 | 0.0 | 1322.8 |
| 37 | 639.0 | 255.6 | 0.4 | 383.4 |
| 38 | 645.0 | 258.0 | 0.4 | 387.0 |
| 39 | 411.4 | 336.6 | 0.8 | 74.8 |
| 40 | 604.8 | 201.6 | 0.3 | 403.2 |
| 41 | 1456.0 | 546.0 | 0.4 | 910.0 |
| 42 | 489.6 | 36.7 | 0.1 | 452.9 |
| 43 | 1528.0 | 45.8 | 0.0 | 1482.2 |
| 44 | 941.2 | 43.4 | 0.0 | 897.8 |
| 45 | 553.8 | 51.1 | 0.1 | 502.7 |
| 46 | 608.4 | 202.8 | 0.3 | 405.6 |
| 47 | 544.0 | 40.8 | 0.1 | 503.2 |
| 48 | 912.0 | 54.7 | 0.1 | 857.3 |
| 49 | 1266.0 | 25.3 | 0.0 | 1240.7 |
| 50 | 598.4 | 224.4 | 0.4 | 374.0 |
| 51 | 1108.8 | 73.9 | 0.1 | 1034.9 |
| 52 | 900.0 | 180.0 | 0.2 | 720.0 |
| 53 | 648.0 | 38.9 | 0.1 | 609.1 |
| 54 | 1518.0 | 30.4 | 0.0 | 1487.6 |
| 55 | 563.2 | 211.2 | 0.4 | 352.0 |
| 56 | 740.0 | 44.4 | 0.1 | 695.6 |
| 57 | 2217.6 | 184.8 | 0.1 | 2032.8 |
| 58 | 1470.0 | 294.0 | 0.2 | 1176.0 |
| 59 | 451.2 | 507.6 | 1.1 | -56.4 |
| 60 | 1202.4 | 100.2 | 0.1 | 1102.2 |
| 61 | 258.8 | 621.0 | 2.4 | -362.2 |
| 62 | 117.8 | 282.6 | 2.4 | -164.9 |

names(iqr.vec2) <- 1:n  
  
names(rel.cum.val.vec.ord)<- id.vec.ord  
  
{plot(iqr.vec2 , rel.cum.val.vec.ord[as.character(1:n)] , xlab="IQR", pch=16, ylab="cum. rel. value share")  
abline(h=c(.8, .95), lty=2, lwd=2, col="darkgrey")  
text(x = c(60,rep(100,3)) , y = c(0.2,.5,.875,.99), labels = c("fast mover", LETTERS[1:3]) )}



## Cost Analysis

#with no constraint  
paste("Seperate Ordering with no capacity constraint, cost is €", round(obj.fun(product\_data$eoq.max)))#172280

## [1] "Seperate Ordering with no capacity constraint, cost is \200 173539"

#with no capacity constraint optimized.  
#paste("Seperate Ordering with no capacity constraint optimized, cost is €",round(obj.fun(copt\_sol1$solution)))# eoq.min  
  
#with constraint  
paste("Seperate Ordering with capacity constraint, cost is €",round(obj.fun(copt\_sol$solution))) #202280

## [1] "Seperate Ordering with capacity constraint, cost is \200 202866"

#JRP  
  
paste("Joint Ordering with no capacity constraint, cost is €",round(c.cost.int)," for m")# € 49943

## [1] "Joint Ordering with no capacity constraint, cost is \200 49943 for m"

paste("Joint Ordering with no capacity constraint, cost is €",round(copt\_sol2$objval)," for T")# € 49943

## [1] "Joint Ordering with no capacity constraint, cost is \200 151952 for T"

paste("Joint Ordering with capacity constraint, cost is €",round(copt\_sol3$objval)," for m")# € 49943

## [1] "Joint Ordering with capacity constraint, cost is \200 268254 for m"

cost.matrix <- data.frame("unconstrained"=c(round(obj.fun(product\_data$eoq.max)),round(c.cost.int)),  
 "constrained"=c(round(obj.fun(copt\_sol$solution)),round(copt\_sol3$objval)))  
rownames(cost.matrix)=c("Separate Ordering","Joint Ordering")  
  
kable(cost.matrix,digits = c(0,1,1,1),caption = "Total cost Matrix", row.names = T, escape = F,format = "pandoc")%>%  
 kable\_styling("striped")

## Warning in kable\_styling(., "striped"): Please specify format in kable.  
## kableExtra can customize either HTML or LaTeX outputs. See https://  
## haozhu233.github.io/kableExtra/ for details.

Total cost Matrix

|  |  |  |
| --- | --- | --- |
|  | unconstrained | constrained |
| Separate Ordering | 173539 | 202866 |
| Joint Ordering | 49943 | 268254 |

in the matrix above, it costs more to When capacity is an issue for both seperate Ordering and Joint Ordering. When non capacited ordering, there is far more cost savings than Separate ordering.

#?seq.int  
#Seperate ordering sequence  
  
so.mat<- matrix(0,nrow = 62, ncol = 40)  
  
for (i in 1:62) {  
   
 so.mat[i,]<- seq.int(round(product\_data$eoq.max[i]),round(copt\_sol$solution[i]) ,length.out = 40)  
  
}  
  
#JRP  
# lower cost  
q.vec

## [1] 107.58017 57.36636 44.15604 95.61060 19.12212 71.74444 38.24424  
## [8] 38.24424 17.95436 22.99033 22.99033 15.98376 71.74444 143.41590  
## [15] 123.05303 95.68358 57.36636 344.19815 57.36636 71.81743 286.97776  
## [22] 143.63485 191.22119 382.44239 143.63485 63.93502 76.48848 76.48848  
## [29] 152.97695 152.97695 191.22119 208.73764 51.08963 71.81743 176.62416  
## [36] 38.24424 159.47264 573.66358 159.47264 239.02649 191.22119 286.83179  
## [43] 179.54356 179.54356 63.86204 57.36636 40.36081 27.07750 40.36081  
## [50] 61.81845 27.07750 61.30756 40.87171 61.30756 46.12664 51.89247  
## [57] 51.89247 79.48087 57.65830 72.25534 79.48087 140.49649

quant.no.const <- data.frame(rbind(round(q.vec )))  
colnames(quant.no.const) <- reo.id   
rownames(quant.no.const) <- c("$q\_i$")  
kable(quant.no.const,"pandoc", row.names = T)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 10 | 7 | 15 | 9 | 41 | 58 | 19 | 20 | 3 | 4 | 57 | 59 | 1 | 4 | 5 1 | 33 | 6 | 1 13 | 29 | 1 | 8 5 | 0 3 | 4 2 | 1 5 | 5 60 | 23 | 52 | 3 | 8 3 | 7 2 | 5 3 | 9 51 | 32 |  | 8 28 | 4 | 6 6 | 2 4 | 0 2 | 4 1 | 6 1 | 1 1 | 7 2 | 2 27 | 35 | 43 | 54 | 12 | 44 | 49 | 30 | 36 | 31 | 56 | 53 | 48 | 45 | 6 | 47 | 42 | 2 | 6 |
|  | 108 | 57 | 44 | 96 | 19 | 72 | 38 | 38 | 18 | 23 | 23 | 16 | 72 | 143 | 123 | 96 | 57 | 344 | 57 | 72 | 287 | 144 | 191 | 382 | 144 | 64 | 76 | 76 | 153 | 153 | 191 | 209 | 51 | 72 | 177 | 38 | 159 | 574 | 159 | 239 | 191 | 287 | 180 | 180 | 64 | 57 | 40 | 27 | 40 | 62 | 27 | 61 | 41 | 61 | 46 | 52 | 52 | 79 | 58 | 72 | 79 | 140 |

library(gtools)  
quant.no.const<-quant.no.const[mixedorder(colnames(quant.no.const))]  
quant.no.const

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21  
## $q\_i$ 96 108 23 23 123 58 44 177 19 57 287 40 57 143 96 191 180 287 38 18 382  
## 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42  
## $q\_i$ 180 76 239 191 140 64 38 72 61 61 72 57 191 57 41 153 153 209 159 72 79  
## 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62  
## $q\_i$ 40 62 79 159 72 52 27 144 51 76 52 27 144 46 16 38 72 64 344 574

# higher cost arranged  
q.start

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24  
## $q\_i$ 17 14 18 18 16 17 18 18 18 15 15 16 18 19 13 20 19 19 20 17 13 19 18 19  
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48  
## $q\_i$ 19 19 19 18 19 17 17 19 18 19 18 16 20 20 14 17 19 17 17 18 18 17 17 17  
## 49 50 51 52 53 54 55 56 57 58 59 60 61 62  
## $q\_i$ 16 19 17 18 16 16 19 16 17 15 19 17 15 15

length(which((quant.no.const > q.start)== TRUE))# 61

## [1] 61

which((quant.no.const < q.start)== TRUE)# only 57

## [1] 57

quant.no.const[57]

## 57  
## $q\_i$ 16

q.start[57]

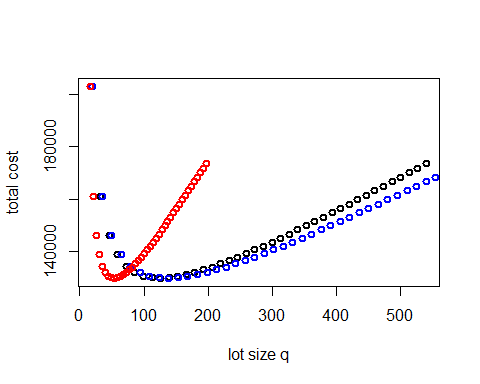
## 57  
## $q\_i$ 17

so.df <- data.frame(rbind(product\_data$eoq.max,copt\_sol$solution))  
  
so.df <-cbind(sum=c(sum(product\_data$eoq.max),sum(copt\_sol$solution)),so.df)  
rownames(so.df)<- c("Non Constrained","Constrained")  
so.df

## sum X1 X2 X3 X4 X5  
## Non Constrained 35235.00 540.00000 599.00000 333.00000 333.00000 631.00000  
## Constrained 1140.33 19.21277 19.98082 18.10408 18.10408 20.07066  
## X6 X7 X8 X9 X10 X11  
## Non Constrained 287.00000 274.00000 960.00000 179.00000 354.00000 1386.00000  
## Constrained 16.50759 19.87768 20.22267 18.02317 20.07752 20.34807  
## X12 X13 X14 X15 X16 X17  
## Non Constrained 198.00000 293.0000 636.00000 787.00000 825.00000 802.00000  
## Constrained 16.36283 18.5035 20.28845 15.09718 20.37517 20.21815  
## X18 X19 X20 X21 X22 X23  
## Non Constrained 1130.00000 381.00000 190.00000 1553.00000 802.00000 328.00000  
## Constrained 19.93476 20.25461 18.49918 15.51115 20.21815 19.03311  
## X24 X25 X26 X27 X28 X29  
## Non Constrained 993.00000 924.00000 638.00000 305.00000 189.00000 370.00000  
## Constrained 20.33883 20.33375 18.30819 18.16358 17.33315 19.59133  
## X30 X31 X32 X33 X34 X35  
## Non Constrained 258.00000 266.00000 352.00000 261.00000 763.00000 246.00000  
## Constrained 17.14746 17.15822 18.92858 18.91119 20.31552 17.49555  
## X36 X37 X38 X39 X40 X41  
## Non Constrained 180.00000 719.00000 715.00000 1046.0000 675.00000 652.00000  
## Constrained 16.37103 20.20202 20.20152 20.1963 20.09364 19.85988  
## X42 X43 X44 X45 X46 X47  
## Non Constrained 357.00000 180.00000 285.00000 371.00000 673.00000 339.00000  
## Constrained 17.04693 16.50265 17.29302 17.36417 20.09317 17.03609  
## X48 X49 X50 X51 X52 X53  
## Non Constrained 234.00000 161.00000 714.00000 313.00000 425.00000 275.00000  
## Constrained 16.63235 15.59037 19.53891 17.19712 18.05721 16.17151  
## X54 X55 X56 X57 X58 X59  
## Non Constrained 147.00000 736.00000 257.00000 247.00000 470.00000 1162.00000  
## Constrained 15.52704 19.54321 16.15005 17.44414 19.59295 20.38314  
## X60 X61 X62  
## Non Constrained 336.00000 1692.00000 2508.00000  
## Constrained 17.61382 14.94486 14.83263

As seen from the table, with no constraint you tend to order far more than constrained.

#SO  
so.cost.vec <- c(numeric(0),1:40)  
for (i in 1:40) {  
 so.cost.vec[i] <- obj.fun(so.mat[,i])  
}  
#so.cost.vec  
so.data.mat <- as.data.frame(so.mat)  
  
  
{plot(x=so.mat[1,], round(so.cost.vec), xlab ="lot size q", ylab = "total cost", type = "p", lwd = 2)  
points(x=so.mat[2,], y=round(so.cost.vec), lwd=2, col="blue")  
points(x=so.mat[12,], y=round(so.cost.vec), lwd=2, col="red")}



#JO

Without considering capacity constraint, Using some examples for illustration,for all items using the EOQ model. for Items 1 and 2, EOQ is 97 and 108 respectively without paying extra 1500.

When ordering cost of 1500 and capacity for items 1 and 2 has order quantities of 19 and 20 respectively, thus costing far more as seen in the graph above. This is largely due to lack of capacity constraint because so little is ordered, that 1500 is paided all the time order takes place for each items.

jrp.mat<- matrix(0,nrow = 62, ncol = 40)  
  
for (i in 1:62) {  
   
 jrp.mat[i,]<-seq.int(unlist(quant.no.const[i]),unlist(q.start[i]),length.out = 40)  
  
}  
  
jrp.m.mat <- matrix(0,nrow = 62, ncol = 40)  
  
m.mat<- data.frame(rbind(m.vec.int,df2[1,]))  
m.mat <-cbind(sum=c(sum(m.vec.int),sum(df2[1,])),m.mat)  
rownames(m.mat)<- c("No Constrained","Constrained")  
m.mat

## sum X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16 X17  
## No Constrained 285 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3  
## Constrained 544 4 1 6 6 3 22 3 3 7 2 2 21 7 3 1 4 4  
## X18 X19 X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33  
## No Constrained 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4  
## Constrained 2 4 7 1 4 7 3 3 11 11 14 6 17 17 8 7  
## X34 X35 X36 X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49  
## No Constrained 4 4 4 5 5 5 5 5 5 5 5 5 6 7 7 7  
## Constrained 3 14 24 4 4 2 4 2 18 22 15 15 4 18 22 31  
## X50 X51 X52 X53 X54 X55 X56 X57 X58 X59 X60 X61 X62  
## No Constrained 7 7 8 8 8 8 9 9 9 10 10 11 11  
## Constrained 4 10 7 21 31 4 21 8 3 2 8 1 1

As seen above, with no capacity, you generally have smaller order frequencies across the 62 items which shows but larger order quantity with lower cost generally.

Since we have limited capacity, we will order frequently and therefore will not take advantage of cost savings we could have gotten if we have larger capacity.

#obj.fun()  
#so.mat[2,]  
#so.cost.vec  
#so.mat  
#t<-1  
#for(i in 1:62){  
   
 #df <- paste("Item", i, (sum(Periodic.inventory(i)[3,1:7])/5))  
 #print(df)  
   
#}  
  
#floor(product\_data$cycle\_time\_in\_days[i])  
#kable(df,"html", row.names = T, escape = F) %>%  
 #kable\_styling("striped")