An open-source tree bucking optimizer based on dynamic programming Supplementary material

Caroline Bennemann, Jean-Martin Lussier & Eric R. Labelle

Version March 21st 2025

Table of Contents

[INTRODUCTION 2](#_Toc193451646)

[CONVENTIONS 3](#_Toc193451647)

[Variable names 3](#_Toc193451648)

[Working folder 3](#_Toc193451649)

[Input files 3](#_Toc193451650)

[Price list 3](#_Toc193451651)

[Stem profile 4](#_Toc193451652)

[SETUP 4](#_Toc193451653)

[1.1. Package installation 4](#_Toc193451654)

[1.2 Definition of function for log Volume 4](#_Toc193451655)

[1.3 Selection of the working directory 5](#_Toc193451656)

[1.4 Import of input files 5](#_Toc193451657)

[1.5 Data preprocessing 6](#_Toc193451658)

[Modification data type 6](#_Toc193451659)

[Price list 6](#_Toc193451660)

[Calibration of price model 7](#_Toc193451661)

[Tree list 9](#_Toc193451662)

[Identification of merchantable length 9](#_Toc193451663)

[Suppression of short trees 9](#_Toc193451664)

[Summary per Tree 10](#_Toc193451665)

[Creation of important variables 10](#_Toc193451666)

[OPTIMIZATION 10](#_Toc193451667)

[2.1 Creation of the log list for the longest tree 10](#_Toc193451668)

[2.2 Creation of the log list for each tree 12](#_Toc193451669)

[2.3 Computation of logs attributes 13](#_Toc193451670)

[2.4 Calculation of log volumes 13](#_Toc193451671)

[2.5 Classification of logs and imputation of price per cubic meter 13](#_Toc193451672)

[2.6 Valuation of logs 14](#_Toc193451673)

[2.7 Selection of the best bucking solution 16](#_Toc193451674)

[Best Number of logs per tree 16](#_Toc193451675)

[Best log sequence per tree 16](#_Toc193451676)

[2.8 Export of output files 17](#_Toc193451677)

[2.9 THE END ! 17](#_Toc193451678)

[REFERENCES 18](#_Toc193451679)

# INTRODUCTION

This bucking optimization script is based on dynamic programming, as proposed by Dykstra (1984). The algorithm is searching for the sequence of logs that maximizes the value of a tree, given its diameter profile, rules to grade the logs based on their attributes, and a price list by log category. The decision variable is the position of the cuts along the stem from its base.

The script is based on the Example 10.2 in the section 10.4 of Dykstra (1984). This version is meant to be published with a scientific article.

The algorithm is tested on a single tree of 200 cm diameter at stump with a merchantable length of 15.0 m and a diameter taper reduction of 12 cm per liner meter. The length of logs that can be generated are 3.0, 4.5 and 6.0 m long with different minimal diameters and different associated prices (provided in the input file).

The price list provides the value in $/m³ of logs for each species, log length and diameter range (SEDmin and SEDmax). The computation of the value of each log, based on its attributes, is facilitated through the use of a tree classification model, calibrated with the price list data. This approach will hopefully simplify the trial-and-error process and editing of the price list.

The script prompts the end-user to select the working directory and the input files. See the section “CONVENTIONS” for more details on data preparation.

This script was developed with RStudio 2024.09.0 and R version 4.4.1.

As a bonus, the script will celebrate your success at the end!

## CONVENTIONS

The example input files provide the structure of the input data.

### Variable names

**ALL LENGTHS AND DIAMETERS ARE IN CENTIMETERS AND INTEGER VALUES**

* **Tree.No** Tree Number (numeric)
* **ID.Tree** Tree ID (factor)
* **SP** Log Species (factor)
* **LG** In the stem profile, distance from first cross-cut at the stump (numeric, cm)
* **DG** In the stem profile, diameter measured on the stem at LG (numeric, mm)
* **L** Log Length (numeric, cm)
* **SED** Log Small-end diameter (numeric, mm)
* **LED** Log Large-end diameter (numeric, mm)
* **G** Log Grade (integer)
* **V** Log Volume (numeric, m³)
* **VAM** Log Value per m³ (numeric, $/m³ or naira/m³ in Dykstra’s example)
* **VAL** Log Value per log (numeric, $/log or naira/log in Dykstra’s example)

### Working folder

The input and output files for an optimization project are placed in a working directory, which name is at the discretion of the end-user (make it short, though!). However, the folder **must have** two sub-folders named IN and OUT.

The input folder (IN) **must** include two types of data, in this example stored within one same Excel file with two sheets. The sheets are named “prices” and referred to as PRICE.LIST in the underlaying script and STEM.PROFILES, stored in the sheet “tree”, including the stem taper in 50 cm increments).

## Input files

### Price list

* The price list is provided by the end-user in a CSV file (see example file).
* The Price List as a fixed format with the following variables: Species (SP), ProductID (unique ID of products), log length (L), smallest diameter accepted (SEDmin), largest diameter accepted (SEDmax) and value per m³ (VAM). **All values are in millimeters**.
* For each length and species, the user must specify prices for all diameter ranges from 0 to 200 cm. For example, based on the product specifications in the example by Dykstra, the log with a length of 300 cm ranges between 200 and 499 mm. In this case, as all other products are foreseen up to a diameter of 2000 mm, a log of 300 cm length and diameters ranging from 50 cm to 200 cm with a VAM of 0 is added to the database.
* Prices do not need to be real market values, but rather relative ones. (Be careful: a long log of low value per m³ with a large volume may be preferred to a shorter one with higher value per m³, but that has less volume. It is the value per log that counts for the optimization process!
* It is necessary to ensure all tree species that occur in PROFILE have products and prices associated. Otherwise, a bug will occur.

### Stem profile

* The stem profile data table provides a series of pairs of measures of diameter (DG) and position from the ground (LG) for each tree to be processed. **All values of LG are in centimeters, all values of DG are in millimeters, following the StanForD 2010 standard (Skogforsk 2021)**.
* For each tree, the first LG/DG data is for the first cut at the stump.
* TreeID (ID.Tree) and species (SP) must also be included in the file.
* Other variables such as tree DBH and total height can be included but will not be considered in the process.

# SETUP

## 1.1. Package installation

#install.packages("pacman")  
  
library(pacman) # Load the packages, and install them if necessary  
  
pacman::p\_load(plyr, dplyr, ggplot2,flextable, beepr, rpart, caret)

## 1.2 Definition of function for log Volume

Function to calculate log volume (in m³) using formula 10.3 from Dykstra (1984).

**Input**

* **Log.Length** Length of the log in centimeters
* **SED** small-end diameter of the log in millimeters

**Output**

* **V** log volume in m³

# Log volume function from the Dykstra's example  
  
fun\_volume\_Dykstra <- function(Log.Length, SED) {  
 SED = SED/10 # Convert from mm to cm  
 Log.Length = Log.Length / 100 # Convert from cm to m  
 V = 0.9 \* pi \* ((SED / 200)^2) \* Log.Length # volume in cubic meters  
 V <- round(V, digits = 2)  
 return(V)  
}

# Log volume function from Smalian's formula  
  
fun\_volume\_Smalian <- function(Log.Length, SED, LED) {  
  
 # Convert SED, LED and Log.Length in meters  
 SED\_m <- SED \* 0.001  
 LED\_m <- LED \* 0.001  
 L\_m <- Log.Length \* 0.01  
   
 # Calculate the volume in cubic meters  
 V <- (L\_m \* pi / (8 \* 10^10)) \* (SED^2 + LED^2)  
 V <- round(V, digits = 2)  
 return(V)  
}

## 1.3 Selection of the working directory

The following chunks are modified by the end-user.

The working directory is a folder dedicated to a single optimization process, with 2 folders: IN and OUT. These must be created before working on this script.

**THE END-USER MUST MODIFY THE FOLLOWING CHUNK**

WorkDir <- "C:/BuckR/Validation-algo"

## 1.4 Import of input files

PRICE.LIST <- read.csv("C:/Users/BuckR/Validation-algo/IN/PRICE.LIST.csv")  
  
PROFILE <- read.csv("C:/Users/BuckR/Validation-algo/IN/PROFILE.csv")  
  
flextable(PRICE.LIST)

| SP | ProductID | Lmin | Lmax | SEDmin | SEDmax | VAM |
| --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 300 | 449 | 200 | 499 | 90 |
| A | 2 | 300 | 449 | 500 | 2,000 | 0 |
| A | 3 | 450 | 599 | 200 | 499 | 100 |
| A | 4 | 450 | 599 | 500 | 2,000 | 125 |
| A | 5 | 600 | 749 | 200 | 499 | 115 |
| A | 6 | 600 | 749 | 500 | 799 | 150 |
| A | 7 | 600 | 749 | 800 | 2,000 | 200 |

#flextable(PROFILE)

## 1.5 Data preprocessing

### Modification data type

PROFILE$SP <- as.factor(PROFILE$SP)  
PROFILE$ID.Tree <- as.factor(PROFILE$ID.Tree)  
  
PRICE.LIST$SP <- as.factor(PRICE.LIST$SP)

### Price list

# Delete empty cases  
PRICE.LIST <- PRICE.LIST[complete.cases(PRICE.LIST), ]   
  
flextable(PRICE.LIST)

| SP | ProductID | Lmin | Lmax | SEDmin | SEDmax | VAM |
| --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 300 | 449 | 200 | 499 | 90 |
| A | 2 | 300 | 449 | 500 | 2,000 | 0 |
| A | 3 | 450 | 599 | 200 | 499 | 100 |
| A | 4 | 450 | 599 | 500 | 2,000 | 125 |
| A | 5 | 600 | 749 | 200 | 499 | 115 |
| A | 6 | 600 | 749 | 500 | 799 | 150 |
| A | 7 | 600 | 749 | 800 | 2,000 | 200 |

# Compile what are the smallest log Length and SED for the price list.

MIN.merchantable.L = min(PRICE.LIST$Lmin)  
MIN.merchantable.SED = min(PRICE.LIST$SEDmin)

### Calibration of price model

This function transforms the price list into a classification tree model to facilitate the application and tuning of the price list. The model predicts the ProductID code. The price can then be predicted with the use of a lookup table.

# STEP ONE : Conversion of the price list into a "long" format  
# with increments of SED.  
  
Nb.rows <- nrow(PRICE.LIST)  
  
# Creation of a empty dataframe  
PRICE.LIST.LONG = data.frame(L = numeric(0),  
 SEDmin = numeric(0),  
 SEDmax = numeric(0),  
 ProductID = numeric(0))   
  
for (i in 1:Nb.rows) {  
   
 # Extract the needed variable values  
 val.L <- as.numeric(PRICE.LIST[i,3]) # length  
 val.SEDmin <- as.numeric(PRICE.LIST[i,5]) # SEDmin  
 val.SEDmax <- as.numeric(PRICE.LIST[i,6]) # SEDmax  
 val.ProductID <- as.numeric(PRICE.LIST[i,2]) # ProductID   
   
 # Generate the list of all combinations of SED between  
 # SEDmin & SEDmax with intervals of 1cm  
  
 tmp <- expand.grid(L = val.L,  
 SED = seq(from = val.SEDmin, to = val.SEDmax, by = 1),  
 ProductID = val.ProductID)  
   
 PRICE.LIST.LONG <- rbind(PRICE.LIST.LONG, tmp)  
}  
  
PRICE.LIST.LONG$ProductID <- as.factor(PRICE.LIST.LONG$ProductID)  
  
# STEP TWO Training of the classification model  
price.model <- rpart(ProductID ~ L + SED,  
 data = PRICE.LIST.LONG, method = "class")  
  
# STEP THREE Performance measure  
PRICE.LIST.LONG$predict <- predict(price.model, PRICE.LIST.LONG,  
 type = "class")  
  
  
Confusion <- confusionMatrix(PRICE.LIST.LONG$predict, PRICE.LIST.LONG$ProductID)  
  
# To check additional parameters of the confusion matrix  
# options(max.print = 999999)  
# confusion  
  
print("CLASSICATION MODEL ACCURACY")

## [1] "CLASSICATION MODEL ACCURACY"

Confusion$overall # accuracy needs to be 1.00

## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 1.0000000 1.0000000 0.9993175 1.0000000 0.2778086   
## AccuracyPValue McnemarPValue   
## 0.0000000 NaN

print("CONFUSION MATRIX")

## [1] "CONFUSION MATRIX"

Confusion$table

## Reference  
## Prediction 1 2 3 4 5 6 7  
## 1 300 0 0 0 0 0 0  
## 2 0 1501 0 0 0 0 0  
## 3 0 0 300 0 0 0 0  
## 4 0 0 0 1501 0 0 0  
## 5 0 0 0 0 300 0 0  
## 6 0 0 0 0 0 300 0  
## 7 0 0 0 0 0 0 1201

# Confusion table to check where possible problems may be  
  
rm(tmp, Confusion)

### Tree list

Cleaning the dataframe: sort rows and put variables in the right order (IMPORTANT) and delete those who are not necessary for the rest of the script (DB and Tree.No which will be newly created) DBH and max.LG deleted from PROFILE as not present in example.

PROFILE <- PROFILE %>%  
 arrange(Tree.No, LG) %>% # Sort the rows  
 select(Tree.No, ID.Tree, SP, LG, DG)

Creating the list of trees

Summary.Trees.Initial <- PROFILE %>%  
 ungroup() %>%  
 group\_by(Tree.No, ID.Tree, SP) %>%  
 summarise(Stump.LG = min(LG),  
 Stump.DG = max(DG),  
 Top.LG = max(LG),   
 Top.DG = min(DG),  
 Nb.Meas = n(),  
 max.LG = max(LG))  
  
Summary.Trees.Initial$SP <- as.factor (Summary.Trees.Initial$SP)

### Identification of merchantable length

Calculating the merchantable length for each tree

tmp <- PROFILE %>% ungroup() %>% # Creation of a temporary table  
 group\_by(Tree.No) %>%   
 filter(DG >= MIN.merchantable.SED) %>% # Subset observations with  
 # a diameter greater or equal to the merchantable diameter  
 summarise(merch.LC = max(LG))  
  
Summary.Trees.Initial <- left\_join(Summary.Trees.Initial, tmp,  
 by = "Tree.No")  
  
rm(tmp)

### Suppression of short trees

All trees with merchantable height shorter than the shortest possible product are extracted.

# List of trees removed  
LIST.SHORT.TREES <- Summary.Trees.Initial %>% ungroup() %>%  
 filter(is.na(merch.LC) | merch.LC < MIN.merchantable.L) %>%  
 select(ID.Tree, merch.LC) %>%  
 unique()  
  
# List of remaining trees  
Summary.Trees <- Summary.Trees.Initial %>% ungroup() %>%  
 filter(merch.LC >= MIN.merchantable.L)  
  
print("LIST OF REMOVED TREES BECAUSE OF TOO SHORT STEMS")

## [1] "LIST OF REMOVED TREES BECAUSE OF TOO SHORT STEMS"

print(LIST.SHORT.TREES)

## # A tibble: 0 × 2  
## # ℹ 2 variables: ID.Tree <fct>, merch.LC <int>

rm(Summary.Trees.Initial)

### Summary per Tree

flextable(Summary.Trees)

| Tree.No | ID.Tree | SP | Stump.LG | Stump.DG | Top.LG | Top.DG | Nb.Meas | max.LG | merch.LC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | A | 0 | 2,000 | 1,500 | 200 | 1,501 | 1,500 | 1,500 |

### Creation of important variables

# List unique tree numbers  
Tree.LIST <- unique(Summary.Trees$Tree.No)  
# Number of trees in the list  
NB.Trees <- length(Tree.LIST)  
# Merchantable length of the largest tree  
max.merch.LC <- max(Summary.Trees$merch.LC)  
# List of unique nominal lengths in the price list  
Length.LIST <- unique(PRICE.LIST$Lmin)

# OPTIMIZATION

### 2.1 Creation of the log list for the longest tree

This is done for each stage (log number from the stump), using Summary.Trees as an input. The table “logperms” contains all log permutations of the longest tree for all stages, with the length of the log at stage “LogNo”, and the cumulative log length at the previous and current stage (TLprev, TLact).

gc() # Free memory before computation of possibilities

# Generate the maximal number of logs possible in the highest tree of the dataset  
Max.Nb.Logs <- trunc(max.merch.LC / min(Length.LIST))  
  
# Creation dataframe containing all variables  
logperms <- data.frame(LogNo = 1,  
 TLprev = 0,  
 L = Length.LIST,  
 TLact = Length.LIST)  
  
# LogNo: log ID (here, the first possible log)  
# TLprev: total length of previous logs (is zero for LogNo = 1)  
# L : log length  
# TLact: current log length (current height)  
  
# Reordering variables  
logperms <- logperms %>% select (LogNo, TLprev, L, TLact)   
  
  
# Loop for all logs except of the first one  
for (logNo in 2:Max.Nb.Logs) {   
   
 # Extraction of current log length of previous LogNo  
 List.TLact <- logperms %>% filter(LogNo == (logNo - 1)) %>% select(TLact)   
  
   
 # All permutations of logs at current height   
 df <- expand.grid(Length.LIST, List.TLact$TLact)   
  
 df <- df %>% rename(L = Var1, TLprev = Var2) %>%  
 mutate(LogNo = logNo) %>%  
 mutate(TLact = L + TLprev) %>%  
 select (LogNo, TLprev, L, TLact) %>%   
 filter(TLact <= max.merch.LC)  
 # Filter: height reached does not exceed maximal height of tree  
   
 # Join of dataframes  
 logperms <- rbind (logperms, df)   
   
 rm(df)  
   
 # Suppression of duplicates, if present  
 logperms <- distinct(logperms)   
  
}  
  
rm(logNo, List.TLact)

### 2.2 Creation of the log list for each tree

COMPLETE.LOG.LIST contains all possible log combinations for every study tree, based on logperms.

A counter is set to following the calculation loop.

COMPLETE.LOG.LIST <- data.frame(Tree.No = numeric(),  
 SP = character (0),   
 LogNo = numeric(0),  
 TLprev = numeric(0),  
 L = numeric (0),   
 TLact = numeric (0))  
   
# Loop to fill COMPLETE.LOG.LIST  
for (Tree.Index in 1:NB.Trees) {  
   
 print(paste0("Tree.no:",Tree.Index))  
 print("--------")  
   
 # Filling values by values stored in Summary.Trees  
 Tree.No1 <- as.numeric(Summary.Trees[Tree.Index, 1]) # Tree.No  
 SP1 <- as.character(Summary.Trees[Tree.Index, 3]) # SP  
 max.LG1 <- as.numeric(Summary.Trees[Tree.Index, 9]) # max.LG  
 SLG1 <- as.numeric(Summary.Trees[Tree.Index, 4]) # Stump.LG  
   
 # Addition of variables to dataframe tmp  
 tmp <- data.frame(Tree.No = Tree.No1,  
 SP = SP1,  
 max.LG = max.LG1,  
 Stump.LG = SLG1)  
   
 temporary <- cbind(tmp, logperms) # join of dataframes  
   
 # Filter: reached log length is not exceeding tree height  
 temporary <- temporary %>% filter(TLact <= (max.LG - Stump.LG))  
   
 # Suppression of NA if necessary  
 temporary <- na.omit(temporary)  
   
 COMPLETE.LOG.LIST <- rbind(COMPLETE.LOG.LIST, temporary)  
   
 rm(tmp, temporary, Tree.No1, SP1, max.LG1, SLG1)  
 gc()  
  
}

## [1] "Tree.no:1"  
## [1] "--------"

### 2.3 Computation of logs attributes

Compute SED, LED, V, G and VAL for all logs for each stage.

Nb.rows <- nrow(COMPLETE.LOG.LIST)  
  
tmp1 = data.frame(Tree.No = numeric(),  
 SP = character (0),   
 LogNo = numeric(0),  
 TLprev = numeric(0),  
 L = numeric (0),   
 TLact = numeric (0))  
  
tmp <- PROFILE %>% select (Tree.No, LG, DG) %>% rename(SED = DG)  
COMPLETE.LOG.LIST <- left\_join (COMPLETE.LOG.LIST, tmp,  
 by = c("Tree.No" = "Tree.No",  
 "TLact" = "LG"))  
rm(tmp)  
  
COMPLETE.LOG.LIST$Ttemp <- COMPLETE.LOG.LIST$TLact - COMPLETE.LOG.LIST$L  
tmp <- PROFILE %>% select (Tree.No, LG, DG) %>% rename(LED = DG)  
  
COMPLETE.LOG.LIST <- left\_join (COMPLETE.LOG.LIST, tmp,  
 by = c("Tree.No" = "Tree.No",  
 "Ttemp" = "LG"))  
COMPLETE.LOG.LIST$Ttemp <- NULL  
rm(tmp)

### 2.4 Calculation of log volumes

With previously defined function.

COMPLETE.LOG.LIST$V <- fun\_volume\_Dykstra(COMPLETE.LOG.LIST$L, COMPLETE.LOG.LIST$SED)

#### 2.5 Classification of logs and imputation of price per cubic meter

Prediction of price based on price.model by the prediction of ProductID.

COMPLETE.LOG.LIST$ProductID <- predict(price.model, COMPLETE.LOG.LIST,  
 type = "class")  
  
# Lookup is a dataframe to see prices of each product  
Lookup <- PRICE.LIST %>% select(ProductID, VAM)   
  
Lookup$ProductID <- as.factor(Lookup$ProductID)  
  
# Join of COMPLETE.LOG.LIST and Lookup   
COMPLETE.LOG.LIST <- left\_join(COMPLETE.LOG.LIST, Lookup, by = "ProductID")  
  
# Value of logs depending on their volume   
COMPLETE.LOG.LIST$VAL <- COMPLETE.LOG.LIST$VAM \* COMPLETE.LOG.LIST$V

### 2.6 Valuation of logs

The value of logs is stored in the variable “f”, along with the best value for the total length of logs from the previous stage (Best.Prev.f).

The outcomes are the updated COMPLETE.LOG.LIST with “f” and BEST.LOG.LIST, which gives the best value per TOT.L, for each log number and each tree.

# Initialization  
COMPLETE.LOG.LIST2 = data.frame(Tree.No = as.numeric(),  
 SP = as.character(),  
 LogNo = as.integer(),  
 TLact = as.numeric(),  
 L = as.numeric(),  
 SED = as.numeric(),  
 LED = as.numeric(),  
 V = as.numeric(),  
 G = as.numeric(),  
 VAL = as.numeric(),  
 TLprev = as.numeric(),  
 Best.Prev.f = as.numeric(),  
 f = as.numeric())   
BEST.LOG.LIST <- COMPLETE.LOG.LIST2  
  
# Computations  
  
for (Tree.Index in 1:NB.Trees) {  
   
 print("--------")  
 print(paste0("TreeNo:",Tree.Index))  
   
 # Extract the necessary values for the tree  
 Tree.No <- as.numeric(Summary.Trees[Tree.Index, 1]) # Tree.No  
 SPECIES <- as.character(Summary.Trees[Tree.Index, 3]) # SP  
 Max.Length <- as.numeric(Summary.Trees[Tree.Index, 9]) # max.LG  
  
 ## First log  
   
 print(paste0("LogNo:1"))  
   
 # Subset only the needed tree and Log data & Put new variables  
 TMP <- COMPLETE.LOG.LIST %>%  
 filter(Tree.No == Tree.Index & LogNo == 1) %>%  
 mutate(TLprev = 0, Best.Prev.f = 0, f = VAL)  
   
 # Find the best sequence for the current log number  
 BEST.TMP <- TMP %>% group\_by(TLact) %>% slice(which.max(f))  
   
 # Glue everything in the destination dataframes  
 COMPLETE.LOG.LIST2 <- rbind(COMPLETE.LOG.LIST2, TMP)  
 BEST.LOG.LIST <- rbind(BEST.LOG.LIST, BEST.TMP)  
   
 rm(TMP, BEST.TMP)  
  
 ## Loop for the other logs  
   
 if (Max.Nb.Logs > 1) {  
   
 for (Log.Index in 2 : Max.Nb.Logs) {  
   
 print(paste0("LogNo:", Log.Index))  
   
 # Subset only the needed tree and Log data & Rename for later join  
 TMP <- COMPLETE.LOG.LIST %>%  
 filter(Tree.No == Tree.Index & LogNo == Log.Index) %>%   
 mutate(TLprev = TLact - L)   
   
 # Subset the best sequence from the previous log number  
 TMP2 <- BEST.LOG.LIST %>%   
 filter(Tree.No == Tree.Index & LogNo == Log.Index - 1) %>%  
 select(TLact, f) %>%   
 rename("Best.Prev.f" = "f",  
 "TLprev" = "TLact")   
   
 # Join based on total length of previous products  
 TMP <- left\_join(TMP, TMP2, by = ("TLprev"))  
   
 # Compute f  
 TMP <- TMP %>% mutate(f = VAL + Best.Prev.f)  
   
 # Find the best sequence for the current log number  
 BEST.TMP <- TMP %>% group\_by(TLact) %>% slice(which.max(f))  
   
 # Glue everything in the destination dataframes  
 COMPLETE.LOG.LIST2 <- rbind(COMPLETE.LOG.LIST2, TMP)  
 BEST.LOG.LIST <- rbind(BEST.LOG.LIST, BEST.TMP)  
   
 rm(TMP, TMP2, BEST.TMP)  
 }  
 }  
}

## [1] "--------"  
## [1] "TreeNo:1"  
## [1] "LogNo:1"  
## [1] "LogNo:2"  
## [1] "LogNo:3"  
## [1] "LogNo:4"  
## [1] "LogNo:5"

COMPLETE.LOG.LIST <- COMPLETE.LOG.LIST2  
rm(COMPLETE.LOG.LIST2)

### 2.7 Selection of the best bucking solution

Based on the above, this selects the sequence of logs that maximizes value.

#### Best Number of logs per tree

# Find the total length with the maximum value  
Best.Number.of.Logs <- BEST.LOG.LIST %>%   
 group\_by(Tree.No) %>%   
 slice(which.max(f)) %>%  
 select(Tree.No, LogNo) %>%  
 rename("Best.Nb.of.Log" = "LogNo")

#### Best log sequence per tree

# Find the best sequence  
BEST.SEQUENCE <- left\_join(BEST.LOG.LIST, Best.Number.of.Logs,  
 by = ("Tree.No"))  
  
BEST.SEQUENCE <- BEST.SEQUENCE %>%  
 filter (LogNo <= Best.Nb.of.Log) %>%  
 group\_by(Tree.No, LogNo) %>%   
 slice(which.max(f))  
  
id <- Summary.Trees %>% select (Tree.No, ID.Tree)  
BEST.SEQUENCE <- left\_join(BEST.SEQUENCE, id)  
  
BEAUTIFUL.BEST.SEQUENCE <- BEST.SEQUENCE %>%  
 select(ID.Tree, SP, LogNo, L, SED, V, VAL, ProductID) %>%  
 mutate(SED = round(SED, digits = 0),  
 V = round(V, digits = 2),  
 VAL = round(VAL, digits = 2))  
   
flextable (BEAUTIFUL.BEST.SEQUENCE)

| Tree.No | ID.Tree | SP | LogNo | L | SED | V | VAL | ProductID |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 600 | 1,280 | 6.95 | 1,390.0 | 7 |
| 1 | 1 | 1 | 2 | 450 | 740 | 1.74 | 217.5 | 4 |
| 1 | 1 | 1 | 3 | 300 | 380 | 0.31 | 27.9 | 1 |

val.tot <- sum(BEAUTIFUL.BEST.SEQUENCE$VAL, na.rm = TRUE)

## 2.8 Export of output files

* **COMPLETE.LOG.LIST** provides for each tree all the possible single log lengths for each log number (the first, second, third, …) by total length of logs. This is computed iteratively from the first log to the n-th log.
* **BEST.LOG.LIST** is the log length that maximize f (the cumulative best value) for each log number, from the COMPLETE.LOG.LIST.
* **BEST.SEQUENCE** is the log sequence that maximize f considering all log numbers, from BEST.LOG.LIST
* **Summary.Trees** is the above without the trees with a stem shorter than the shortest product.
* **LIST.SHORT.TREES** is the list of stems excluded because they are shorter than the shortest possible log.

write.csv(Summary.Trees,paste0(WorkDir,"/OUT/SummaryTrees.csv"),  
 row.names = F)  
write.csv(logperms,paste0(WorkDir,"/OUT/logperms.csv"),  
 row.names = F)  
write.csv(COMPLETE.LOG.LIST,paste0(WorkDir,"/OUT/COMPLETE\_LOG\_LIST.csv"),  
 row.names = F)  
write.csv(BEST.LOG.LIST,paste0(WorkDir,"/OUT/BEST\_LOG\_LIST.csv"),  
 row.names = F)  
write.csv(BEST.SEQUENCE,paste0(WorkDir,"/OUT/BEST\_SEQUENCE.csv"),  
 row.names = F)  
write.csv(LIST.SHORT.TREES,paste0(WorkDir,"/OUT/Too\_Short\_Trees.csv"),  
 row.names = F)

## 2.9 THE END !

beep(sound = 3, expr = NULL)

**We recommend to the end-user to save this Markdown report in the same folder as the input and output files and to keep a separate folder for each analysis.**

# REFERENCES

* Dykstra, D.P. 1984. Chapter 10 - Dynamic Programming. In Mathematical Programming for Natural Resources Management. McGraw-Hill, New York. pp. 287-310.
* Skogforsk. 2021. StanForD 2010 – modern communication with forest machines. Skogforsk. Available from <https://www.skogforsk.se/cd_20210625001813/contentassets/1a68cdce4af1462ead048b7a5ef1cc06/stanford2010-info-english-2021.zip>. Accessed online on 2025-03-18