

**A THREE-STAGE CONTROL MECHANISM FOR  
THE LUMBER PRODUCTION PROCESS OF A  
SAWMILL BASED ON A POWERS-OF-TWO  
MODELLING APPROACH**

by

Pegah Sohrabi

Submitted in partial fulfillment of the requirements  
for the degree of Master of Applied Science

at

Dalhousie University  
Halifax, Nova Scotia  
December 2012

©Copyright by Pegah Sohrabi, 2012

DALHOUSIE UNIVERSITY  
DEPARTMENT OF INDUSTRIAL ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “A THREE-STAGE CONTROL MECHANISM FOR THE LUMBER PRODUCTION PROCESS OF A SAWMILL BASED ON A POWERS-OF-TWO MODELLING APPROACH” by Pegah Sohrabi in partial fulfilment of the requirements for the degree of Master of Applied Science.

Dated: 6, December, 2012

Co-Supervisors: \_\_\_\_\_

\_\_\_\_\_

Readers: \_\_\_\_\_

\_\_\_\_\_

DALHOUSIE UNIVERSITY

DATE: 6, December, 2012

AUTHOR: Pegah Sohrabi

TITLE: A THREE-STAGE CONTROL MECHANISM FOR THE  
LUMBER PRODUCTION PROCESS OF A SAWMILL  
BASED ON A POWERS-OF-TWO MODELLING AP-  
PROACH

DEPARTMENT OR SCHOOL: Department of Industrial Engineering

DEGREE: MASc CONVOCATION: May YEAR: 2013

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than the brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

---

Signature of Author

تقدیم به همسر و خانواده عزیزم که همواره پشتیبان و مشوقم بوده اند...

*To my beloved husband and family who have always  
been supportive and encouraging to me. . .*

# Table of Contents

|  |           |
|--|-----------|
| List of Tables                                   | vii       |
| List of Figures                                  | viii      |
| Abstract   | x         |
| List of Abbreviations Used                       | xi        |
| Acknowledgments                                  | xii       |
| <b>Chapter 1 Introduction</b>                    | <b>1</b>  |
| 1.1 Sawmill Operations . . . . .                 | 1         |
| 1.2 Problem Description . . . . .                | 3         |
| <b>Chapter 2 Literature Review</b>               | <b>5</b>  |
| 2.1 Log Breakdown . . . . .                      | 5         |
| 2.2 Lumber Production Planning . . . . .         | 7         |
| 2.3 ELSP and Powers-of-Two . . . . .             | 9         |
| <b>Chapter 3 Creating Campaigns</b>              | <b>15</b> |
| 3.1 Methodology . . . . .                        | 16        |
| 3.1.1 Generating Cutting Patterns . . . . .      | 20        |
| 3.1.2 Generating Logs . . . . .                  | 29        |
| 3.1.3 Creating Price Lists . . . . .             | 30        |
| 3.1.4 Creating Lumber Output Fractions . . . . . | 31        |
| 3.2 Example . . . . .                            | 37        |
| 3.2.1 Data and Procedures . . . . .              | 38        |
| 3.2.2 Results . . . . .                          | 45        |
| 3.3 Discussion . . . . .                         | 51        |
| <b>Chapter 4 Powers-of-Two Model</b>             | <b>53</b> |
| 4.1 Model Description . . . . .                  | 54        |
| 4.2 Mathematical Model . . . . .                 | 56        |
| 4.2.1 Strengthening Constraints . . . . .        | 61        |
| 4.3 Examples . . . . .                           | 62        |
| 4.3.1 Data . . . . .                             | 62        |
| 4.3.2 Results . . . . .                          | 66        |
| <b>Chapter 5 Control Approaches</b>              | <b>73</b> |
| 5.1 Assumptions . . . . .                        | 75        |
| 5.2 Approach 1 . . . . .                         | 79        |
| 5.3 Approach 2 . . . . .                         | 81        |
| 5.4 Approach 3 . . . . .                         | 83        |

|                   |  |            |
|-------------------|--|------------|
| 5.5               | Approach 4 . . . . .                                 | 85         |
| 5.6               | Approach 5 . . . . .                                 | 87         |
| 5.7               | Example . . . . .                                    | 88         |
| 5.7.1             | Data . . . . .                                       | 88         |
| 5.7.2             | Results and Discussion . . . . .                     | 92         |
| <b>Chapter 6</b>  | <b>Conclusion</b>                                    | <b>103</b> |
|                   | <b>Bibliography</b>                                  | <b>108</b> |
| <b>Appendix A</b> | <b>Formulas for Right-left Cuts</b>                  | <b>115</b> |
| <b>Appendix B</b> | <b>Market Price for Doug Fir Lumber</b>              | <b>116</b> |
| <b>Appendix C</b> | <b>Length of Sub-cuts for Horizontal Above-below</b> | <b>117</b> |
| <b>Appendix D</b> | <b>Length of Sub-cuts for Vertical Above-below</b>   | <b>119</b> |
| <b>Appendix E</b> | <b>Length of Sub-cuts for Vertical Right-left</b>    | <b>121</b> |
| <b>Appendix F</b> | <b>Length of Sub-cuts for Horizontal Right-left</b>  | <b>123</b> |
| <b>Appendix G</b> | <b>Campaign Definitions</b>                          | <b>125</b> |
| <b>Appendix H</b> | <b>Campaign Data</b>                                 | <b>127</b> |
| <b>Appendix I</b> | <b>Fractional Outputs (<math>\alpha_{cp}</math>)</b> | <b>128</b> |
| <b>Appendix J</b> | <b>Example of Cycle Stock Inventory Estimate</b>     | <b>142</b> |
| <b>Appendix K</b> | <b>Cutting Pattern Code</b>                          | <b>143</b> |
| <b>Appendix L</b> | <b>GLPK Code</b>                                     | <b>166</b> |
| <b>Appendix M</b> | <b>Simulation Codes</b>                              | <b>168</b> |

# List of Tables

|     |  |    |
|-----|--|----|
| 2.1 | Data for ELSP example . . . . .                                      | 11 |
| 3.1 | Values for $W_2^m$ . . . . .   | 23 |
| 3.2 | Formulas for above-below cuts . . . . .                              | 26 |
| 3.3 | Nominal lumber dimensions . . . . .                                  | 38 |
| 3.4 | Nominal, target and actual data . . . . .                            | 38 |
| 3.5 | Volume percent yields in comparison case 1 . . . . .                 | 46 |
| 4.1 | Product data for example 1 . . . . .                                 | 63 |
| 4.2 | Product data for Example 2 . . . . .                                 | 64 |
| 4.3 | PoT solution, Example 1 . . . . .                                    | 66 |
| 4.4 | Total approximate cycle stock inventory of each product, Example 1   | 67 |
| 4.5 | PoT solution, Example 2 . . . . .                                    | 69 |
| 4.6 | Total approximate cycle stock inventory of each product, Example 2   | 70 |
| 5.1 | Case 1: Basic data of stochastic demand generation process . . . . . | 90 |
| 5.2 | Case 2: Basic data of stochastic demand generation process . . . . . | 90 |

# List of Figures

|      |  |    |
|------|--|----|
| 1.1  | The proposed three-stage mechanism . . . . .                             | 4  |
| 2.1  | Schedule for the ELSP example with zero setup times . . . . .            | 12 |
| 3.1  | Methodology . . . . .  | 17 |
| 3.2  | A typical log . . . . .  | 17 |
| 3.3  | Assumed sawing process . . . . .   | 18 |
| 3.4  | Lumber with wane . . . . .   | 19 |
| 3.5  | A typical cutting pattern . . . . .                                      | 19 |
| 3.6  | Main cut . . . . .   | 23 |
| 3.7  | A pattern with above-below and right-left cuts . . . . .                 | 24 |
| 3.8  | Eligible pattern . . . . .   | 32 |
| 3.9  | $rWaneUD$ and $rWaneSide$ . . . . .                                      | 34 |
| 3.10 | Touching point and lumber length . . . . .                               | 35 |
| 3.11 | Algorithm for generating campaigns . . . . .                             | 37 |
| 3.12 | Minitab probability plot for small end radius . . . . .                  | 40 |
| 3.13 | Real data for log class 2 . . . . .                                      | 41 |
| 3.14 | Fitted uniform distribution for log class 2 . . . . .                    | 41 |
| 3.15 | Price fit results . . . . .  | 43 |
| 3.16 | Price list 2, log classes 1-2 . . . . .                                  | 45 |
| 3.17 | Price list 1, log classes 3-7 . . . . .                                  | 49 |
| 3.18 | Log class 2, price lists 6-9 . . . . .                                   | 49 |
| 3.19 | Log class 1, price lists 10-13 . . . . .                                 | 50 |
| 3.20 | Log class 2, price lists 16-20 . . . . .                                 | 50 |
| 4.1  | Difference between $I_{cp}$ and $\tilde{I}_{cp}$ . . . . .               | 60 |
| 4.2  | Demand of products, Example 1 . . . . .                                  | 63 |
| 4.3  | Demand of products, Example 2 . . . . .                                  | 65 |
| 4.4  | Total approximate cycle stock inventory, Example 1 . . . . .             | 66 |
| 4.5  | Demand vs. supply, Example 1 . . . . .                                   | 67 |
| 4.6  | Impact of strengthening constraints on model's performance . . . . .     | 68 |
| 4.7  | Total approximate cycle stock inventory, Example 2 . . . . .             | 69 |
| 4.8  | Demand vs. supply, Example 2 . . . . .                                   | 70 |
| 5.1  | Approach 1 flowchart . . . . .   | 80 |
| 5.2  | Approach 2 flowchart . . . . .   | 83 |
| 5.3  | Approach 3 flowchart . . . . .   | 85 |
| 5.4  | Approach 4 flowchart . . . . .   | 86 |
| 5.5  | Approach 5 flowchart . . . . .   | 88 |
| 5.6  | Case 1: Average demand size per order ( $\frac{D_p}{Rate_p}$ ) . . . . . | 91 |
| 5.7  | Case 2: Average demand size per order ( $\frac{D_p}{Rate_p}$ ) . . . . . | 91 |
| 5.8  | Case 1: Demands of 7 products over the first year . . . . .              | 93 |
| 5.9  | Case 2: Demands of 7 products over the first year . . . . .              | 93 |



|      |   |    |
|------|---|----|
| 5.10 | Case 1: Inventory status of 7 products using Approach 1 . . . . .                       | 94 |
| 5.11 | Case 1: Inventory status of 7 products using Approach 2 . . . . .                       | 94 |
| 5.12 | Case 1: Inventory status of 7 products using Approach 3 . . . . .                       | 94 |
| 5.13 | Case 1: Inventory status of 7 products using Approach 4 . . . . .                       | 95 |
| 5.14 | Case 1: Inventory status of 7 products using Approach 5 . . . . .                       | 95 |
| 5.15 | Case 1: Weekly campaign schedules for the first 16 weeks of 5 Ap-<br>proaches . . . . . | 95 |
| 5.16 | Case 2: Inventory status of 7 products using Approach 1 . . . . .                       | 96 |
| 5.17 | Case 2: Inventory status of 7 products using Approach 2 . . . . .                       | 96 |
| 5.18 | Case 2: Inventory status of 7 products using Approach 3 . . . . .                       | 96 |
| 5.19 | Case 2: Inventory status of 7 products using Approach 4 . . . . .                       | 96 |
| 5.20 | Case 2: Inventory status of 7 products using Approach 5 . . . . .                       | 97 |
| 5.21 | Case 2: Weekly campaign schedules for the first 16 weeks of 5 Ap-<br>proaches . . . . . | 97 |
| 5.22 | Case 1: Number of runs of each campaign over the first year of<br>simulation . . . . .  | 97 |
| 5.23 | Case 2: Number of runs of each campaign over the first year of<br>simulation . . . . .  | 98 |

# Abstract

To control the lumber production in a sawmill, a three-stage system is proposed.

First, a quick program creates many cutting patterns and chooses the most valuable pattern for each log within a log class given a price list. A combination of a log class and a price list resulting in a set of lumber output proportions creates a “campaign”.

Second, a Powers-of-Two optimization model calculates “campaign lot sizes” to minimize inventory and meet deterministic demand. The goal is to develop a minimum cycle stock inventory level for all the products over a time horizon.

Third, five control approaches are created based on the results of the PoT model and evaluated using simulation environment to monitor inventory levels in the case of stochastic demand.

This research indicates that in a divergent-stochastic environment such as a sawmill, situations with noisy batch order arrivals do pose difficulties when Powers-of-Two control approaches are used.

# List of Abbreviations Used

|       |  |
|-------|--|
| BOF   | Best Opening Face                          |
| BP    | Basic Period                               |
| CC    | Common Cycle                               |
| CP    | Constraint Programming                     |
| CT    | Computed Tomography                        |
| DP    | Dynamic Programming                        |
| EBP   | Extended Basic Period                      |
| ELSP  | Economic Lot Scheduling Problem            |
| EOQ   | Economic Order Quantity                    |
| GA    | Genetic Algorithm                          |
| HH    | Haessler's Heuristic                       |
| JIT   | Just in Time                               |
| MILP  | Mixed Integer Linear Programming           |
| MIP   | Mixed Integer Programming                  |
| PoT   | Powers of Two                              |
| SELSP | Stochastic Economic Lot Scheduling Problem |

# Acknowledgments

I would like to express my deepest gratitude to Dr. Eldon Gunn for his continuous encouragement, expert guidance and positive attitude throughout this study. This thesis would not have been possible without his invaluable comments and advice in all stages of the work.

I would also like to extend my sincerest thanks and appreciation to Dr. M. Mahdi Tajbakhsh, for providing me with the opportunity to follow my dreams and pursue my education in Canada. His endless enthusiasm, encouragement and support has been extremely valuable for the commencement and completion of this work.

A special thanks goes to Dr. Corinne MacDonald, for her valuable comments and great collaboration during the research. I would like to thank dear committee members Dr. John Blake and Dr. H. I. Gassmann for agreeing to be my examiners on short notice. I wish to thank my colleague and friend, Sina Saadatyar, for his excellent contribution in completing the joint chapter of this thesis.

I am also very grateful to Dr. Uday Venkatadri, head of the department of Industrial Engineering, and Cindi Slaunwhite, secretary of the department of Industrial Engineering, for providing great research services and facilities.

I wish to thank GNU Group, Python Development Team and Gusek Company for providing great open-source softwares and tutorials that made this research possible.

Last but not least, I would like to express my warmest thanks to my husband, Mehran Zamani, and my parents for continuing to believe in me and heartily supporting me throughout the way.

# Chapter 1

## Introduction

This chapter presents an introduction to the sawmill operations as well as an outline of the remainder of the thesis. Section 1.1 is a general description of the sawmill processes, while Section 1.2 presents a description of the problem and the objectives of the thesis.

### 1.1 Sawmill Operations

A sawmill is a capacity constrained facility with a divergent production system where many products can be produced from a given set of logs. The basic processes for producing lumber include logging, log-bucking, limbing, decking, debarking, sawing, drying and planing. Logging is the process of cutting, skidding and loading the tree logs onto trucks to transport them from the forest to the sawmill. Logs are sometimes cut to length in the log-bucking process and their branches are removed in the limbing stage. In the decking stage, the logs can be sorted based on a number of characteristics including size, species or end use before their bark is removed in the debarking process. Not all sawmills engage in all of these stages, depending on facility capacity, technology and production policies, they may combine or eliminate some stages.

Sawing is the process where the log is broken down into lumber pieces using

several mills equipped with scanners and optimizers. Scanners are located in front of the sawing machine and provide a detailed outline of the shape and dimensions of a given log. The optimizer, a computer program, then calculates the best cutting pattern among all available patterns, the one that results in maximum lumber value with reference to a set of lumber prices (known as a “price list”). Logs usually arrive in batches where they have been classified according to specifications such as, length, diameter, species, etc. These batches are called “log classes”.

The sawing stage is typically broken down into two separate steps. First, logs are cut into large flat pieces called cants. The cant will be broken down into lumber pieces through a specific sawing machine such as gang saw. In addition to the resulting lumber pieces, the remaining parts called flitches, are produced. These flitches may go on to the edger, trimmer or re-sawing stages in which dimensional lumber pieces can be produced.

The lumber produced in the sawing stage is divided into different batches based on similarities such as moisture content, size, and species to feed a kiln where they lose their moisture. In some cases the lumber might be air-dried before kiln drying operations. Any necessary finishing is done on the planers. The drying and planing processes are batch-based with consideration given to setup time and capacity. APICS (American Production and Inventory Control Society) defines a continuous process to be the one that has minimum number of interruptions for any one production run or between different runs of similar processes (Fransoo and Rutten [16]). By this definition sawmills are considered to be process industries.

Sawmill products can be classified into two categories: High-value products, such as high quality lumber pieces, and low-value products, such as lower quality lumber pieces, wood chips, saw dust and residual bark. The quality of lumber pieces, known as grade, is based on the number and location of internal defects, as well as final dimensions. The pieces are graded and from this grade their value is derived. Though high-value products are more valuable, it is important that sawmills produce low-value products to supply other sectors of the forestry

industry, including the pulp and paper, pellet, and biofuel industries.

## 1.2 Problem Description

In the modern, competitive marketplace, companies are, by necessity, implementing lean manufacturing concepts in their processes (Ray et al. [36]). Although principles of lean production, such as setup time reduction, inventory reduction and Just-in-Time (“JIT”) delivery, have been widely used in different sectors and especially in discrete manufacturing sectors, these principles have not been as fully implemented in process industries such as the sawmill industry.

In a sawmill, a given set of logs can be transformed into multiple products. Consequently, one product may be produced from different log classes. One key to the diversification of products from each log class (a set of logs classified based on certain characteristics) is the product price list given to the cutting pattern optimizers during the sawing process. The challenge lies in planning and executing a lumber production process that can fulfill uncertain demands while keeping the inventory requirements as low as possible.

This research proposes a 3-stage procedure (shown in figure 1.1). In the first stage, a joint work with Sina Saadatyar<sup>1</sup>, a fast and novel algorithm is proposed to find the best cutting patterns for each class of logs, based on a given price list. The algorithm estimates the output of a certain class of logs with regard to lumber recovery and is conducted prior to the actual sawing process. We define a “campaign” as a choice of log classes with certain characteristics under a given price list that results in a set of lumber outputs. The campaign concept is proposed to simplify the joint production process. Thus, instead of working with individual products, we will be working with campaigns.

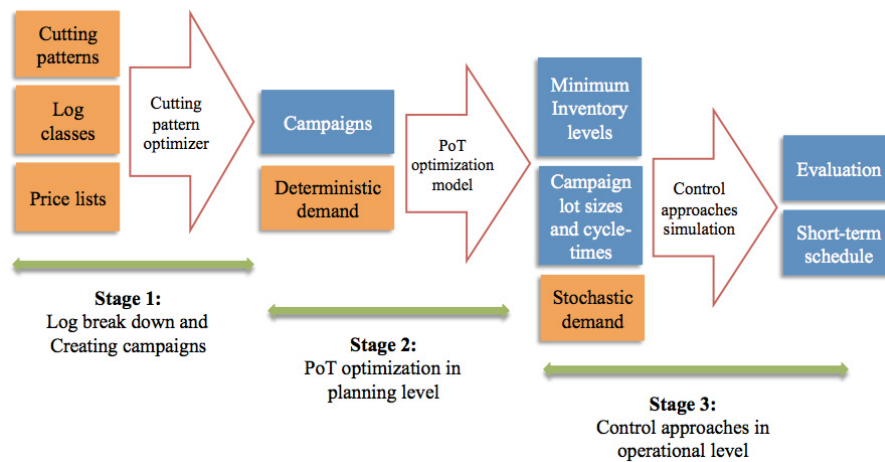
The second stage involves an optimization model based on the Powers-of-Two policy (a version of ELSP) to calculate how to fulfill constant deterministic de-

---

<sup>1</sup>See his thesis (Saadatyar [41])

mands. This model is a campaign lot sizing model in which the objective is to keep inventory levels to a minimum while being able to respond to demand and capacity requirements.

The final stage examines various control approaches for campaign scheduling by simulating how they respond to uncertainty in demand. The goal of the final stage is to examine if it is possible to control inventory using the PoT lot sizes in a situation of uncertain demands.



**Figure 1.1:** The proposed three-stage mechanism

To complete the objectives set forth, Chapter 2 presents a literature review on the background and conventional approaches. Chapter 3 explains the procedure used to create different campaigns. Chapter 4 sets forth the details of the Powers-of-Two optimization model and Chapter 5 describes the simulation of the control strategies under various scenarios. Finally, conclusions and further areas of research are presented in Chapter 6.



# Chapter 2

## Literature Review

This chapter presents a literature review on the three main areas to be discussed; log breakdown, lumber production planning and ELSP (Economic Lot Scheduling Problem) and Powers-of-Two.

### 2.1 Log Breakdown

Lumber is produced from the breaking down logs through a series of cutting operations. Sawmills apply various approaches based on the capabilities of their scanners and optimizers to get the maximum yield from each log conversion. Steele [45] identified seven major factors affecting lumber recovery: log specifications, kerf width (the width of a notch or groove made by a saw), sawing variation, product mix, management decision making, equipment maintenance and sawing method. An investigation of the relationship between log specifications (log diameter, length and taper) and the lumber yield of eight sawing systems using the Best Opening Face (“BOF”) program (Lewis [29]) is conducted by Hallock et al. [21] to propose a potential framework for sawing system selection when the log input is known.

Although there are a large number of classifications for log sawing patterns, they can be categorized into two general groups: common methods such as live-

sawing, sawing with a breaking cut and quarter sawing and potential methods such as sawing around 90 and 180-degree angles and tangential sawing (How et al. [24]). Each cutting method has its own distinct advantages and disadvantages based on its applicability to large or domestic sawmills, production speed, production cost, ability to produce high quality lumber, shape of logs and degree of mechanization.

Most sawmills optimize their log cutting processes by only considering the external shape of logs and the information regarding the desired final products with no knowledge about any internal defects in the logs. By applying internal log scanning tools such as Computed Tomography (“CT”) to detect internal defect of logs, lumber yield can be improved. An extensive discussion of this type of optimization can be found in the works done by Bhandarkar et al. [3] and Rinnhofer et al. [38].

The majority of models involving sawing optimization use dynamic programming (“DP”) principles. In 1981, Tejavibulya [48] proposed two DP approaches for live-sawing and cant-sawing methods without consideration of log taper or defects. Todoroki and Ronnqvist [51] developed a DP formulation that integrates the primary (converting logs into slabs) and secondary (edging and trimming to generate final boards) phases of the log breakdown based on live-sawing patterns that consider interior defects. Reinders and Hendriks [37] proposed an optimization approach based on nested DP sub-algorithms for the conversion of logs to lumber. In addition to the DP optimization algorithms, heuristic algorithms such as the Genetic Algorithm (“GA”) are also being used in the optimization of lumber cutting patterns (Cook and Wolfe [12]).

Some challenges the sawmill industry faces, include variable demands, competition and increased utilization of wood as a raw material (Björheden [5]). As a result, sawmills commonly use computer simulation tools in different stages of their operations. There are a number of simulation packages used to increase lumber recovery and improve management decision-making such as BOF, SIMSAW (Singmin and Africa [44]), SAW3D (Zeng [55]), SAW3DG (Zeng [56]) and

AUTOSAW (Todoroki et al. [52]). Each has distinctive capabilities with regards to log representation, internal defect considerations and the flexibility of the log breakdown procedures (Todoroki and Ronnqvist [50]).

Another challenge facing the sawmill industry is the cutting stock problem that tries to find the best way to cut lumber into dimension parts to fulfill their customer (such as furniture companies) demand while being cost effective. This problem has been widely discussed in the literature and solutions have proposed a number of techniques, such as knapsack-based optimizations and heuristics (Carnieri et al. [9], Carnieri et al. [8], Carnieri and Mendoza [7], Gilmore and Gomory [19]).

In summary, a number of techniques have been designed for and applied to the process of breaking down logs to lumber pieces and dimensional parts. These techniques have been developed and continue to evolve quickly as a result of the dynamic and competitive nature of sawmilling technology and the lumber market as a whole.

In this thesis, we do not consider the entire range of sawing techniques. Instead, we focus on fairly simple patterns used in many commercial softwood sawmills. These essentially consist of patterns that create a large cant suitable for subsequent breakdown by a gang saw as well as flitches that can be broken down subsequently in an edger. These patterns are discussed in more detail in chapter 3.

## **2.2 Lumber Production Planning**

Important measures of sawmill performance include throughput and yield from a sawmill's cutting process. However, the ability of a sawmill to fulfill diverse demands is crucial. Models that relate the customer needs to the log breakdown optimization stage with the consideration of product value are crucial. An adaptive control model called product-controlled optimization was proposed by Todoroki and Ronnqvist [49] to fulfill the demand of different timber grades while dealing with dynamic timber values. The model was implemented using AUTOSAW;

the results demonstrated that optimization controlled by dynamic product values better utilizes the log supply in fulfilling customer needs than when the production is controlled by pure value-optimization or volume-optimization.

Mendoza et al. [34] developed an integration of optimization and simulation which maximizes economic efficiency in optimization levels. This generates an optimal mix of log inputs to fulfill demand while the simulation process checks the feasibility of the plan for short-term operations, taking into account the flow of raw materials and lumber inventory.

Most North American sawmilling companies use a “push production” strategy for their lumber production planning to simplify their operations and allow for continuous running of production (Gaudreault et al. [18]). This strategy attempts to create maximum yield from available raw material and resources without substantial consideration of actual customer demand. In such an approach, the focus is on making lumber efficiently and then attempting to sell it. Conversely, “pull production” uses customer demand as input to the production process, adjusting production to meet demand on schedule. However, due to the divergent nature of the throughput and limitations on the accessibility of raw material, it is not possible to implement pull production in its purest sense where the production process begins only when a specific order is received. It is true that sawmills do attempt to modify their log purchase and processing strategies to match demand. However this is usually ad hoc and there are few formal procedures that facilitate this. As a result, there is a need to implement an integrated push-pull system that achieves the benefit of both.

To design such a system and investigate the impact of push-pull integration, Gaudreault et al. [18] developed a platform that first models the production planning and scheduling of each stage of the lumber production process (sawing, drying and planing) based on a Mixed Integer Programming (“MIP”) approach and then proposed three coordination mechanisms to demonstrate the applicability of this platform. This platform is also used by Cid-Yanez et al. [11] and Cid-Yanez et al.

[10] to evaluate the impact of different decoupling point positions on lumber production performance through a case study conducted in Quebec using simulation tools.

Since production planning and scheduling in divergent production systems is complex, it is necessary to implement a plan which reflects the details of the processes. There have been a number of studies on this topic with regard to sawmill operations. Gaudreault et al. [17] compared two modelling approaches of MIP and Constraint Programming (“CP”) for the integration of production planning and scheduling in drying and finishing operations. Kazemi Zanjani et al. [27] proposed and compared two models based on robust optimization for sawmill production planning considering uncertainty in logs as raw materials. Another recent work (Alvarez and Vera [2]) discussed the application of robust optimization in sawmill planning derived from a linear programming model that provides a tool to investigate “tradeoff between robustness requirements and loss in optimality”. To provide a dynamic model that proactively deals with uncertainty in sawmill production planning, Maturana et al. [32] developed a scheduling model that deals with uncertainties by using a rolling planning horizon and a heuristic that serves as a benchmark. The advantage of their model is that it is solved for the first period and is progressively updated for each of the following periods as unexpected events arise.

The analytic work discussed above has generally been implemented with examples that consider only very limited representation of the product mix and sawing options in the sawmill. The work that we discuss in this thesis is aimed at studying problems at a more realistic scale and framework.

## **2.3 ELSP and Powers-of-Two**

A significant challenge facing a multi-product manufacturing environment is how to cyclically schedule production of a set of products, with different setup times,

on a single machine to minimize overall inventory and setup costs while meeting customer demand. This is the essence of the Economic Lot Scheduling Problem (“ELSP”) in which demand is deterministic and constant over an infinite horizon. The ELSP has been widely studied since 1957 by Eilon [13] and Rogers [39] where the Economic Order Quantity (“EOQ”) model was extended to the case of multiple products produced on a single machine to find optimal lot sizes (batches) and their run schedules. The complexity in solving an ELSP arises from this fact, that a single machine can only process at most one production run at any point of time. Thus, when two or more production runs compete for being processed on the machine, “interference” may occur. The challenge is how to cyclically process the production runs so that the associated cost is minimized (Elmaghraby [14]). However, no optimal policy has been found to solve this problem. The existing solution approaches are mainly based on heuristics and methods under specific limitations and policies that provide near optimal solutions (Sun et al. [47]).

Economic lot scheduling problems, in the form of cyclic schedules known as EPEI (“every product every interval”) have become a key focus of the lean manufacturing literature (Kerber and Dreckshage [28]). Bicheno et al. [4] show that it is possible to improve on a straightforward cyclic schedule.

The ELSP problems can be categorized according to three main approaches; Common Cycle (“CC”) (Hanssmann [22]), Basic Period (“BP”) (Bomberger [6]) and Extended Basic Period (EBP) (Narro Lopez and Kingsman [35]). The simplest and most restrictive policy is the CC policy in which the objective is to find a cycle time where each production run can be fitted in, exactly once. This forces each product to be produced once in each cycle. To illustrate the general form of ELSP with the CC policy, we assume a single machine can process  $n$  different products. The production rate of product  $i$  is  $p_i$  and the associated sequence-independent setup time is  $s_i$ . The demand rate for product  $i$  is set to be  $d_i$ . Related holding cost per unit per unit time and setup cost for product  $i$  are  $h_i$  and  $a_i$ , respectively. The problem seeks the best cycle ( $T$ ) that can accommodate the production run of each

product once, where the cycle length of all products are identical ( $T_i = T$ ). The objective function is the minimization of total cost including setup and holding cost. Formulation 2.1 demonstrates the total average cost per unit time.

$$\text{Minimize } TC = \sum_{i=1}^n \left( \frac{a_i}{T} + \frac{1}{2} h_i \left( d_i T - \frac{d_i^2 T}{p_i} \right) \right) \quad (2.1)$$

$$\text{Subject to } \sum_{i=1}^n \left( s_i + \frac{d_i}{p_i} T \right) \leq T, \quad (2.2)$$

The time of production run of product  $i$  in a cycle is  $\frac{d_i T}{p_i}$  to fulfill demand  $d_i$ . Therefore, if initial inventory is zero at the beginning, the inventory of product  $i$  will increase until amount  $(p_i - d_i) \frac{d_i T}{p_i}$ . After that point, the inventory will decrease at rate  $d_i$ . Consequently, the average inventory is calculated as  $\frac{1}{2} (p_i - d_i) \frac{d_i T}{p_i}$ . Constraint 2.2 indicates that the total setup time and production time for all products should be less than or equal to the cycle time  $T$ . The optimal cycle time is obtained by taking derivative of the  $TC$  function and setting it equal to zero. It is calculated based on formulation 2.3.

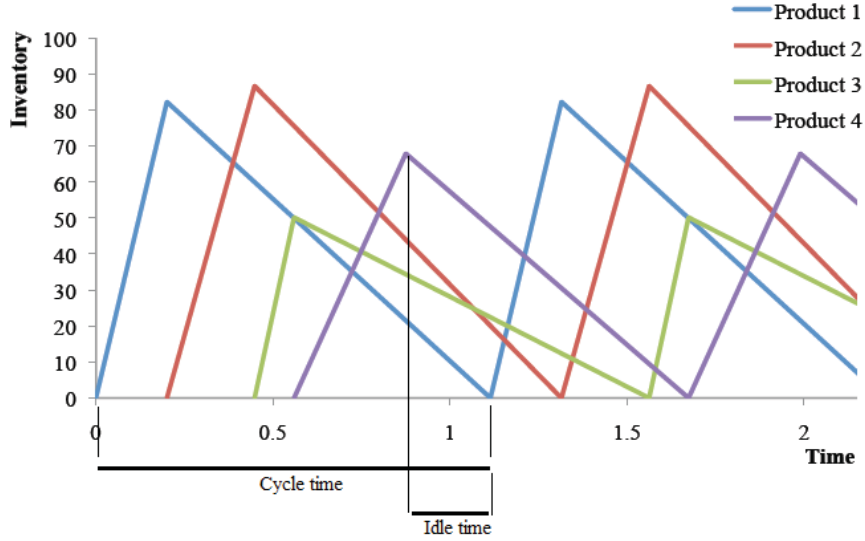
$$T^* = \sqrt{\frac{\sum_{i=1}^n a_i}{\left( \sum_{i=1}^n \frac{h_i d_i (p_i - d_i)}{2 p_i} \right)^{-1}} \quad (2.3)$$

A simple example of ELSP with CC policy including 4 products with the given production rate, demand rate, setup cost and holding cost, presented in Table 2.1, is provided to show how the model works.

| <i>Products</i> | $d_i$ | $p_i$ | $h_i$ | $a_i$ |
|-----------------|-------|-------|-------|-------|
| 1               | 90    | 500   | 40    | 2000  |
| 2               | 100   | 450   | 10    | 2500  |
| 3               | 50    | 500   | 30    | 800   |
| 4               | 85    | 300   | 70    | 500   |

Table 2.1: Data for ELSP example

The optimal cycle time is  $T^* = 1.114$  and the total cost per unit time is  $TC = 10411.04$ . Figure 2.1 illustrates the cycle time and inventory level of each product.



**Figure 2.1:** Schedule for the ELSP example with zero setup times

If setup times are sequence-independent and the summation over them is less than the idle time, the calculated  $T^*$  is optimal. Otherwise, the optimum cycle time is calculated based on formula 2.4 in which the production process uses the maximum available capacity.

$$T^* = \frac{\sum_{i=1}^n s_i}{(1 - \sum_{i=1}^n \frac{d_i}{p_i})} \quad (2.4)$$

If setup times are sequence-dependent, the problem is solved by first finding the sequence that minimizes total setup time and using these setup times in equation 2.4.

A key issue for any ELSP model, pure cyclic as illustrated in figure 2.1 or otherwise, is the need to have a “just right inventory” of all the other products when you begin production of any particular product. As can be seen in Figure 2.1, the inventory of products 2, 3, 4 when we begin production of product 1 must be enough to supply demand of those products while they are not being produced.

As pointed out in Silver et al. [43], pure cyclic rotation policies are not necessarily optimal. If the demand for some products is much larger than the demand for others it may make sense to run the lower demand products less frequently.



Bicheno et al. [4] illustrate this point.

The two other categories of the ELSP models deal with a time interval known as a basic period ( $B$ ), for the setup and production of all or most of the products. The objective is to find the basic period ( $B$ ) and a multipliers vector of  $K(B) = \{k_i \mid B\}_{i=1}^n$  (Yao and Elmaghraby [54]), where each product's ( $i$ ) cycle time is an integer multiple of a basic period ( $k_i B$ ). If the basic period is long enough to accommodate all the production runs, the problem is categorized as BP.

However, in EBP, each basic period can accommodate only a number of production runs. In both cases, each product's ( $i$ ) cycle time  $T_i$  will be equivalent to  $k_i B$ , where  $k_i$  is an integer. The difference between BP and EBP is in how the capacity constraint (formula 2.2 in CC approach) is represented. As Yao and Elmaghraby [54] reported, the capacity constraint for the proposed General Integer (GI) BP approach can be presented by  $\sum_{i=1}^n (s_i + \rho_i k_i B) \leq B$ , where  $\rho_i = \frac{d_i}{p_i}$ . This constraint must be replaced with other sets of capacity constraints in the EBP approach such as the one proposed by Sun et al. [46] in which the feasibility constraint is represented as  $\sum_{i \in S_k} (s_i + \rho_i k_i B) \leq B$ , where  $S_k$  is the set of products produced within each basic period  $k$ . ( $1 \leq k \leq \max_i \{k_i\}$ ).

It is sometimes desirable to restrict the multipliers of the basic period. One such restriction is the Powers-of-Two (PoT) policy which sets the multipliers to the powers of 2 ( $k_i = 2^p, p \geq 0; integer$ ). This policy has been widely used as it presents economical and near optimal solutions in a number of contexts. Maxwell and Muckstadt [33] demonstrated that the application of PoT is highly desirable from an economic point of view and many schedulers have benefitted from its application. Yao and Elmaghraby [54] investigated the optimality structure of the unconstrained ELSP with BP approach under the PoT policy and proposed an effective search algorithm that provided a global-optimal solution. They suggested PoT framework provides researchers with the opportunity to apply more efficient and easier heuristics to various ELSP problems. In another work by Sun et al. [47], the problem of EBP under the PoT policy has been considered in which the

optimality conditions were compared with Haessler’s Heuristic (“HH”) (Haessler [20]). They demonstrated that their algorithm is faster and finds better feasible solutions than HH. As presented by Jackson et al. [26], the basis of the PoT policy is used to solve the problem of joint replenishment. They demonstrated even in worst-case scenario situations, their model could respond in a bound of 94 percent of optimality. The importance of identifying cost effective replenishment methods in all the parts and stages of production/distribution systems is undeniable. PoT has been successfully applied in these systems. For instance, Federgruen and Zheng [15] and Maxwell and Muckstadt [33] researched the optimal PoT replenishment methods for capacitated and uncapacitated systems respectively.

ELSP deals with constant and deterministic parameters. In the case of stochastic demand, production rate or setup times, the ELSP problem is referred to as the Stochastic Economic Lot Scheduling Problem (“SELSP”). There is an extensive literature on these topics in Silver et al. [43]. The most common feature in the literature is a search for quick and easy solutions to the problem. Many of the proposed approaches are based on heuristics in attempt to find practical and near optimal solutions.

In the research of this thesis, we cannot schedule production of individual products separately because of the divergent nature of sawmill production where many products are produced from a single log. Instead we show how to extend the economic lot scheduling problem to consider campaign lot sizes, not product lot sizes, and demonstrate that we can solve this problem using a Powers-of-Two approach. The campaign concept is discussed in Chapter 3. The details of the proposed PoT model is described in Chapter 4.

# Chapter 3

## Creating Campaigns

<sup>1</sup>In the sawmilling process, one significant challenge is to estimate the production of lumber from a given set of logs. This is closely related to the question of what logs need to be sawed to meet a given demand for lumber. As previously discussed, most sawmills sort their input logs according to certain specifications prior to the sawing processes. This classification may be done according to size, species, or any number of other characteristics to form “log classes”. Log sawing is an automated process where logs are scanned and sawn in an attempt to maximize value. One key factor to diversify outputs of a certain log class is the value list (known as “price list”) of the outputs which directs implementing cutting patterns on each log in the class. For a given log, with a specific price list, the cutting pattern optimization results in a specific set of outputs. For a class of logs with a characteristic distribution of diameters for that class, a specific price list will result in a specific set of proportional product outputs for all the products that can be produced from that class. A “campaign” is then considered as a combination of a log class given a price list which yields a specific set of outputs through implementation of a set of optimized cutting patterns.

This chapter first details the methodology and its assumptions for creating

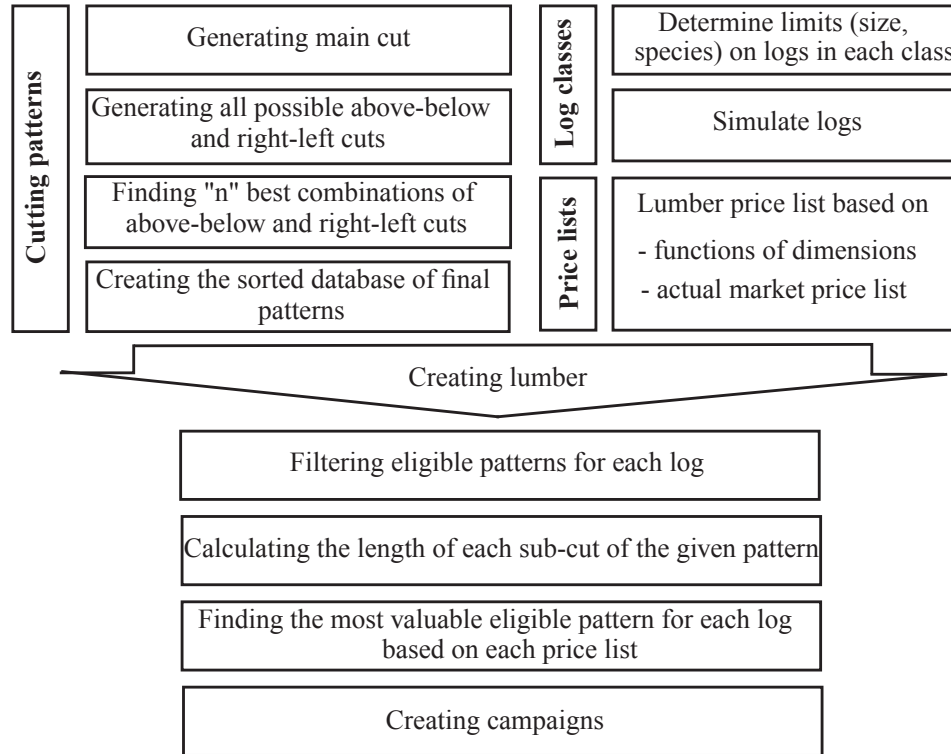
---

<sup>1</sup>The work behind this chapter is a joint work with Sina Saadatyar. See also his thesis (Saadatyar [41])

campaigns through the following steps: 3.1.1) generating cutting patterns, 3.1.2) generating log classes 3.1.3) creating price lists and 3.1.4) creating lumber output fractions. Section 3.2 demonstrates the application of an algorithm to produce different campaigns with a discussion of important parameters surrounding the algorithm. Section 3.3 discusses some issues regarding the algorithm.

## 3.1 Methodology

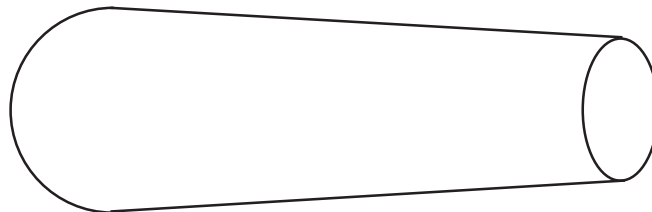
This framework consists of four stages. The first stage provides an algorithm to create a set of possible cutting patterns. These patterns are then stored in a database for future use. In the second stage, log classes are simulated based on certain characteristics. Price lists are then generated to feed the algorithm in the third stage. In the last step, the algorithm uses these inputs to simulate how the sawmill optimizers will perform to break down the logs and transform them into lumber and thus generate product output proportions for the campaign. Figure 3.1 summarizes the methodology used for generating campaigns. The programming language used for the algorithms is Python 2.6.6 (Rossum [40]). Details of the Python code are provided in Appendix K. Python is an open-source programming language and the intention is that our code is readily available and usable.



**Figure 3.1:** Methodology

The following considerations are key to our approach:

- The proposed methodology is implementable in softwood sawmills. This is because hardwood sawmills typically have more complex sawing strategies.
- Logs are considered as truncated cones, without any defects and curves as shown in figure 3.2. Although this assumption does not perfectly reflect the real world situation, it provides a reasonable estimation of actual logs.

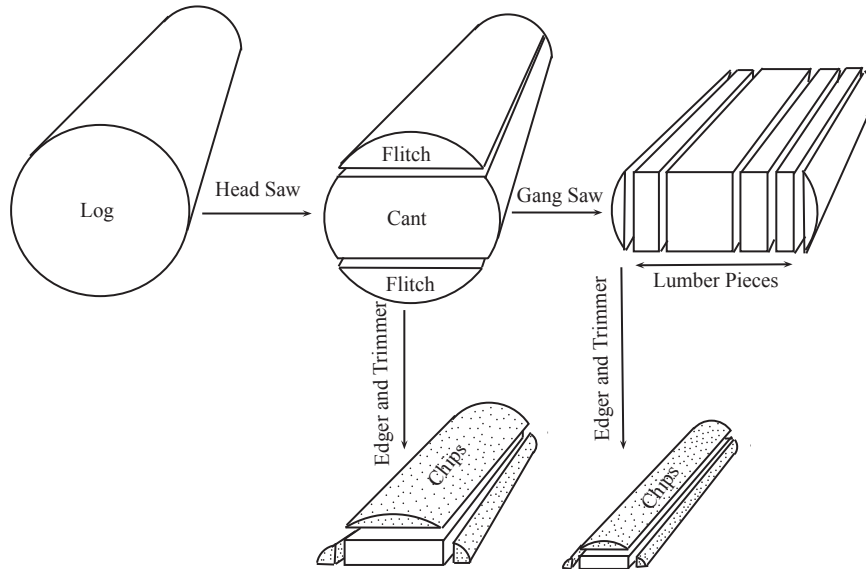


**Figure 3.2:** A typical log

- The main products are lumber pieces. We do not consider chips, dust and

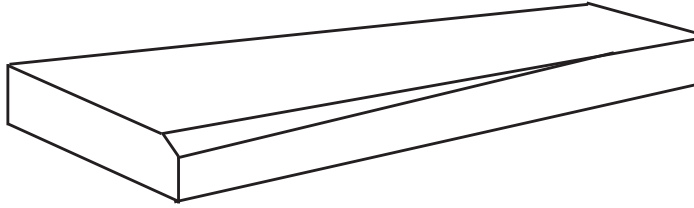
shaving as products that enter into the overall economic decision. However, this is readily modified.

- The assumed sawing procedure includes the following processes: First, initial cuts are made through a head saw in which a cant and two flitches are produced; then the cant is broken down into dimensional lumber pieces and flitches through a gang saw. If the flitches are capable of producing lumber, they can go to a re-sawing process, presumably in an edger. Figure 3.3 illustrates the assumed sawing process.



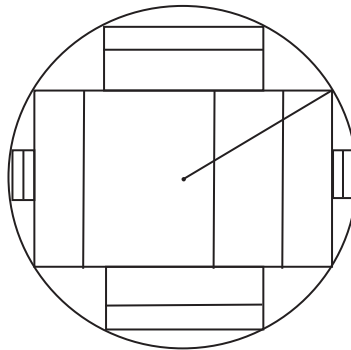
**Figure 3.3:** Assumed sawing process

- Wane on some pieces of lumber is allowed but limited (e.g. all the lumber pieces are not necessarily premium. This is a parameter that the user can set but in the work reported here, it is assumed that at most 25% of each side can have wane). Figure 3.4 illustrates a piece of lumber with wane.
- The proposed cutting patterns consist of one main cant, two potential above-below and two potential right-left sets of edge cuts. We assumed the cant's sub-cuts can be non-symmetric. However, above-below cuts are symmetric



**Figure 3.4:** Lumber with wane

to each other as well as right-left cuts. This assumption is shown in figure 3.5.



**Figure 3.5:** A typical cutting pattern

- The main cut has vertical sub-cuts. However, above-below and right-left cuts can have either vertical or horizontal sub-cuts..
- The dimensions of lumber are known in the format of “width x thickness x length”. In the market, lumber is sold based on “nominal” dimensions not “actual” ones. Due to the loss in the processes following sawing (kiln and planner), “target” dimensions are used in sawing operations. The “target” setting guarantees that final products end up with “actual” sizes. In other words, the cuts are slightly larger than the desired finished dimensions, depending on the sawmill. For instance, the actual dimensions of a “2x6x10” product are “1.5x5.5x10”. To get this product, the target dimensions are set as “1.66x5.875x10”. The 0.16 and 0.375 differences between actual and target numbers are considered because of the shrinkage and loss in the kiln

and planner respectively.

- We considered allowances, which is the difference between “target” and “actual”, for each dimension of the lumber except its length.
- Results of the final products are reported by “nominal” numbers.

### 3.1.1 Generating Cutting Patterns

Sawmills are highly automated manufacturing systems. This automation chooses the best cutting pattern to apply to a given log. Scanners are available in front of the head saw that allows a detailed outline of its dimensions. A computer then uses this scan to choose from a set of allowable patterns, the pattern that will give the maximum value to the lumber produced from that log, given a certain price list for the types of lumber products produced. We use a simplifying assumption that all logs are truncated cones. In a real scanner, the complexity and unique shape of each log due to its biological nature affect the cutting pattern chosen. The pattern minimizes waste during the sawing process and maximizes revenue from each log. In an actual mill, there are multiple scanners, not just the head rig but also at the gang saws and edgers. This permits the pattern to be re-adjusted as more information is gained during sawing.

Our approach starts by pre-generating a large set of cutting patterns, sorting these in a database and only later determining which of these cutting patterns is best for each log. The procedure used in creating cutting patterns includes the following steps: 3.1.1.1) Generating main cut; 3.1.1.2) generating all possible above-below and right-left cuts; 3.1.1.3) finding “n” best combinations of above-below and right-left cuts; and 3.1.1.4) creating a database of final patterns. We make a number of assumptions about the properties of the patterns that we generate. Our purpose is not to generate all possible patterns, but to have a convenient method to generate a reasonable set of patterns that can be used in our campaign generation process. Others might legitimately question these and use other



assumptions.

### 3.1.1.1 Generating Main Cut

The main cut is a rectangle (see figures 3.5 and 3.6) whose dimensions determine the smallest diameter log that can accommodate this pattern. It is assumed its thickness can be a choice from a set of allowable thicknesses of final lumber dimensions. This main cut can then be cut vertically along its width into individual sub-cuts. We introduce a set of ratios representing the maximum proportions of a main cut's total width to its thickness. The main point of creating a cant is to exploit most of the log's volume via the gang saw. We restrict the width to thickness ratio of the main cant to be less than or equal to 2, although again this is a user definable parameter. Therefore, for each possible thickness, the maximum total width of the cant is calculated. Then, we generate all possible combinations of sub-cuts. The notations used to formulate this stage are presented as follows:

#### Notations:

|               |   |
|---------------|---|
| $W$           | width set   |
| $T^m$         | thickness set for the main cut  |
| $i$           | index of the elements in $W$ set  |
| $j$           | index of the elements in $T^m$ and $wRatioMax$ sets                       |
| $T_j^m$       | thickness of the main cut when we choose the $j^{th}$ thickness           |
| $W_i$         | width of the $i^{th}$ sub-cut   |
| $W_j^m$       | total width of the main cut when thickness $j$ is chosen                  |
| $X_{i,j}$     | number of sub-cuts with width $W_i$ and thickness $T_j^m$ in the main cut |
| $wRatioMax$   | set of maximum ratios (of total width to the thickness of the main cut)   |
| $wRatioMax_j$ | maximum ratio when thickness $j$ is chosen                                |

|             |  |
|-------------|--|
| $kerf$      | sawing kerf (width of the saw)                                 |
| $W_j^{max}$ | maximum possible width for the main cut when $T_j^m$ is chosen |
| $R$         | radius of the pattern  |

The calculations are shown in the following steps:

**Step 1:** For each thickness  $T_j^m$ , we find the maximum width  $W_j^{max}$ .

$$W_j^{max} = T_j^m * wRatioMax_j \quad (3.1)$$

**Step 2:** For each  $X_{ij}$  find the maximum value according to the formula 3.2.

$$X_{i,j} \leq \frac{W_j^{max}}{W_i} \quad (3.2)$$

**Step 3:** Find all possible combinations of sub-cuts and calculate  $W_j^m$ .

$$W_j^m = \left( \sum_i X_{i,j} * (W_i + kerf) \right) - kerf, \quad \text{where } W_j^m \leq W_j^{max} \quad (3.3)$$

The function used for this step is called “combos” in the Python code (Appendix K).

Following, we present a simple example. Assume  $T^m = \{3, 4\}$ ,  $W = \{2, 4\}$ ,  $wRatioMax = \{2, 2\}$  and  $kerf = 0$ .

For  $j = 2$ ,  $T_2^m = 4$  and  $wRatioMax_2 = 2$ , according to 3.1,  $W_2^{max} = 8$ .

Using formula 3.2, we have  $X_{1,2} \leq 4$ ,  $X_{2,2} \leq 2$ .

Based on 3.3,  $W_2^m$  can have all the values presented in Table 3.1:

Using this procedure we generate all possible main cuts to presumably obtain several patterns for each main cut in the next step. In developing a proposed order for each sub-cut in a pattern, we put the smallest sub-cut at the right end, the next smallest at the left end and continue to get the largest sub-cut at the center of the pattern. An example can be seen in figure 3.5. This procedure ensures that the larger (generally most valuable) cuts are at the center of the log,

| $X_{1,2}$ | $W_1$ | $X_{2,2}$ | $W_2$ | $W_2^m$ |
|-----------|-------|-----------|-------|---------|
| 1         | 2     | 0         | 4     | 2       |
| 2         | 2     | 0         | 4     | 4       |
| 3         | 2     | 0         | 4     | 6       |
| 4         | 2     | 0         | 4     | 8       |
| 0         | 2     | 1         | 4     | 4       |
| 1         | 2     | 1         | 4     | 6       |
| 2         | 2     | 1         | 4     | 8       |
| 0         | 2     | 2         | 4     | 8       |

Table 3.1: Values for  $W_2^m$

therefore trying to avoid any waste that may reduce their value. Note that we are basically assuming a softwood sawmill. In a hardwood sawmill the heartwood is typically not the most valuable wood and, as a result, sawing patterns are not usually based on a cant. For each given main cut, we then calculate its radius based on its thickness and total width (considering its allowances and kerfs) as shown in figure 3.6. This radius represents the circle in which a pattern can be located.

$$R = \sqrt{\left(\frac{T_j^m}{2}\right)^2 + \left(\frac{W_j^m}{2}\right)^2} \quad (3.4)$$

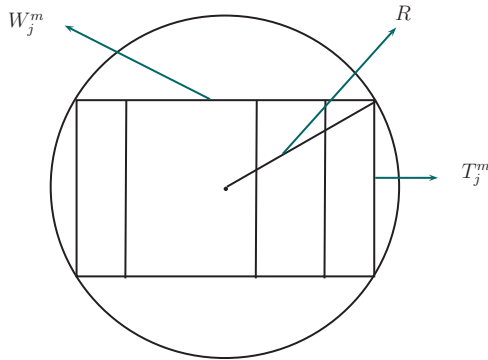
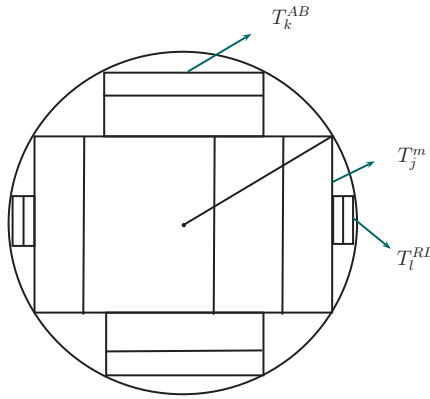


Figure 3.6: Main cut

### 3.1.1.2 Generating All Possible Above-below and Right-left Cuts

Based on the remaining area between the main cut and assumed pattern circle, it is possible to calculate above-below and right-left cuts by the same procedure described for the main cut. The only difference is in calculating maximum allowable thickness and width. We consider both vertical and horizontal sub-cuts for above-below and right-left cuts. As there are many numbers of different combinations for them, only the best patterns are chosen. These are the “n” patterns for each given main cut that have the highest area yields. By doing so, many undesirable patterns are eliminated from further calculations; we use the most of the remaining area and therefore most of the log volume. In figure 3.7, a typical pattern is shown where  $T_j^m$ ,  $T_k^{AB}$  and  $T_l^{RL}$  are the thicknesses for main, above-below and right-left cuts, respectively. To keep consistency in formulations we assume that sub-cuts are cut along the width parallel to the thickness of a rectangle (cant).



**Figure 3.7:** A pattern with above-below and right-left cuts

**Notations:**

|            |  |
|------------|--|
| $T^{AB}$   | thickness set for above-below cuts   |
| $T^{RL}$   | thickness set for right-left cuts  |
| $T_k^{AB}$ | thickness of the above-below cut, when the $k^{th}$ element of $T^{AB}$ is selected      |
| $T_l^{RL}$ | thickness of the right-left cut, when the $l^{th}$ element of $T^{RL}$ is selected       |
| $W_k^{AB}$ | width of above-below cut when thickness is $T_k^{AB}$                                    |
| $W_l^{RL}$ | width of right-left cut when thickness is $T_l^{RL}$                                     |
| $Y_{i,k}$  | number of sub-cuts with width $W_i$ and thickness $T_k^{AB}$ in the above-below cut      |
| $Z_{i,l}$  | number of sub-cuts with width of $W_i$ and thickness of $T_l^{RL}$ in the right-left cut |

Knowing  $T_j^m$ ,  $W_j^m$  and  $R$  from the previous section, we then find the potential above-below and right-left cuts. As it is possible to have both vertical and horizontal sub-cuts for these cuts, we consider four categories including:

1. Vertical above-below sub-cuts
2. Horizontal above-below sub-cuts
3. Vertical right-left sub-cuts
4. Horizontal right-left sub-cuts

Due to the similarities in calculations among all these classifications, Table 3.2 shows only the details of categories 1 and 2. The rest are presented in Appendix A.

The idea underlying calculations of all categories is similar. In the first stage, based on each pattern's radius, we find the maximum allowable space for above-below cuts (formulas 3.5 and 3.6). In the second step all the combinations of possible widths are generated by considering sawing kerf (formulas 3.7 and 3.8). Meanwhile, the final width of each cut should be fitted in the allowable remaining area (formulas 3.9 and 3.10).

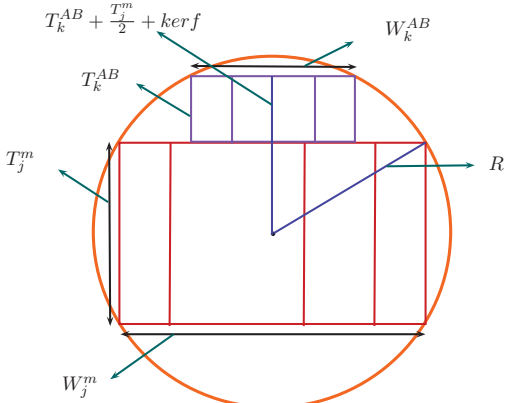
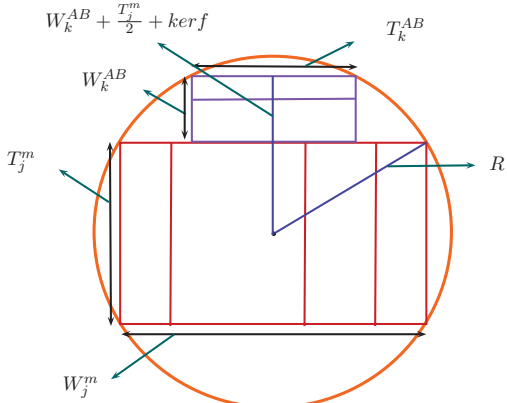
| Vertical above-below sub-cuts   | Horizontal above-below sub-cuts  |
|---|--|
| For each $T_k^{AB}$ where:  | For each $T_k^{AB}$ where:   |
| $T_k^{AB} \leq R - \left(\frac{T_j^m}{2} + kerf\right) \quad (3.5)$                           | $T_k^{AB} \leq W_j^m \quad (3.6)$  |
| We calculate all the combinations of possible widths and find $W_k^{AB}$ .                    | We calculate all the combinations of possible widths and find $W_k^{AB}$ .                           |
| $W_k^{AB} = \left(\sum_i Y_{i,k} * (W_i + kerf)\right) - kerf \quad (3.7)$                    | $W_k^{AB} = \left(\sum_i Y_{i,k} * (W_i + kerf)\right) - kerf \quad (3.8)$                           |
| Where:  | Where:   |
| $W_k^{AB} \leq 2 * \sqrt{R^2 - \left(T_k^{AB} + \frac{T_j^m}{2} + kerf\right)^2} \quad (3.9)$ | $W_k^{AB} \leq \sqrt{R^2 - \left(\frac{T_k^{AB}}{2}\right)^2} - \frac{T_j^m}{2} - kerf \quad (3.10)$ |
|             |                   |

Table 3.2: Formulas for above-below cuts

### 3.1.1.3 Finding “n” Best Combinations of Above-below and Right-left Cuts

For each given  $T_j^m$ ,  $T_k^{AB}$  and  $T_l^{RL}$  the area percent yield of the pattern is calculated according to formula 3.11. By area percent yield, we mean the total area of the target sizes of all the pieces of lumber making up the pattern divided by the area of the circle of radius R. The terms of the numerator represent the area of main, above-below and right-left cuts respectively. However, both above-below and right-left areas should be multiplied by 2, because of the symmetry assump-

tion. The denominator stands for area of the log.

$$\begin{aligned}
\text{Area Percent Yield} = & \text{Main cut area} + 2 * \text{above below area} + 2 * \text{right left area} = \\
& [(W_j^m - ((\sum_i X_{i,j}) - 1) * \text{kerf}) * T_j^m + \\
& 2 * (W_k^{AB} - ((\sum_i Y_{i,k}) - 1) * \text{kerf}) * T_k^{AB} + \\
& 2 * (W_l^{RL} - ((\sum_i Z_{i,l}) - 1) * \text{kerf}) * T_l^{RL}] / (\pi * R^2) * 100
\end{aligned} \tag{3.11}$$

This area percent yield is calculated in terms of target dimensions versus the area defined by the circular radius of the pattern.

For each main cut, we limit the number of total patterns generated so that only the best “n” patterns based on their area percent yields are chosen. Note that “n” is a user defined variable and its value can be set in the Python code (Appendix K). A discussion on how to choose “n” value in our example is provided in Section 3.2.1.

#### 3.1.1.4 Creating A Sorted Database of Final Patterns

All generated patterns are exported into a file that can be used as a database for future calculations. The information about each cutting pattern includes the pattern number, its radius, all the thicknesses and widths of sub-cuts and the orientations of above-below and right-left sub-cuts in terms of horizontal or vertical. With this information, a specific cutting pattern is determined that will be used in the log break down process. We export the information of each one into a list as follows:

$$\begin{aligned}
& [R, T_j^m, [X_{1,j}, X_{2,j}, X_{3,j}, \dots], T_k^{AB}, [Y_{1,k}, Y_{2,k}, Y_{3,k}, \dots], T_l^{RL}, [Z_{1,l}, Z_{2,l}, Z_{3,l}, \dots], \\
& H \text{ or } V \text{ or } N \text{ AB}, H \text{ or } V \text{ or } N \text{ RL}, [\text{Pattern\#}]]
\end{aligned}$$

Where:

|                                      |  |
|--------------------------------------|--|
| $R$                                  | Pattern radius   |
| $T_j^m$                              | Thickness of the main cut  |
| $[X_{1,j}, X_{2,j}, X_{3,j}, \dots]$ | A list for number of each sub-cut in the main cant with thickness $T_j^m$ (in ascending order of widths $W_1, W_2, W_3, \dots$ )           |
| $T_k^{AB}$                           | Thickness of the above-below cuts  |
| $[Y_{1,k}, Y_{2,k}, Y_{3,k}, \dots]$ | A list for number of each sub-cut in the above-below cuts with thickness $T_k^{AB}$ (in ascending order of widths $W_1, W_2, W_3, \dots$ ) |
| $T_l^{RL}$                           | Thickness of the right-left cuts   |
| $[Z_{1,l}, Z_{2,l}, Z_{3,l}, \dots]$ | A list for number of each sub-cut in the right-left cuts with thickness $T_l^{RL}$ (in ascending order of widths $W_1, W_2, W_3, \dots$ )  |
| $H$                                  | Horizontal orientation of the above-below or right-left cuts   |
| $V$                                  | Vertical orientation of the above-below or right-left cuts   |
| $N$                                  | No above-below or right-left cuts  |
| $AB$                                 | Above-below cuts   |
| $RL$                                 | Right-left cuts  |

For example, if  $W = \{0.866, 1.66, 3.75, 5.875\}$ , list  $[8.422, 11.875, [3, 5, 0, 0], 3.75, [2, 0, 0, 0], 2.75, [2, 0, 0, 0], H UD, V RL, [P 5087]]$  demonstrates that pattern number 5087 has the radius of 8.422(in). The cant (main cut) thickness is 11.875(in) and this includes 3 sub-cuts with width 0.866(in) and 5 sub-cuts with width 1.66(in). Each of the above-below edges consists of two sub-cuts with thickness 3.75(in) and width 0.866(in) which are horizontally cut. In each of right-left edges there are two vertical sub-cuts with thickness 2.75(in) and width 0.866(in). We should re-emphasize that all thicknesses and widths are reported as target values.

Eventually, the patterns are sorted according to their radii. This will simplify the process of selecting eligible patterns for each log from the database.



### 3.1.2 Generating Logs

In this section we discuss how logs are simulated using the characteristics of the class to feed the algorithm. They can be categorized based on various characteristics such as their geometry, species or other specifications. Since we assumed logs as truncated, well-shaped cones without any defects, we specifically defined them based on three basic factors; i) small-end radius, ii) taper and iii) length. Large radius (of large end) is calculated based on the three previous factors (Equation 3.12). In our approach we generate log classes based on the following notation:

|           |   |
|-----------|---|
| $Class$   | set including classes of logs $\{1, \dots, N\}$ . Each class involves the domains of the small end radius, taper and length random variables and the distribution functions on these domains. |
| $q$       | index for the elements of $Class$   |
| $R_q^s$   | small end radius of the log in the $q^{th}$ class   |
| $R_q^l$   | large end radius of the log in the $q^{th}$ class   |
| $L_q$     | length of the log in the $q^{th}$ class   |
| $Taper_q$ | taper of the log in length unit in the $q^{th}$ class   |

$$R_q^l = R_q^s + Taper_q * L_q \quad (3.12)$$

Due to the random nature of log sizes in each class, distribution functions are required to generate logs. If enough data are available, a distribution function can be constructed directly from the actual data. This would be the case if the mill had a reasonably large amount of scanner data available. For the classes without enough data, assumed distribution functions are used. We had only one day's scanner data available and that we will discuss how it was used in Section 3.2.1. In this case,  $R_q^s$  and  $L_q$  are generated from distribution functions.  $R_q^l$  is then calculated as a function of these two elements considering the logs' taper. Also  $Taper_q$  is assumed to be a random variable within a certain range that along with the smaller radius of the log determines the log's larger radius. Various log classes

can be created according to different policies that a sawmill use for log sorting process. For instance, they can be generated based on any of small end, large end, length, taper, species or different combinations of these factors.

### 3.1.3 Creating Price Lists

If a sawmill intends to get the maximum value in terms of lumber yields, it attempts to choose patterns that produce lumber with highest values. The price list is a driving factor that cutting pattern optimizers consider to distinguish among all possible choices of patterns and select the most valuable one. Since cutting pattern selection is sensitive to the price list, it can be used as a tool to control the outputs of the log breaking down process. Management can influence the output of certain log classes by deliberately setting the appropriate price list.

We use two types of price lists: The first is based on actual market prices at a certain point in time (Example: Appendix B) (ACE. [1]). This is the viewpoint that prices are real indicators of lumber value. If lumber can be sold into an infinite market with no requirements to meet any particular demand pattern, then optimizing patterns against these price lists makes sense. The other viewpoint is that prices are just signals to the optimizer of what to produce. In this viewpoint, a variety of price sets gives a variety ways of influencing what the optimizer produces. We can use price lists based on the dimensions of the lumber and a fixed unit price (Example: equation 3.13). Optimizers that use such a price list will maximize volume yields. However, as we will see below, other price lists can be created that emphasize certain dimensions.

$$Lumber\ value(\$) = Lumber\ volume(ft^3) * unit\ price(\$/ft^3) \quad (3.13)$$

Even with market based prices, since the exact data are not always available for all lumber pieces, it is necessary to find the relationship between lumber values (i.e. relative prices). We have used only a set of “actual” market prices and this

set was not complete for all the products. In Section 3.2.1 we demonstrate fitting a price function to these values. This could be repeated for several sets of market prices.

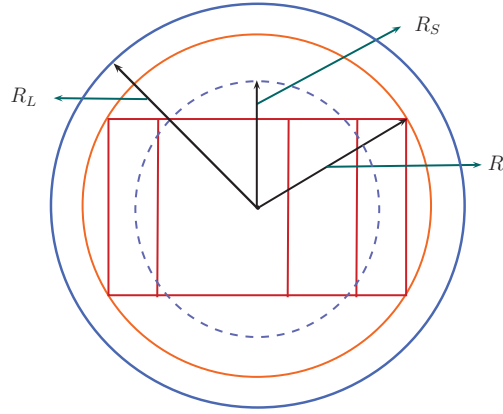
However, there are other possible methods to create additional price lists. One approach, as we will discuss in chapter 4, is that, given an initial set of campaigns, the linear programs will produce shadow prices on each product. These shadow prices can then be used to generate additional campaigns. This idea is similar to that in Maness and Norton [31].

### 3.1.4 Creating Lumber Output Fractions

After generating all patterns, classes of logs and price lists, the next step is implementing patterns on logs to create final products (lumber). As shown in figure 1 this process is done through the following stages: 3.1.4.1) filtering eligible patterns for each log, 3.1.4.2) calculating the length of each sub-cut of the given pattern, 3.1.4.3) finding the most valuable pattern for each log in each price list and 3.1.4.4) creating campaigns.

#### 3.1.4.1 Filtering Eligible Patterns for Each Log

To reduce computational effort, it is useful, as a first step, to filter patterns under consideration, based on the radius of each pattern and two end radii of each log. We reject patterns for which the pattern radius is not within the range of two end radii of each log (e.g. eligible  $R$  must satisfy  $R_S \leq R \leq R_L$  as shown in figure 3.8). This filtering is reasonable since patterns with  $R < R_S$  will result in wasted wood volume, as there might be a larger pattern which can completely fit to the log and produce higher volume yield. On the other side, the patterns with  $R_L > R$  will cause excessive waness and usually loss in the sub-cuts of the sides and even in the main cut.



**Figure 3.8:** Eligible pattern

### 3.1.4.2 Calculating the Length of Each Sub-cut of the Given Pattern

Because the logs are tapered, often not all pieces of lumber produced from the log are the length of the log. In this stage by using geometric calculations we find the touching point along the length where each sub-cut touches the edge of log. This provides the information about the final length for each lumber piece when a pattern is used on the log, based on a reasonable estimation of the allowable wane.

The formulations presented in this section find the length of each sub-cut in the main cut. As described earlier, the smallest sub-cut is located at the right end of the main cut, the next smallest is at the left end. This process of locating sub-cuts in ascending order of their widths continues till the largest sub-cut is the last one located presumably in the middle of the main cut.

The procedure begins with the smallest (right-end) sub-cut and calculates its length (touching point) according to the allowable wane. Then length of the next smallest sub-cut at the left end is computed. This procedure continues till either we reach a sub-cut with the same length as the log or the last sub-cut.

The following are the formulations to find the length of the main cut when considering wane.

**Notations:**

|               |  |
|---------------|--|
| $rWaneUD$     | radius of sub-cuts by accepting the maximum wane from up and down  |
| $rWaneSide$   | radius of the sub-cuts by accepting the maximum wane from sides  |
| $rWane$       | maximum of the $rWaneUD$ and $rWaneSide$   |
| $wanePrUD$    | maximum allowable wane percentage from up and down   |
| $wanePrSide$  | maximum allowable wane percentage from sides   |
| $Lwane$       | length of the sub-cut by considering acceptance of the maximum wane  |
| $Length$      | set of lumber lengths  |
| $W_R$         | right half of the total width of the main cut  |
| $W_L$         | left half of the total width of the main cut   |
| $B, B_R, B_L$ | binary variables for switching between $W_R$ and $W_L$   |
| $L_k$         | length of the lumber when the $k^{th}$ length is chosen from $Length$ set  |
| $X_{ijk}$     | number of lumber pieces in the main cut with the width $W_i$ , thickness $T_j^m$ and length $L_k$  |
| $Y_{ijk}$     | number of lumber pieces in each of the above-below cut with the width $W_i$ , thickness $T_j^{AB}$ and length $L_k$  |
| $Z_{ijk}$     | number of lumber pieces in each of the right-left cut with the width $W_i$ , thickness $T_j^{RL}$ and length $L_k$   |
| $Seq_j^m$     | Set of sub-cuts widths of the main cut with $X_{i,j}$ number of $W_i$ for each $T_j^m$<br><br>For example: the main cut includes two sub-cuts with width 0.866(in) and one sub-cut with width 3.75(in) so $Seq_j^m = \{0.866, 0.866, 3.75\}$ |
| $W_{min}$     | smallest value over the members of $Seq_j^m$   |

We use the following algorithm and formulations for the main cut:

For each thickness  $T_j^m$ :

set  $W_R = W_L = \frac{W_j^m}{2}$ ,  $B_R = 1$ ,  $B_L = 0$  (starting from the right half),  $X_{ijk} = 0$

**Step 1:** Find  $W_{min} = Min[Seq_j^m]$

**Step 2:** Keep  $i$  if  $W_{min} = W_i$

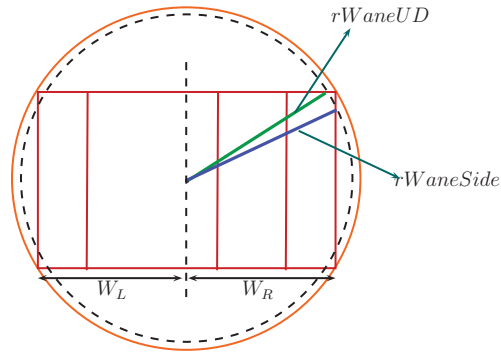
**Step 3:** Calculate  $rWaneUD$ ,  $rWaneSide$  and  $rWane$

$$rWaneSide = \sqrt{((W_R * B_R) + (W_L * B_L))^2 + \left(\frac{1 - wanePrSide}{2} * T_j^m\right)^2} \quad (3.14)$$

$$rWaneUD = \sqrt{((W_R * B_R) + (W_L * B_L) - (wanePrUD * W_{min}))^2 + \left(\frac{T_j^m}{2}\right)^2} \quad (3.15)$$

As calculated above there can be two radii for waness ( $rWaneSide$ ,  $rWaneUD$ ) in the pattern. When one of them touches the edge of log we should stop, because further than that the resulting cut will get more than allowable wane. So the maximum of  $rWaneUD$  and  $rWaneSide$  is the determining factor to find the lumber length. (Equation 3.16)

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (3.16)$$

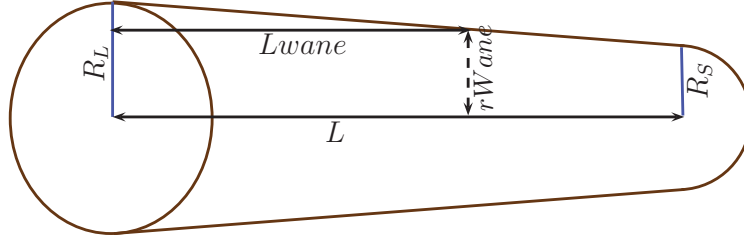


**Figure 3.9:**  $rWaneUD$  and  $rWaneSide$

**Step 4:** Calculate  $L_{wane}$ :

$$L_{wane} = \lfloor Min\{L, L * (1 - \frac{rWane - R_S}{R_L - R_S})\} / 2 \rfloor * 2 \quad (3.17)$$

This calculation, with rounding down, assumes that allowable lumber lengths are 8, 10, 12, 14, 16 (ft). Figure 3.10 demonstrates the touching point position.



**Figure 3.10:** Touching point and lumber length

**Step 5:** If  $L_{wane} = L$ , set  $L$  as the length of remaining cuts ( $X_{ijk} = X_{ijk} + X_{i,j}$  where  $L_k = L$ ) and STOP.

Otherwise go to step 6.

**Step 6:** If  $L_{wane} = L_k$ , set  $X_{ijk} = X_{ijk} + 1$  and  $X_{i,j} = X_{i,j} - 1$

**Step 7:** Let  $W_R = W_R - (W_{min} + kerf) * B_R$  and  $W_L = W_L - (W_{min} + kerf) * B_L$

**Step 8:** Update the set of width,  $[Seq_j^m] = [Seq_j^m] - [W_{min}]$  and let  $B = B_R, B_R = B_L, B_L = B$  (switch between  $W_R$  and  $W_L$ )

**Step 9:** If  $[Seq_j^m] \neq \emptyset$  go to step 1.

Otherwise STOP.

By applying this algorithm we will find the length of each sub-cut in the main cut and all  $X_{ijk}$ s. We use a similar approach for above-below and right-left cuts to find  $Y_{ijk}$  and  $Z_{ijk}$  (Appendices C, D, E, F). There are just minor differences in calculating  $rWaneSide$  and  $rWaneUD$  for these cuts.

### 3.1.4.3 Finding the Most Valuable Eligible Pattern for Each Log Based on Each Price List

After calculating the outputs of all eligible patterns on each log, including lumber dimensions and number of each piece, we are able to compute the value of lumber pieces produced from each pattern on a specific log based on each price list. Among all those patterns, the most valuable one is chosen for each price list. Formula

3.18 shows the value of the best pattern for all given price lists.

$$MaxValue^u = Max_{all\ patterns} \left\{ \sum_i \sum_j \sum_k P_{ijk}^u (X_{ijk} + 2 * (Y_{ijk} + Z_{ijk})) \right\} \text{for all } u \quad (3.18)$$

Where:

|               |   |
|---------------|---|
| $rMaxValue^u$ | The maximum over the values of all eligible patterns on each log based on price list u                              |
| $P_{ijk}^u$   | Price of lumber with width $W_i$ , thickness $T_j$ and length $L_k$ for price list u                                |
| $X_{ijk}$     | number of lumber pieces in the main cut with the width $W_i$ , thickness $T_j^m$ and length $L_k$                   |
| $Y_{ijk}$     | number of lumber pieces in each of the above-below cut with the width $W_i$ , thickness $T_j^{AB}$ and length $L_k$ |
| $Z_{ijk}$     | number of lumber pieces in each of the right-left cut with the width $W_i$ , thickness $T_j^{RL}$ and length $L_k$  |

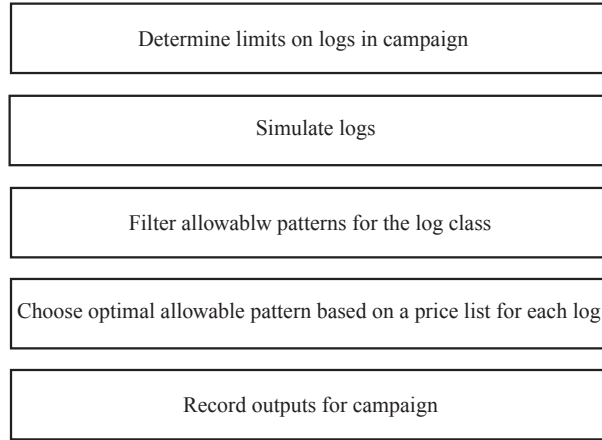
#### 3.1.4.4 Creating Campaigns

For each log in a specific log class, the most valuable cutting pattern is chosen based on a given price list. We repeat this process for all the logs within the log class. So, the number of optimal cutting patterns are equal to the number of logs in the class. The resulting outputs of all logs in terms of lumber yields are obtained and accumulated. A combination of a log class under a certain price list resulting in a set of specific outputs, is considered as a “campaign”.

The goal is to determine, for a particular campaign, the expected outputs of lumber. These include the number of each lumber piece (for each log and log class), total lumber value (for each log and log class) and volume percent yield (for each log class) (formula 3.19). This process enables us to estimate the expected outputs of different log classes by applying various price lists and consequently create diverse campaigns. In summary, campaigns can be generated through the steps shown in figure 3.11.



$$\text{Volume percent yield} = \frac{\text{Total lumber volume}}{\text{Total volume of logs}} * 100 \quad (3.19)$$



**Figure 3.11:** Algorithm for generating campaigns

However, different log sorting decisions and additional price lists (e.g. the ones based on shadow prices) would create more potential campaigns.

In the following section, different proposed campaigns resulting from applying this algorithm are presented.

## 3.2 Example

To illustrate the performance of the proposed algorithm, various campaigns are generated based on seven log classes and twenty price lists. The details of each are presented in the following sections. Several comparisons are developed to evaluate the effectiveness of the algorithm. In the first case, different log classes are examined using the same price list, while the second case shows the effectiveness of different price lists to create various products of the same log classes. All log classes are assumed to be chosen from one species of tree.

### 3.2.1 Data and Procedures

According to the previous notation, we used the following assumed data.

#### Sawing process and lumber specifications

There are 70 products presented in nominal lumber dimensions ( $W^{nom}(in)$  x  $T^{nom}(in)$  x  $L^{nom}(ft)$ ) as shown in Table 3.3.

|        |         |         |         |         |        |         |         |         |         |
|--------|---------|---------|---------|---------|--------|---------|---------|---------|---------|
| 1x3x8  | 1x3x10  | 1x3x12  | 1x3x14  | 1x3x16  | 2x3x8  | 2x3x10  | 2x3x12  | 2x3x14  | 2x3x16  |
| 1x4x8  | 1x4x10  | 1x4x12  | 1x4x14  | 1x4x16  | 2x4x8  | 2x4x10  | 2x4x12  | 2x4x14  | 2x4x16  |
| 1x6x8  | 1x6x10  | 1x6x12  | 1x6x14  | 1x6x16  | 2x6x8  | 2x6x10  | 2x6x12  | 2x6x14  | 2x6x16  |
| 1x8x8  | 1x8x10  | 1x8x12  | 1x8x14  | 1x8x16  | 2x8x8  | 2x8x10  | 2x8x12  | 2x8x14  | 2x8x16  |
| 1x10x8 | 1x10x10 | 1x10x12 | 1x10x14 | 1x10x16 | 2x10x8 | 2x10x10 | 2x10x12 | 2x10x14 | 2x10x16 |
| 1x12x8 | 1x12x10 | 1x12x12 | 1x12x14 | 1x12x16 | 2x12x8 | 2x12x10 | 2x12x12 | 2x12x14 | 2x12x16 |
| 4x4x8  | 4x4x10  | 4x4x12  | 4x4x14  | 4x4x16  | 6x6x8  | 6x6x10  | 6x6x12  | 6x6x14  | 6x6x16  |

Table 3.3: Nominal lumber dimensions

Nominal, target and actual data for lumber dimensions according to Harry Freeman and Son Ltd reports (Harry Freeman and Son Ltd. [23]) are shown in Table 3.4.

| Nominal ( <i>in</i> ) | Target ( <i>in</i> ) | Actual ( <i>in</i> ) |
|-----------------------|----------------------|----------------------|
| 1                     | 0.866                | 0.75                 |
| 2                     | 1.66                 | 1.5                  |
| 3                     | 2.75                 | 2.5                  |
| 4                     | 3.75                 | 3.5                  |
| 6                     | 5.875                | 5.5                  |
| 8                     | 7.875                | 7.25                 |
| 10                    | 9.875                | 9.25                 |
| 12                    | 11.875               | 11.25                |

Table 3.4: Nominal, target and actual data

Thickness, width, length and  $wRatioMax$  of allowable lumber are shown respectively as:

$$T^m = T^{AB} = T^{RL} = \{2.75, 3.75, 5.875, 7.875, 9.875, 11.875\}(in)$$

$$W = \{0.866, 1.66, 3.75, 5.875\}(in)$$

$$Length = \{8, 10, 12, 14, 16\}(ft)$$

$$wRatioMax = \{2, 2, 2, 2, 1.5, 1.2\}$$

From economic and technological points of view, most sawmills do not intend to re-saw their in-process products (above-below and right-left sub-cuts). Therefore, the intention to get the maximum yield from the main cut leads to generating a somewhat square-shaped main cuts which decreases the chance of re-sawing necessitated by multiple above-below and right-left sub-cuts.

$wRatioMax$  provides a tool for controlling shape of the main cut. By adjusting various values of this parameter, different outputs are obtained. In this specific problem we assumed  $wRatioMax = 2$  for smaller thicknesses (2.75, 3.75, 5.875, 7.875), and for larger ones, it is assumed such that the resulting cant can be fit in the largest possible log. Greater values for  $wRatioMax$  will result in a cant which cannot be fit into even the largest log.

It is assumed that the saw kerf is 0.15(in). Also the number of best combinations of above-below and right-left cuts for each main cant used in the filtering process is 20, although this is a user defined parameter which can be changed.

### Log Classes:

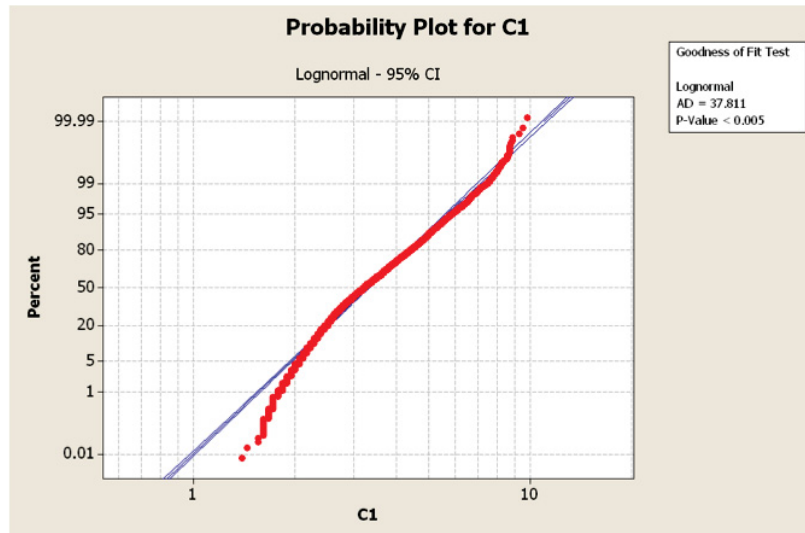
Because we did not have much access to real data on the distribution of log dimensions, we created various log classes through simulation. In actual application, it should be possible to build up log distributions through scanner information. If enough scanned data is available it can be used directly. Otherwise, a fitting process, similar to that described below for the large log class, can be used. We now describe how we generated seven log classes based on logs' dimensions (small end radius and length).

The first log class is called the small log class. Due to the lack of data for this class, two distribution functions are assumed for the small end radius (with average 2.5(in)) and length as follows:

$$R_1^s \sim \text{Uniform}[2, 3]$$

$$L_1 = \begin{cases} P(8 \leq L_1 < 10(ft)) = 0.4 \\ P(10 \leq L_1 < 12(ft)) = 0.4 \\ P(12 \leq L_1 < 14(ft)) = 0.2 \end{cases}$$

The second class is called large class. For this log class, we had access to the data of a 9-hour work shift in which 9613 logs were sawn at Bowater Mersey Oakhill sawmill (Sawmill [42]). Since we assumed the average radius of 3.5(*in*) for the large log class, we scaled all the data to obtain this average. As the available data were not enough for our experiment, we required to find their fitted distribution functions. For the small end radius, the lognormal distribution function was a satisfactory fit. Using the Minitab software, the suggested distribution for the small end radius of the logs is Lognormal with the mean of 1.198 and standard deviation of 0.323. The results of fitting the lognormal distribution function to log class 2 data by using Minitab are shown in Figure 3.12. As can be seen from the probability plot, the fit is reasonable over the mid range of the radii.



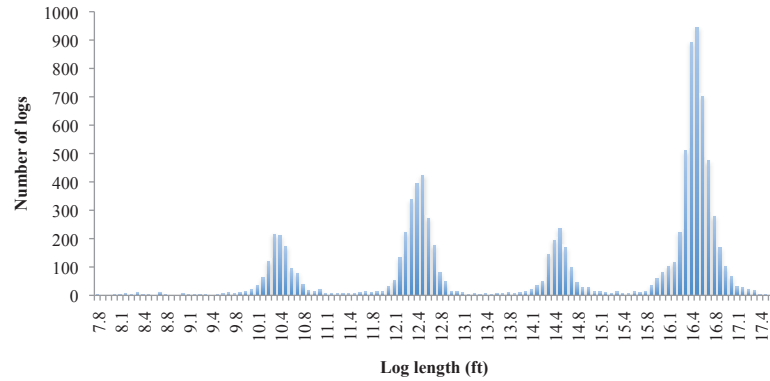
**Figure 3.12:** Minitab probability plot for small end radius

$$R_2^s \sim \text{lognormal}(1.198, 0.323)$$

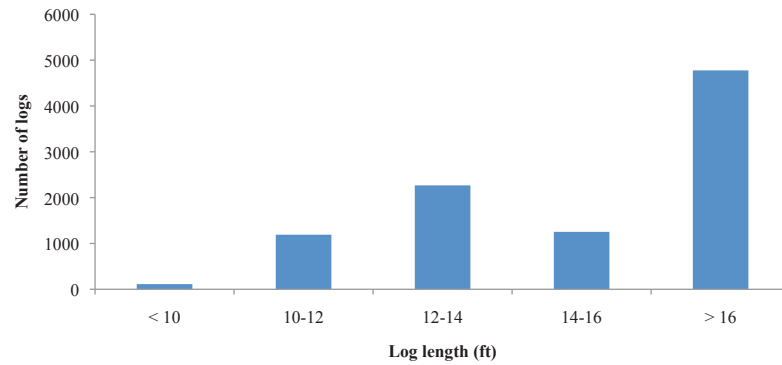
Log lengths are classified in equal uniform intervals (with length  $2(ft)$ ) and

the number of logs within each interval was counted to estimate the probability of each interval.

The real data of logs length for log class 2 and our estimation by fitting the uniform distribution function within each interval are respectively shown in Figures 3.13 and 3.14.



**Figure 3.13:** Real data for log class 2



**Figure 3.14:** Fitted uniform distribution for log class 2

The following shows the mentioned distribution functions.

$$L_2 = \begin{cases} P(8 \leq L_2 < 10(ft)) = 0.012 \\ P(10 \leq L_2 < 12(ft)) = 0.124 \\ P(12 \leq L_2 < 14(ft)) = 0.237 \\ P(14 \leq L_2 < 16(ft)) = 0.131 \\ P(16 \leq L_2 < 18(ft)) = 0.496 \end{cases}$$

There is a relationship between the difference of small and large end diameters, and the length of a tree. As a tree grows, the difference between these two end diameters usually increases. Thus, we assume a log radius can change 0.05 to 0.2 inches per each foot of length. To reasonably estimate the larger end radius of a log, by considering its taper (between 5% and 20% of the log's length), we assume the relationship between smaller radius and larger radius for both classes as follows:

$$R_q^L = F(R_q^s, L_q)(in) = R_q^s + Uniform[0.05, 0.2](\frac{in}{ft}) * L_q(ft), q = 1, 2 \quad (3.20)$$

The number of logs within each class is required to be a large number to ensure the stability of the campaign outputs. We assumed 100,000 logs within each individual log class.

We combine small and large classes (200,000 logs) together and sort their logs according to specific log lengths into 5 categories. This creates 5 more log classes. The idea of sorting logs based on their lengths prior to the sawing process is often used in most sawmills. This will simplify and accelerate their sawing processes.

Since, the optimal outputs of each log were calculated once, we do not need to re-calculate them. Available logs are only sorted in specific lengths. Our classifications include the log length less than 10(ft) (creating log class 3 with 41255 logs), between 10(ft) and 12(ft) (creating log class 4 with 52394 logs), between 12(ft) and 14(ft) (creating log class 5 with 43769 logs), between 14(ft) and 16(ft) (creating log class 6 with 12900 logs) and greater than 16(ft) (creating log class 7 with 49682 logs).

Price lists:

Two categories of price lists are considered. The first price list is derived from Appendix B. Due to lack of data for some products such as lumber pieces with width 1(in), a function is fitted to the available ones. To give an estimate of relative prices for products without data. The fit function is computed as

described in equation 3.21.

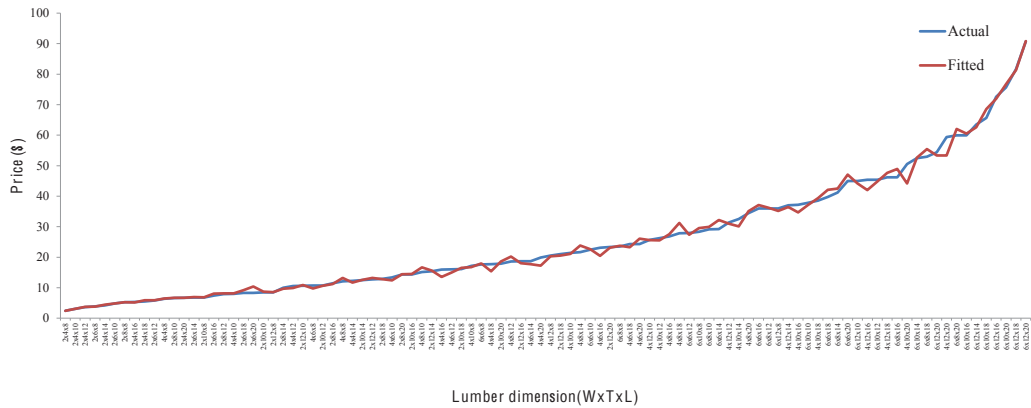
$$\begin{aligned}
 & \text{Fit function}(W^{nom}, T^{nom}, L^{nom}) = \\
 & a + b * W^{nom} + c * T^{nom} + d * L^{nom} + e * (W^{nom} * T^{nom}) + f * (W^{nom} * T^{nom} * L^{nom})
 \end{aligned}
 \tag{3.21}$$

where  $a, b, c, d, e$  and  $f$  are constant coefficients and  $W^{nom}, T^{nom}, L^{nom}$  represent nominal width, thickness and length of the lumber piece, respectively.

Figure 3.15 demonstrates the difference between actual and fitted prices for all the available patterns. We obtain the values of coefficients using the “Least Squares” method (“LINEST” function in Excel). The fit function is then used to calculate the prices for our assumed products. The appropriate fit function is as equation 3.22.

$$\begin{aligned}
 & \text{Fit function}(W^{nom}, T^{nom}, L^{nom}) = \\
 & 2.498 + 1.047 * W^{nom} + 0.198 * T^{nom} + 0.049 * L^{nom} - 0.0357 * (W^{nom} * T^{nom}) \\
 & - 0.0000278 * (W^{nom} * T^{nom} * L^{nom})
 \end{aligned}
 \tag{3.22}$$

The MAPE (Mean Absolute Percentage Error) of this function is 4.15% which illustrates the reasonable accurate performance of the fit function.



- Price list 2: Since the unit price is the same for all products, this price list emphasizes obtaining the most lumber volume out of the log (equation 3.23).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}(ft) * L^{nom}(ft) * 1\left(\frac{\$}{ft^3}\right) \quad (3.23)$$

- Price list 3: This price list aims to focus on lumber with greater widths (equation 3.24).

$$Lumber\ value(\$) = W^{nom}\sqrt{W^{nom}}(ft)^{1.5} * T^{nom}(ft) * L^{nom}(ft) * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (3.24)$$

- Price list 4: This price list emphasizing lumber with greater thicknesses (equation 3.25).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}\sqrt{T^{nom}}(ft)^{1.5} * L^{nom}(ft) * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (3.25)$$

- Price list 5: This price list attempts to focus on lumber with greater lengths (equation 3.26).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}(ft) * L^{nom}\sqrt{L^{nom}}(ft)^{1.5} * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (3.26)$$

The remaining price lists (6-20) aim to obtain more lumber with a specific width, thickness or length by multiplying the unit price of these special products by a user defined positive coefficient (here it is assumed 20). Thus, price lists 6-9 emphasize lumber with specific widths. Since allowable lumber pieces are of four different widths, four price lists are generated. The same procedure is used for allowable thicknesses through price lists 10-15 and allowable lengths through price lists 16-20.

In summary, twenty price lists are created to serve as driving factors to create



126 various campaigns. By adding more price lists we get more diversified campaigns, at the expense of increased program running time. Although we choose 20 price lists in this research, our program is flexible enough to easily create new campaigns by taking new price lists.

### 3.2.2 Results

With 7 log classes and 20 price lists, we are able to create, at most, 140 campaigns. According to the specifications of some log classes, 14 of the campaigns are repeated (see Section 3.3). Therefore, the total number of campaigns ends up with 126. In this section, the results of several campaigns used in five comparison cases are provided to demonstrate the roles of price list and log class in diversifying campaigns outputs.

#### 3.2.2.1 Comparison Case 1 (Different Log Classes, Same Price List):

In this section two examples are presented to show how log class specifications result in different lumber outputs. First, small and large log classes are compared when price list 2 is applied. This price lists maximizes the lumber output volume. The second example corresponds to log classes 3 to 7 when price list 1 is used. Price list 1 is the actual market price list. Figure 3.16 shows the number of each lumber piece for two campaigns in the first example.

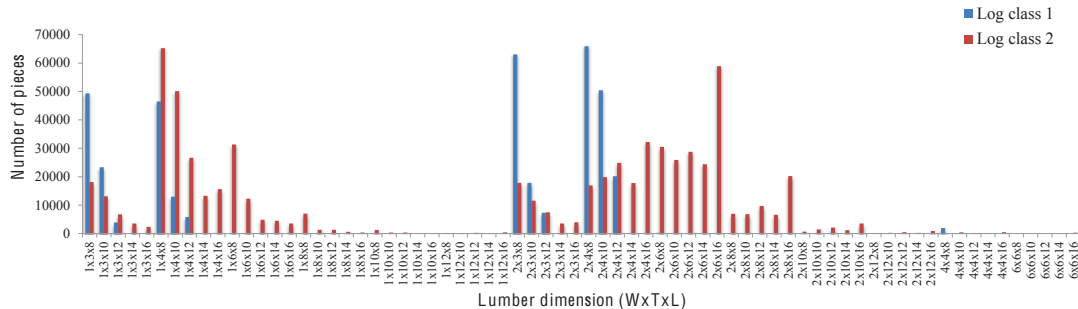


Figure 3.16: Price list 2, log classes 1-2

As can be seen in figure 3.16, the small log class never produces lumber pieces

with length 14(ft) or more. This was expected, as the distribution function for log lengths does not include logs with length 14(ft) or more. Also, most of the products have length 8(ft) due to the probability distribution of log lengths. Regarding the large log class, since the unit prices for all products are the same and there is no advantage for bigger pieces over smaller ones, the algorithm prefers to choose from smaller cuts because they create more flexibility in making various combinations. Consequently, larger cuts are rarely produced.

Table 3.5 compares actual, target and nominal volume percent yields of the two campaigns. Actual percent yield is the campaign yield when actual lumber dimensions are considered. Target and nominal percent yields correspond to the target and nominal lumber dimensions, respectively.

|                               | Actual (%) | Target (%) | Nominal (%) |
|-------------------------------|------------|------------|-------------|
| Small log class, Price list 2 | 36.26      | 43.77      | 56.04       |
| Large log class, Price list 2 | 44.25      | 53.00      | 65.55       |

Table 3.5: Volume percent yields in comparison case 1

The difference between volume percent yields of these two campaigns results from the limitations on the number of eligible cutting patterns for the small log class, due to their small radii. This causes more waste in comparison to the large class in which more patterns are applicable. Sawmills use different standards (actual, target or nominal) to report their percent yields.

Figure 3.17 demonstrates the difference in outputs of the campaigns in the second comparison. As log classes are created by sorting logs in specific lengths, it can be seen that Log class 3 (blue), in which logs are with the length less than 10(ft), produce only the products with length 8(ft). This behaviour is similar for the rest of classes. The only one that is capable of producing all lumber pieces is log class 7(orange) which contains the logs with the length more than 16(ft). The role of price list 1 in these campaigns is to emphasize larger lumber pieces, since the unit prices for these products are often more than smaller ones.

### 3.2.2.2 Comparison Case 2 (Different Price Lists, Same Log Class):

Three examples are provided to show how various price lists can result in different campaign outputs from the same log class. The first example deals with price lists 6-9 in which the main focus is on the products with specific widths implemented on log class 2. In the second example, price lists 10-13 are implemented on log class 1 to show campaign sensitivity to different lumber thicknesses. In the last example, the influence of applying price lists 16-20 on log class 2 is discussed. The objective is to demonstrate the effectiveness of those price lists with highlighting special lengths on creating lumber outputs. The number of lumber pieces in the first comparison is illustrated in figure 3.18. Referring to the figure, most products with width 1(in) are generated by applying price list 6 (blue). Lumber pieces with width 2(in) are mainly resulted from price list 7 (red). Although one might expect that this price list should have produced nothing but the products with width 2(in), some lumber pieces with width 1(in) can also be seen. The reason is that, in many cases, if there is no possibility to get all lumber pieces with 2(in) width, creating some lumber with smaller widths would be more beneficial than nothing. It is obvious no products with width 4(in) or 6(in) are produced since they can be easily transformed to sub-products with width 2(in). Price list 8 (green) and 9 (purple) mostly create products with width 4(in) and 6(in) respectively.

The second example is demonstrated via figure 3.19 in which the main focus is on different thicknesses for the small log class. Due to specifications of the logs lengths in this class we do not see products with length 14(ft) or more. Also lumber pieces with thickness 10(in) and 12(in) never can be obtained from these logs; therefore campaigns 14 and 15 are eliminated from consideration. Price lists 10 (blue), 11 (red), 12 (green) and 13 (purple) mainly produce products with thickness 3(in), 4(in), 6(in) and 8(in) respectively.

In the last case (figure 3.20), when price lists focus on specific lengths, we see most products with length 8(ft) are usually created by applying price list 16(blue).

Consequently, Price lists 17 (red), 18 (green), 19 (purple) and 20 (orange) mostly create lumber pieces with length 10(ft), 12(ft), 14(ft) and 16(ft), respectively.

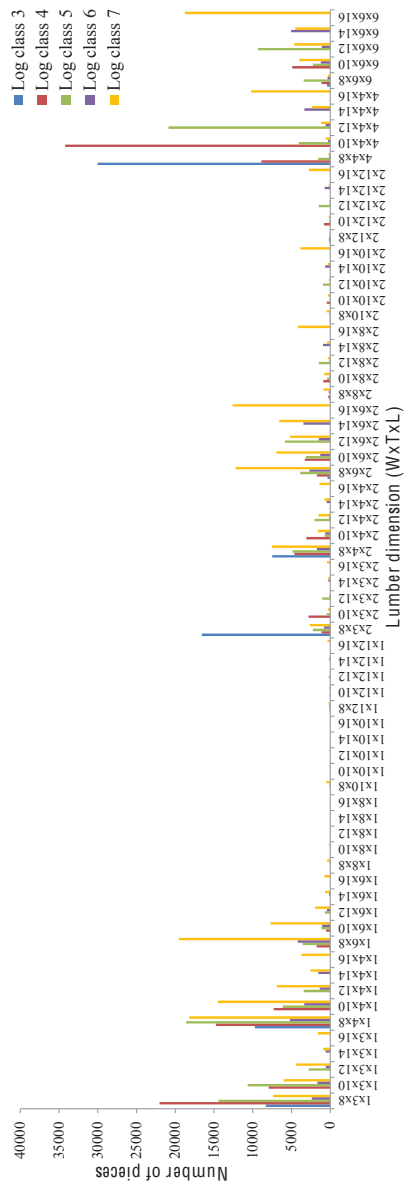


Figure 3.17: Price list 1, log classes 3-7

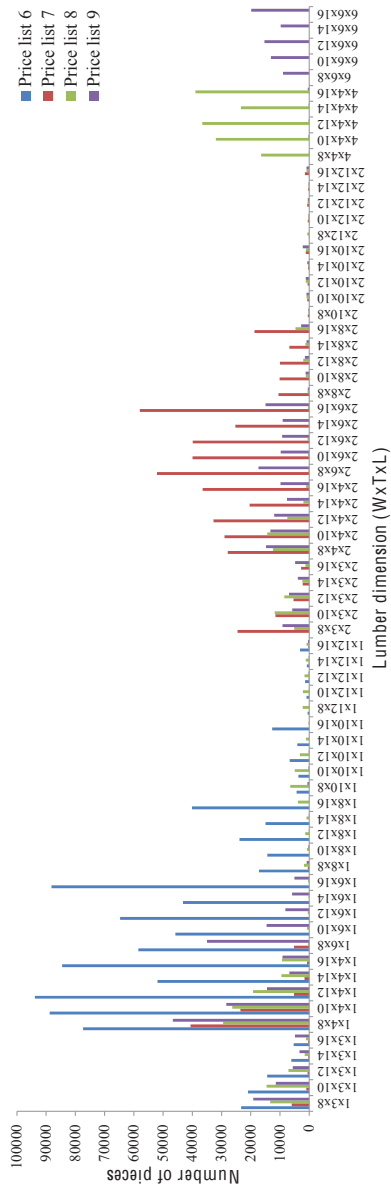


Figure 3.18: Log class 2, price lists 6-9

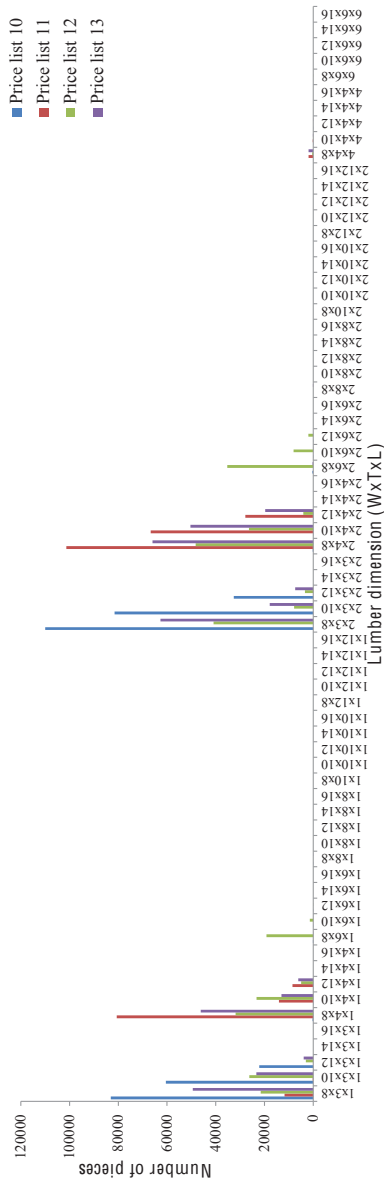


Figure 3.19: Log class 1, price lists 10-13

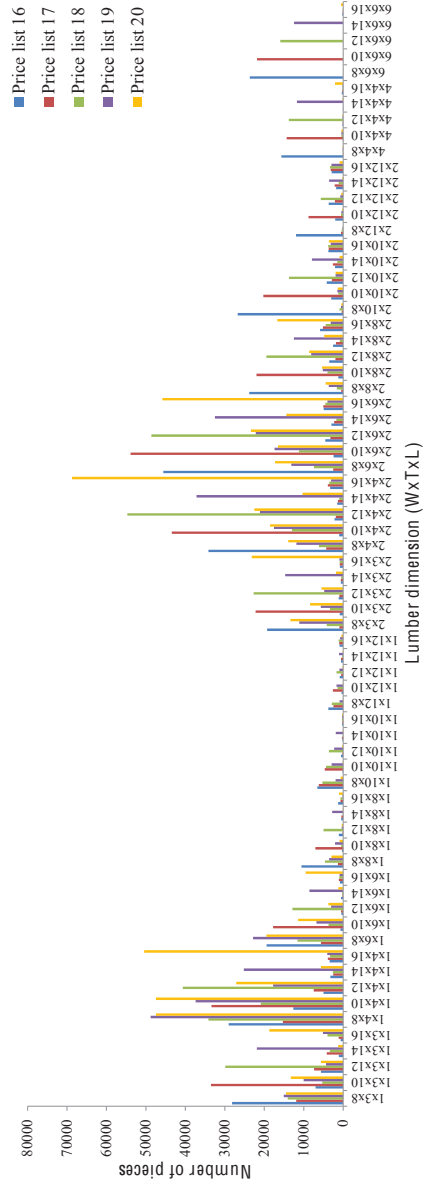


Figure 3.20: Log class 2, price lists 16-20

### 3.3 Discussion

The example discussed in this chapter demonstrated the features of the approach. Using the proposed algorithm, one can create other campaigns with other desirable specifications. In this example, logs are created from one species of tree, constituting 7 different log classes. By assuming 20 price lists, 140 campaigns are created from which 14 are eliminated. The reason is that they regenerate some existing campaigns. For instance, Price lists 14, 15, 19 and 20 in the small log class (class 1) create the same results as price list 2. Since the small log class (1) can not create some lumber pieces, these price lists, that emphasize large pieces, are not effective. For example, price list 19 emphasizes pieces with length 14(ft). These products can not be produced from the small log class, because, logs in this class are shorter than 14(ft). The products that can be produced, all are evaluated on volume. This leads to the same results for these campaigns. Also, few price lists that force the creation of lumber pieces with non-sensical lengths are discarded in log classes 3-7. For instance, log class 4 in which logs lengths are less than 12(ft), price lists 18, 19 and 20 emphasizing length 12(ft), 14(ft) and 16(ft) respectively, are meaningless.

One important factor to evaluate the performance of sawmills, is the proportion of the output (total lumber volume) to the input (total log volume). This is represented by campaign volume yield in this work. As there are more candidate patterns to be implemented on large log classes rather than small ones, the possibility of obtaining more output from large classes is higher. In all the cases the nominal percent yield of implementing price list 2 is greater than other price lists. The reason is that unit prices of all the products in price list 2 are the same; therefore, value maximization is equivalent to volume maximization.

We have developed a fast algorithm that provides the possibility to rapidly generate a number of alternative campaigns. These campaigns can be further used in planning and operational stages in sawmill operations. This provides

decision makers with the ability to exploit combinations of various campaigns for planning and scheduling to fulfill demand requirements.

In the next chapter, the results of this algorithm are used as inputs to an MILP problem to select and plan appropriate campaigns to fulfill deterministic-constant demands of various products.



# Chapter 4

## Powers-of-Two Model

The campaign generation procedure enables us to estimate output of different configurations of campaign settings including various log classes and price lists. Each product's expected volume yield from a unit volume of the log under a specific configuration is available. The question remains how to use these potential campaigns to produce required products within capacity requirements to minimize total inventory cost and setup time. This leads to the principles of the ELSP problems. However, a key difference is "campaign lot sizes" need to be scheduled instead of individual "lot sizes".

To provide a fast and efficient approach that can be easily solved, we developed a model based on the Extended Basic Period ("EBP") with the Powers-of-Two ("PoT") policy. This approach seeks optimal campaign cycle times based on some powers of two multipliers of a basic period, as well as campaign run durations. We assume the basic period is known and fixed based on the information from sawmill management. This makes the solution procedure simpler and closer to reality. The cycle time of each campaign is thus equal to  $2^k\tau_0$ , where  $k$  is the "campaign coverage" and can be a small integer (e.g. 0, 1, 2, 3...) and  $\tau_0$  is the basic period. This means for instance, if the coverage of a campaign is found to be 1 and the basic period is "one week", the associated cycle time for that campaign will be  $2^1 = 2$  weeks. In other words, this campaign is processed once every 2

weeks.

The output of this model provides a selection of the appropriate campaigns, the frequency with which they should be run and the resulting average cycle stock inventory of each product. We use the term “appropriate” due to this fact that not all the campaigns can be fit in the schedule. In other words, the algorithm seeks a combination of campaigns that result in total production and setup time less than available time; while the output meets the demand and results in keeping inventory levels to a minimum. “Cycle stock” inventory is the amount of on-hand inventory that is used for the regular demands and is replaced every cycle when a new production run occurs. The objective of this model is thus to provide a framework for organizing campaigns in a sawmill to fulfill annual constant demand of different products while keeping inventory levels at a minimum.

This chapter begins with a description of the PoT model’s specifications and assumptions. The model’s details are then presented through its mathematical formulation. Two examples are created to demonstrate the results and performance of this model.

## 4.1 Model Description

The main assumption underlying the model is that there is enough production capacity to meet annual product demand while combining different campaigns to produce the required products. As the model deals with campaigns, it is obvious that campaign setup times can restrict production capacity. The challenging questions are: which campaigns should be taken into account, how often (cycle time) and for how long (production time) should they be processed to minimize inventory costs. To keep inventory minimum while meeting capacity requirements, we use PoT modelling approach to determine the appropriate campaigns and their run frequency. Additionally, the following assumptions are used in the modelling approach:

1. The amounts of input logs required to form a campaign are always available.
2. Demands are constant and deterministic over a planning horizon (e.g., one year).
3. Setup times are sequence independent.
4. The sawmill can be thought of as a single machine facility.
5. No back-order cost is taken into account. All unfulfilled demands are considered as lost sales. The main focus is minimizing inventory and violations of demand requirements.
6. The basic period ( $\tau_0$ ) is known as a parameter in advance.

What is new about our modelling approach is that the powers of two are applied to campaigns instead of being applied to products. The ELSP and powers of two models discussed in chapter 2 are applied to products. In this model, the campaigns are processed based on the integer powers-of-two multipliers of a basic time period. This basic period can be a week ( $\frac{1}{52}$  of a year), a day ( $\frac{1}{365}$  of a year) or any other time period. Thus, cycle times might be 1, 2, 4, 8, 16, 32, ... multipliers of the basic period. The objective is to find the optimal campaigns, their cycle times and production (running) times while minimizing average inventory (cycle stock inventory) and meeting customer demands. The mathematical formulation is described in the following section.

## 4.2 Mathematical Model

The following notation is used in the PoT optimization model:

**Sets:**

- $P$  set of products
- $C$  set of campaigns
- $K$  set of campaign coverages

**Parameters:**

- $p$  index for products ( $p \in P$ )
- $c$  index for campaigns ( $c \in C$ )
- $k$  index for campaign coverages ( $k \in K$ )
- $\tau_0$  basic production period (year)
- $T_k$  available cycle time (year) if campaign coverage  $k$  is chosen ( $T_k = \tau_0 2^k$ )
- $N_k$  annual number of cycles if campaign coverage  $k$  is chosen ( $N_k = \frac{1}{\tau_0 2^k}$ )
- $D_p$  annual demand for product type  $p$  ( $ft^3$ )
- $\alpha_{cp}$  yield (fractional output) of product type  $p$  per unit input of campaign type  $c$
- $R_c$  production input rate for campaign type  $c$  ( $ft^3$ )
- $S_c$  setup time (year) for campaign type  $c$
- $\nu_p$  inventory value (associated to cycle stock) of product type  $p$  ( $\frac{\$}{ft^3}$ )
- $\rho$  penalty cost for the violation of meeting demand requirements ( $\frac{\$}{ft^3}$ )

**Variables:**

- $t_{ck}$  production time per run (year) for campaign type  $c$  with coverage  $k$ .
- $y_{ck}$  campaign selection =1 if campaign coverage  $k$  is chosen for campaign type  $c$   
=0 otherwise
- $\delta_p^+$  amount by which annual production of product  $p$  exceeds its demand
- $\delta_p^-$  amount by which annual production of product  $p$  fails to meet its demand
- $\Delta_p$  the maximum violation of demand requirements of product  $p$
- $I_p$  the cycle stock inventory of product  $p$

The mathematical formulation of the problem is given as follows:

$$\text{Minimize } \sum_p \nu_p I_p + \rho \sum_p \Delta_p \left( \frac{D_p}{\sum_p D_p} \right) \quad (4.1)$$

subject to :

$$\sum_k y_{ck} \leq 1, \quad \forall c \in C \quad (4.2)$$

$$t_{ck} \leq y_{ck} T_k, \quad \forall c \in C, \forall k \in K \quad (4.3)$$

$$\sum_c \sum_k (N_k t_{ck} + N_k S_c y_{ck}) \leq 1 \quad (4.4)$$

$$\sum_c \sum_k (N_k t_{ck} R_c \alpha_{cp}) = D_p + \delta_p^+ - \delta_p^-, \quad \forall p \in P \quad (4.5)$$

$$\delta_p^+ + \delta_p^- \leq \Delta_p, \quad \forall p \in P \quad (4.6)$$

$$I_p = \frac{1}{2} \sum_c \sum_k (t_{ck} R_c \alpha_{cp}), \quad \forall p \in P \quad (4.7)$$

$$t_{ck}, \delta_p^+, \delta_p^-, \Delta_p \geq 0, y_{ck} \in \{0, 1\}, \quad \forall p \in P, \forall c \in C, \forall k \in K \quad (4.8)$$

The objective function (4.1) aims to minimize the total cycle stock inventory of all products while meeting demand requirements. The first term of the function represents cycle stock inventory of each product multiplied by the inventory value. The second term, indicates that the violations of demand requirements are penalized by enforcing a penalty cost for the products that are underproduced or overproduced. We put more focus on minimizing the violation of demand fulfil-

ment for the products that have higher demands by assuming weighted penalty cost as  $\rho(\frac{D_p}{\sum_p D_p})$  for each product  $p$ .

The first constraint (4.2) ensures at most one campaign coverage  $k$  for each campaign type  $c$ , because  $y_{ck}$ s are binary. A campaign may or may not be selected, but if it is, it has to have exactly one specific cycle time. For instance, if allowable campaign coverages are 0, 1 and 2, this constraint enforces  $y_{c0} + y_{c1} + y_{c2} \leq 1$  for the specific campaign  $c$ , which means either all the variables equal zero or only one of them is equal to 1 and the rest equal zero. Assuming the basic period to be “one week”, campaign  $c$  can only be processed every week ( $k=0$ ) or every two weeks ( $k=1$ ) or every 4 weeks ( $k=2$ ) or not at all.

The next constraint (4.3) ensures that the production time (duration of campaign run) ( $t_{ck}$ ) of campaign type  $c$  with coverage  $k$  can have a positive value, if and only if coverage  $k$  has been chosen for campaign  $c$  and this value should be less than the available cycle time  $T_{ck}$ .

Each time a campaign is processed, it consumes a setup time of  $S_c$  and a run time of  $t_{ck}$ . This occurs  $N_k$  times per year. The annual production and setup time of campaign  $c$  with coverage  $k$  is  $N_k t_{ck} + N_k S_c$  fraction of a year. For instance, assume campaign  $c$  is set to be processed every 4 weeks ( $k=2$ ) for  $t_{c2}$  units production time ( $y_{c2} = 1$ ). This campaign has the setup time  $S_c$ . Assuming that basic period is one week, and one year includes 52 weeks,  $N_2 = \frac{52}{2} = 13$  (number of runs per year) and total time occupied by campaign  $c$  with coverage  $k = 2$  is  $13t_{c2} + 13S_c$ . The summation over all campaigns and all coverages gives the total production plus setup time. Thus, constraint 4.4 is a capacity constraint.

Constraint 4.5 presents the demand requirements where the total amount of each product resulting from all campaigns should meet its annual demand (by considering failure to meet or exceed demand).  $\delta_p^+$  and  $\delta_p^-$ , are surplus and slack variables for deviations from demand fulfillment. The production rate of product  $p$  resulted from processing campaign  $c$  is given by  $R_c \alpha_{cp}$ . Therefore, total amount of product  $p$  produced from campaign  $c$  with coverage  $k$  over a single run

is  $t_{ck}R_c\alpha_{cp}$ . Consequently,  $\sum_c \sum_k (N_k t_{ck} R_c \alpha_{cp})$  is the total amount of product  $p$  produced from processing all the eligible campaigns over all cycles.

The model allows overproduction and underproduction to occur but at the same time, attempts to minimize them. This is done through the next constraint (4.6) where the violation of demand fulfilment of each product should be less than a relevant variable ( $\Delta_p$ ) which we try to minimize in the objective function. Thus, overproduction ( $\delta_p^+$ ) or underproduction ( $\delta_p^-$ ) of product  $p$  will be minimized through  $\Delta_p$  by applying a penalty cost ( $\rho$ ).

Equation 4.7 is an approximation of average cycle stock inventory of product type  $p$  resulted from all possible campaigns.  $I_p$  is the average cycle stock inventory if the production was instantaneous. It is equal to half of the total amount of product  $p$  produced over all cycles of the related campaigns. It is calculated through equations 4.9- 4.11.

$$Q_{cp} = \sum_k t_{ck} R_c \alpha_{cp}, \quad \forall c \in C, \forall p \in P \quad (4.9)$$

$$I_{cp} = \frac{1}{2} Q_{cp} = \frac{1}{2} \sum_k t_{ck} R_c \alpha_{cp}, \quad \forall c \in C, \forall p \in P \quad (4.10)$$

$$I_p = \frac{1}{2} \sum_c Q_{cp} = \frac{1}{2} \sum_c \sum_k (t_{ck} R_c \alpha_{cp}), \quad \forall p \in P \quad (4.11)$$

Where,

$Q_{cp}$ : amount of product  $p$  produced from campaign  $c$ .

$I_{cp}$ : half of the production of product  $p$  due to campaign  $c$ .

To calculate the actual average inventory we would need to consider the demand rate of each product due to campaign  $c$  ( $d_{cp}$ ) shown as equation 4.12.

$$d_{cp} = \sum_k N_k t_{ck} R_c \alpha_{cp}, \quad \forall c \in C, \forall p \in P \quad (4.12)$$

Thus, the actual average inventory of product  $p$  can be calculated through equa-

tions 4.13- 4.15.

$$\tilde{Q}_{cp} = \sum_k t_{ck} R_c \alpha_{cp} - \sum_k t_{ck} d_{cp}, \quad \forall c \in C, \forall p \in P \quad (4.13)$$

$$\tilde{I}_{cp} = \frac{1}{2} \tilde{Q}_{cp} = \frac{1}{2} \sum_k (t_{ck} R_c \alpha_{cp} - t_{ck} d_{cp}), \quad \forall c \in C, \forall p \in P \quad (4.14)$$

$$\tilde{I}_p = \frac{1}{2} \sum_c \tilde{Q}_{cp} = \frac{1}{2} \sum_c \sum_k (t_{ck} R_c \alpha_{cp} - t_{ck} d_{cp}), \quad \forall p \in P \quad (4.15)$$

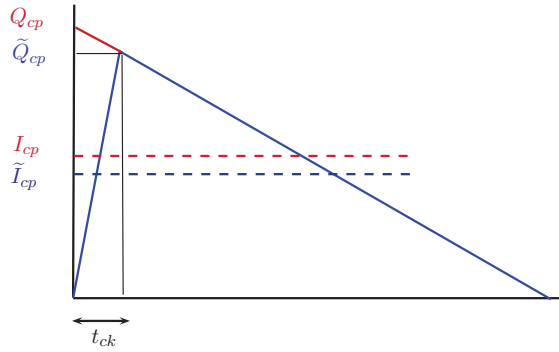
Where,

$d_{cp}$ : the amount of demand  $d$  produced using campaign  $c$  and  $D_p = \sum_c d_{cp} + \delta_p^+ - \delta_p^-$ , ( $\forall p \in P$ ).

$\tilde{Q}_{cp}$ : maximum amount of product  $p$  produced from campaign  $c$  (considering demand rate).

$\tilde{I}_{cp}$ : average inventory of product  $p$  due to campaign  $c$ .

The difference between  $I_{cp}$  and  $\tilde{I}_{cp}$  is shown in figure 4.1.



**Figure 4.1:** Difference between  $I_{cp}$  and  $\tilde{I}_{cp}$

The second term in equation 4.15 is non-linear. To avoid dealing with non-linear integer programming, we use the estimation of average cycle stock inventory represented by equation 4.11 for the optimization model recognizing that this is only an approximation. After the fact, we can calculate the actual average cycle stocks using 4.15. We will present a numerical example of this estimation for one product in Appendix J. The last set of constraints demonstrate the non-



negativity requirements of all continuous decision variables and binary values for the campaign selection variables ( $y_{ck}$ ).

### 4.2.1 Strengthening Constraints

In this section, we suggest adding two more constraints to the model, called “strengthening constraints” that enable us to solve the model quicker. Williams [53] suggested to use tightening bounds to simplify the IP model as well as LP models. However, in this research we work with a MIP model.

This PoT model is a mixed integer programming problem where both integer and linear programming problems should converge to an optimum point. When the size of problem is large (e.g. 70 demands with 126 campaigns) the model can not converge with 0% gap. Our approach was to add two more constraints (4.16 and 4.17) that tighten constraints 4.5 and 4.6 and accelerate the optimization process.

$$N_k t_{ck} R_c \alpha_{cp} \leq D_p y_{ck} + \delta_p^+, \quad \forall p \in P, \forall c \in C, \forall k \in K \quad (4.16)$$

$$\sum_p \Delta_p D_p \leq \mu \sum_p D_p^2 \quad (4.17)$$

Constraint 4.16 is always a true statement that greatly reduces the size of the problem by forcing individual upper bounds for  $N_k t_{ck} R_c \alpha_{cp}$  for each combination of  $c$ ,  $p$  and  $k$ . By putting an upper bound for each  $N_k t_{ck} R_c \alpha_{cp}$ , we tighten the bounds and thus, the model reaches the optimal solution faster.

In inequality 4.17, we dictate a limitation on each  $\Delta_p$  such that the summation of weighted  $\Delta_p$ s is always less than a value. This value is proposed to be a coefficient ( $\mu$ ) multiplied by the summation of demand squared. In other words, if we let  $\Delta_p$  be  $D_p$  in the objective function, then we get the summation of  $D_p$ s.  $\mu$  puts a limitation that this average weighted deviation should not be more than  $\mu$  percent of 100% for every product.

We compare the model performance with and without these constraints by using the data of example 2 (Section 4.3.2.2).

In Section 4.3, we present two examples to show the performance of the PoT model in two different situations.

## 4.3 Examples

This section presents two examples to evaluate the performance of the PoT model in an assumed sawmill production process where it faces two different demand flows. In the first example, specific products (1x3s and 2x3s) have the highest demand among other products. While, in the second example, we have the demand for almost every product. The assumed data of two examples are presented in Section 4.3.1. Results of the PoT model are described in Section 4.3.2.

### 4.3.1 Data

We consider a sawmill in which the assumed annual production capacity (nominal) is 80,000,000 (*fbm*). The *fbm* unit represents the volume of lumber with 1 foot length, 1 foot width and 1 inch thickness. However we convert the capacity unit to  $ft^3$  to keep consistency in calculations. Thus the nominal production capacity is equal to 6,666,666.67 ( $ft^3$ ).

126 campaigns were generated as candidate campaigns for demand of all or some of the 70 products. The products are those discussed in Chapter 3, Section 3.2.1. The campaigns considered here are those discussed in Section 3.2.2.

Two examples of demand are considered. In the first example, special products such as 1x3s and 2x3s have high demands. These products are not the most economically important products. However, every log can produce these products. We try to evaluate the performance of the PoT model with a given campaign setting to fill the demand example where only few products need to be produced. The demand generation procedure for this example is described as followings:

First random numbers are created within ranges (1000-3000) ( $ft^3$ ) for products 1x3s and 2x3s and (0-100) ( $ft^3$ ) for other products. Then they are normalized such that the amounts over all products add up to the production capacity of the sawmill (6,666,666.67 ( $ft^3$ )). The resulting values are assumed to form the demand. Also, the inventory cost ( $\nu_p$ ) is assumed as %20 of the market unit price of product  $p$ .

Table 4.1 illustrates the annual constant demand (nominal  $ft^3$ ) and value per unit inventory ( $\$/ft^3$ ) of each product.

| Product # | Dimensions (in)x(in)x(ft) | $D_p$ (000 $ft^3$ ) | $\nu_p$ ( $\$/ft^3$ ) | Product # | Dimensions (in)x(in)x(ft) | $D_p$ (000 $ft^3$ ) | $\nu_p$ ( $\$/ft^3$ ) |
|-----------|---------------------------|---------------------|-----------------------|-----------|---------------------------|---------------------|-----------------------|
| 1         | 1x3x8                     | 688.47              | 0.884                 | 36        | 2x4x8                     | 27.87               | 1.098                 |
| 2         | 1x3x10                    | 646.98              | 0.904                 | 37        | 2x4x10                    | 6.23                | 1.117                 |
| 3         | 1x3x12                    | 330.43              | 0.923                 | 38        | 2x4x12                    | 22.94               | 1.136                 |
| 4         | 1x3x14                    | 445.06              | 0.943                 | 39        | 2x4x14                    | 25.25               | 1.156                 |
| 5         | 1x3x16                    | 565.17              | 0.962                 | 40        | 2x4x16                    | 19.55               | 1.175                 |
| 6         | 1x4x8                     | 10.18               | 0.917                 | 41        | 2x6x8                     | 3.27                | 1.148                 |
| 7         | 1x4x10                    | 3.65                | 0.936                 | 42        | 2x6x10                    | 29.23               | 1.168                 |
| 8         | 1x4x12                    | 17.38               | 0.956                 | 43        | 2x6x12                    | 14.73               | 1.187                 |
| 9         | 1x4x14                    | 15.82               | 0.975                 | 44        | 2x6x14                    | 9.41                | 1.206                 |
| 10        | 1x4x16                    | 28.24               | 0.995                 | 45        | 2x6x16                    | 6.65                | 1.226                 |
| 11        | 1x6x8                     | 12.39               | 0.982                 | 46        | 2x8x8                     | 5.96                | 1.199                 |
| 12        | 1x6x10                    | 9.08                | 1.001                 | 47        | 2x8x10                    | 22.17               | 1.218                 |
| 13        | 1x6x12                    | 11.25               | 1.021                 | 48        | 2x8x12                    | 29.40               | 1.238                 |
| 14        | 1x6x14                    | 15.95               | 1.040                 | 49        | 2x8x14                    | 26.60               | 1.257                 |
| 15        | 1x6x16                    | 21.58               | 1.060                 | 50        | 2x8x16                    | 16.56               | 1.276                 |
| 16        | 1x8x8                     | 11.38               | 1.047                 | 51        | 2x10x8                    | 2.51                | 1.250                 |
| 17        | 1x8x10                    | 3.86                | 1.066                 | 52        | 2x10x10                   | 19.27               | 1.269                 |
| 18        | 1x8x12                    | 1.58                | 1.086                 | 53        | 2x10x12                   | 7.06                | 1.288                 |
| 19        | 1x8x14                    | 27.81               | 1.105                 | 54        | 2x10x14                   | 15.88               | 1.307                 |
| 20        | 1x8x16                    | 27.21               | 1.125                 | 55        | 2x10x16                   | 22.03               | 1.327                 |
| 21        | 1x10x8                    | 8.63                | 1.112                 | 56        | 2x12x8                    | 11.50               | 1.300                 |
| 22        | 1x10x10                   | 26.41               | 1.131                 | 57        | 2x12x10                   | 5.36                | 1.320                 |
| 23        | 1x10x12                   | 7.99                | 1.151                 | 58        | 2x12x12                   | 14.71               | 1.339                 |
| 24        | 1x10x14                   | 13.26               | 1.170                 | 59        | 2x12x14                   | 22.68               | 1.358                 |
| 25        | 1x10x16                   | 25.88               | 1.190                 | 60        | 2x12x16                   | 26.94               | 1.377                 |
| 26        | 1x12x8                    | 26.46               | 1.177                 | 61        | 4x4x8                     | 29.12               | 1.459                 |
| 27        | 1x12x10                   | 0.16                | 1.196                 | 62        | 4x4x10                    | 13.20               | 1.478                 |
| 28        | 1x12x12                   | 12.30               | 1.216                 | 63        | 4x4x12                    | 25.19               | 1.497                 |
| 29        | 1x12x14                   | 23.70               | 1.235                 | 64        | 4x4x14                    | 12.06               | 1.517                 |
| 30        | 1x12x16                   | 9.84                | 1.255                 | 65        | 4x4x16                    | 12.75               | 1.536                 |
| 31        | 2x3x8                     | 832.59              | 1.072                 | 66        | 6x6x8                     | 3.78                | 1.813                 |
| 32        | 2x3x10                    | 556.50              | 1.092                 | 67        | 6x6x10                    | 24.63               | 1.832                 |
| 33        | 2x3x12                    | 690.47              | 1.111                 | 68        | 6x6x12                    | 20.19               | 1.851                 |
| 34        | 2x3x14                    | 333.64              | 1.131                 | 69        | 6x6x14                    | 15.00               | 1.871                 |
| 35        | 2x3x16                    | 632.53              | 1.150                 | 70        | 6x6x16                    | 3.12                | 1.890                 |

Table 4.1: Product data for example 1

Figure 4.2 illustrates the annual demands.

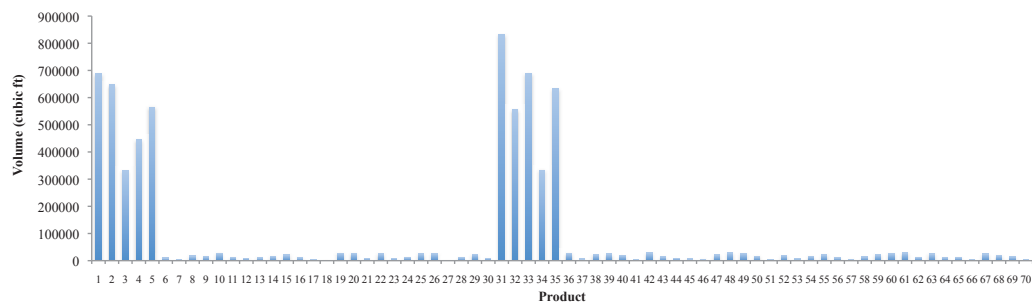


Figure 4.2: Demand of products, Example 1

As shown in this figure, demand for 1x3 and 2x3 products is set to be higher than others.

In the second example, the sawmill faces demand widely distributed among different products. The demand generation procedure for this example is described as followings:

The resulting volume ( $ft^3$ ) of each product (70 products) from every campaign (126 campaigns) through processing 100,000 logs over 20 price list is accessible from the cutting pattern generator. The average amount of each product over all 126 campaigns is calculated and normalized such that the amounts over all products add up to the production capacity of the sawmill (6,666,666.67 ( $ft^3$ )). The resulting values are assumed to form the demands.  $\nu_p$  is again assumed to be %20 of the market unit price of product  $p$ .

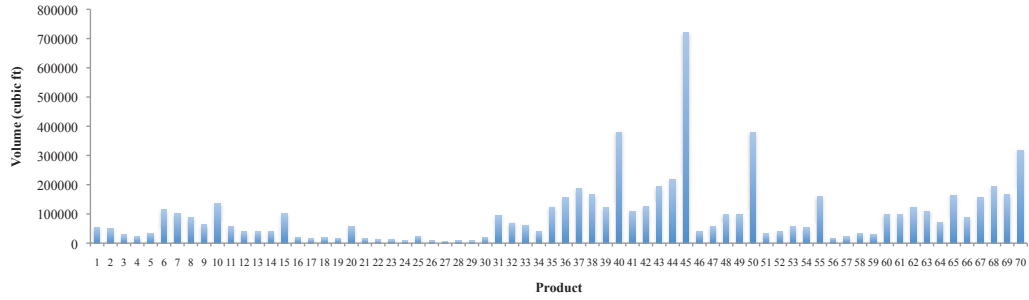
Table 4.2 shows the information about every product.

| Product # | Dimensions (in)x(in)x(ft) | $D_p$ (000 $ft^3$ ) | $\nu_p$ ( $\$/ft^3$ ) | Product # | Dimensions (in)x(in)x(ft) | $D_p$ (000 $ft^3$ ) | $\nu_p$ ( $\$/ft^3$ ) |
|-----------|---------------------------|---------------------|-----------------------|-----------|---------------------------|---------------------|-----------------------|
| 1         | 1x3x8                     | 54.33               | 0.884                 | 36        | 2x4x8                     | 156.40              | 1.098                 |
| 2         | 1x3x10                    | 48.44               | 0.904                 | 37        | 2x4x10                    | 185.51              | 1.117                 |
| 3         | 1x3x12                    | 27.95               | 0.923                 | 38        | 2x4x12                    | 164.84              | 1.136                 |
| 4         | 1x3x14                    | 21.79               | 0.943                 | 39        | 2x4x14                    | 120.40              | 1.156                 |
| 5         | 1x3x16                    | 33.24               | 0.962                 | 40        | 2x4x16                    | 379.48              | 1.175                 |
| 6         | 1x4x8                     | 113.46              | 0.917                 | 41        | 2x6x8                     | 108.67              | 1.148                 |
| 7         | 1x4x10                    | 100.36              | 0.936                 | 42        | 2x6x10                    | 124.79              | 1.168                 |
| 8         | 1x4x12                    | 87.58               | 0.956                 | 43        | 2x6x12                    | 192.11              | 1.187                 |
| 9         | 1x4x14                    | 63.25               | 0.975                 | 44        | 2x6x14                    | 218.21              | 1.206                 |
| 10        | 1x4x16                    | 134.93              | 0.995                 | 45        | 2x6x16                    | 721.34              | 1.226                 |
| 11        | 1x6x8                     | 56.01               | 0.982                 | 46        | 2x8x8                     | 39.99               | 1.199                 |
| 12        | 1x6x10                    | 39.69               | 1.001                 | 47        | 2x8x10                    | 57.25               | 1.218                 |
| 13        | 1x6x12                    | 39.15               | 1.021                 | 48        | 2x8x12                    | 96.75               | 1.238                 |
| 14        | 1x6x14                    | 40.35               | 1.040                 | 49        | 2x8x14                    | 97.32               | 1.257                 |
| 15        | 1x6x16                    | 99.98               | 1.060                 | 50        | 2x8x16                    | 378.92              | 1.276                 |
| 16        | 1x8x8                     | 20.37               | 1.047                 | 51        | 2x10x8                    | 31.59               | 1.250                 |
| 17        | 1x8x10                    | 14.55               | 1.066                 | 52        | 2x10x10                   | 38.96               | 1.269                 |
| 18        | 1x8x12                    | 18.82               | 1.086                 | 53        | 2x10x12                   | 56.87               | 1.288                 |
| 19        | 1x8x14                    | 16.75               | 1.105                 | 54        | 2x10x14                   | 52.63               | 1.307                 |
| 20        | 1x8x16                    | 56.72               | 1.125                 | 55        | 2x10x16                   | 160.55              | 1.327                 |
| 21        | 1x10x8                    | 16.11               | 1.112                 | 56        | 2x12x8                    | 16.68               | 1.300                 |
| 22        | 1x10x10                   | 10.87               | 1.131                 | 57        | 2x12x10                   | 21.40               | 1.320                 |
| 23        | 1x10x12                   | 11.35               | 1.151                 | 58        | 2x12x12                   | 34.59               | 1.339                 |
| 24        | 1x10x14                   | 7.87                | 1.170                 | 59        | 2x12x14                   | 30.54               | 1.358                 |
| 25        | 1x10x16                   | 23.08               | 1.190                 | 60        | 2x12x16                   | 98.48               | 1.377                 |
| 26        | 1x12x8                    | 7.84                | 1.177                 | 61        | 4x4x8                     | 97.12               | 1.459                 |
| 27        | 1x12x10                   | 6.61                | 1.196                 | 62        | 4x4x10                    | 122.72              | 1.478                 |
| 28        | 1x12x12                   | 7.67                | 1.216                 | 63        | 4x4x12                    | 109.17              | 1.497                 |
| 29        | 1x12x14                   | 7.36                | 1.235                 | 64        | 4x4x14                    | 71.70               | 1.517                 |
| 30        | 1x12x16                   | 19.43               | 1.255                 | 65        | 4x4x16                    | 164.24              | 1.536                 |
| 31        | 2x3x8                     | 95.63               | 1.072                 | 66        | 6x6x8                     | 88.36               | 1.813                 |
| 32        | 2x3x10                    | 68.87               | 1.092                 | 67        | 6x6x10                    | 157.49              | 1.832                 |
| 33        | 2x3x12                    | 61.64               | 1.111                 | 68        | 6x6x12                    | 193.89              | 1.851                 |
| 34        | 2x3x14                    | 40.46               | 1.131                 | 69        | 6x6x14                    | 166.17              | 1.871                 |
| 35        | 2x3x16                    | 120.60              | 1.150                 | 70        | 6x6x16                    | 318.41              | 1.890                 |

Table 4.2: Product data for Example 2

Figure 4.3 demonstrates the annual demand of products.

If downtime is excluded, the actual input rate of each campaign can be calculated from equation 4.18. The volume percent yield of each campaign was



**Figure 4.3:** Demand of products, Example 2

calculated in Chapter 3. Appendix H provides input rate and setup time of each campaign.

$$Input\ Rate = \frac{Production\ Output}{Campaign\ Percent\ Yield} \quad (4.18)$$

Total available hours per year is 1820 hours (i.e.  $52(week/year)*5(days/week)*7(hours/day)$ ); therefore, a campaign’s setup time of 1 *hour* is 0.00055 *year*. The fractional volume output of each product per unit input of each campaign ( $\alpha_{cp}$ ) is presented in Appendix I. The basic production period ( $\tau_0$ ) in this problem is assumed to be one week (i.e.  $\frac{1}{52}$  year). Also, the maximum value for the campaign coverage ( $k$ ) is set to 4. This means that if each campaign needs to be run, it is processed at least once every 16 weeks. The assumed value for  $\rho$  is set to 50  $\$/ft^3$ . This value is multiplied by  $D_p/\sum_p D_p$  of each product in the objective function. Therefore, the products with higher demands have more violation-of-demand costs. This helps the optimization problem minimize the violation of those demand requirements prior to the others.

$\mu$  is found through trial and error and set to 0.22 for Example 1 and 0.8 for Example 2. The optimization problem is coded in GNU Linear Programming Kit (GLPK) environment (Makhorin [30]). The solution is then obtained by using Gurobi Optimizer (Inc. [25]).

## 4.3.2 Results

### 4.3.2.1 Example 1

The optimal solution of Example 1 is found in 11 seconds. It has an objective function value of \$6,416,042 with an optimality gap 0.000%. Details of the selected campaigns with their production time and frequencies are shown in Table 4.3.

| $c$ | $k$ | $t_{ck}(\text{hour})$ | $N_k$ | $N_k t_{ck}$ | $N_k S_c$ | Description    |
|-----|-----|-----------------------|-------|--------------|-----------|----------------|
| 6   | 3   | 87.21                 | 6.5   | 566.89       | 7.66      | Every 8 weeks  |
| 10  | 3   | 53.08                 | 6.5   | 345.03       | 4.79      | Every 8 weeks  |
| 46  | 4   | 34.68                 | 3.25  | 112.71       | 2.64      | Every 16 weeks |
| 79  | 3   | 32.05                 | 6.5   | 208.30       | 4.65      | Every 8 weeks  |
| 97  | 4   | 43.15                 | 3.25  | 140.23       | 2.27      | Every 16 weeks |
| 116 | 3   | 64.45                 | 6.5   | 418.92       | 5.91      | Every 8 weeks  |

Table 4.3: PoT solution, Example 1

The model chooses 6 campaigns from 126 potential campaigns. Total production time is 1792.08 hours ( $\sum_c \sum_k N_k t_{ck}$ ) and total setup time is 27.92 hours ( $\sum_c \sum_k N_k S_c$ ). Thus total capacity utilization is 100%.

The total approximate cycle stock inventory is demonstrated in figure 4.4. Values of the approximate cycle stock inventory variables are provided in table 4.4.

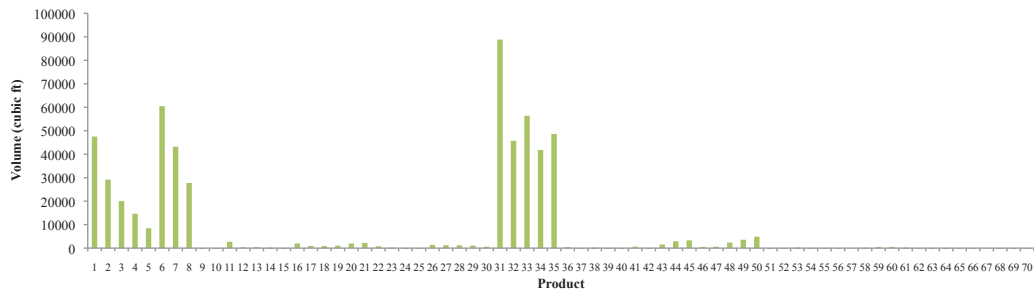


Figure 4.4: Total approximate cycle stock inventory, Example 1

Figure 4.5 compares the amount of each product produces by implementing the PoT optimization (supply) with demand of products.

According to the nature of demand we would expect the model to choose from the campaigns producing more of 1x3 and 2x3 products. Based on the campaigns

| Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ |
|---------|-------------|---------|-------------|---------|-------------|---------|-------------|---------|-------------|
| 1       | 47503.65    | 15      | 134.80      | 29      | 1089.88     | 43      | 1564.36     | 57      | 0.00        |
| 2       | 29184.52    | 16      | 2008.66     | 30      | 605.52      | 44      | 2899.63     | 58      | 207.37      |
| 3       | 20025.12    | 17      | 917.36      | 31      | 88869.09    | 45      | 3320.16     | 59      | 491.35      |
| 4       | 14665.39    | 18      | 885.45      | 32      | 45725.78    | 46      | 523.14      | 60      | 501.77      |
| 5       | 8490.06     | 19      | 1067.70     | 33      | 56380.17    | 47      | 601.61      | 61      | 221.09      |
| 6       | 60465.51    | 20      | 1959.44     | 34      | 41808.45    | 48      | 2345.06     | 62      | 9.93        |
| 7       | 43196.68    | 21      | 2230.52     | 35      | 48656.46    | 49      | 3579.71     | 63      | 141.50      |
| 8       | 27740.79    | 22      | 774.53      | 36      | 484.19      | 50      | 4854.38     | 64      | 93.59       |
| 9       | 60.83       | 23      | 262.00      | 37      | 31.95       | 51      | 63.10       | 65      | 3.40        |
| 10      | 19.28       | 24      | 134.69      | 38      | 199.38      | 52      | 30.22       | 66      | 20.04       |
| 11      | 2712.71     | 25      | 63.78       | 39      | 165.34      | 53      | 262.06      | 67      | 0.00        |
| 12      | 384.56      | 26      | 1386.00     | 40      | 34.02       | 54      | 92.17       | 68      | 40.32       |
| 13      | 471.72      | 27      | 1241.64     | 41      | 621.30      | 55      | 89.30       | 69      | 14.04       |
| 14      | 222.18      | 28      | 1190.68     | 42      | 159.25      | 56      | 8.91        | 70      | 15.31       |

Table 4.4: Total approximate cycle stock inventory of each product, Example 1

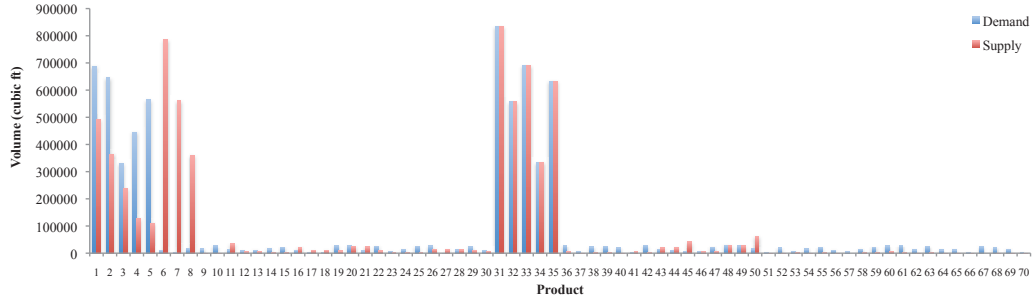


Figure 4.5: Demand vs. supply, Example 1

definition (Appendix G) we expect to have campaigns created by price list 6 (focusing on lumber with width 1 (*in*)), price list 7 (focusing on lumber with width 2 (*in*)) and price list 10 (focusing on lumber with thickness 3 (*in*)). From the obtained results, campaign 6 produces lumber pieces with width 1 (*in*) and the rest focus on lumber pieces with thickness 3 (*in*) (campaigns 10, 46, 79, 97 and 116). Since lumber pieces with thickness 3 (*in*) are just 1x3s and 2x3s, choosing campaigns that use price list 10, results in both product types. This is why most of the campaigns are the ones derived from price list 10.

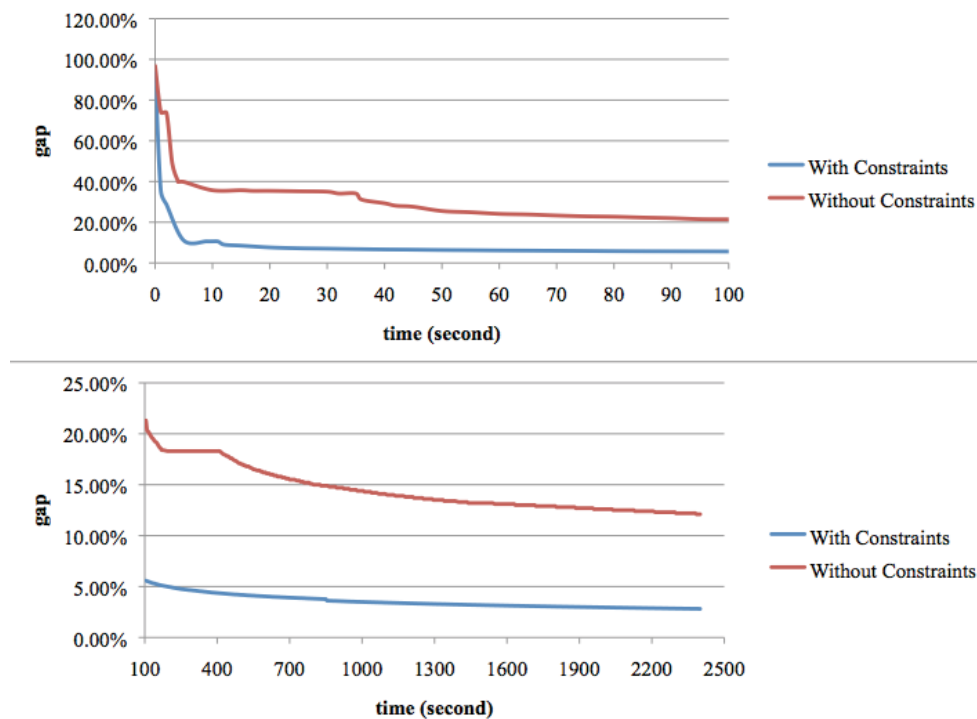
Also, 2x3 lumber pieces (products 31-35) have higher demand. Thus, the model intends to produce these products prior to the other ones. In addition, we see one campaign of price list 6 (focusing on lumber with width 1 (*in*)) that can fill the demand of 1x3s (products 1-5). Using this campaign results in having lumber pieces, other than 1x3s, that have width 1 (*in*) (e.g. 1x4s, 1x6, 1x8s, 1x10s and 1x12s).

In this example, only 6 campaigns are chosen to fill the demand. As shown in figure 4.5, the available campaigns can completely fill the demand of some prod-

ucts such as 2x3s. However, we may have overproduction and underproduction of other products such as 1x3s. To get a better solution, we need to design additional campaigns focusing on the underproduced products. This can be effectively done through feeding back the shadow prices resulting from these products to the campaign generating stage.

#### 4.3.2.2 Example 2

The strengthening constraints discussed in Section 4.2.1 improved the model effectively. We compare the model with and without these constraints. The results in terms of percentage gap between linear and integer solutions are presented in figure 4.6.



**Figure 4.6:** Impact of strengthening constraints on model's performance

As can be seen, the gaps for both models decrease rapidly over the first 50 seconds. The model with strengthening constraints reaches 5% gap in less than 20 seconds. At time 50, the model without constraints has 35% gap. After 100 seconds, both models converge with low speed, but at each point of time the model



with strengthening constraint has less gap than the original model. We use these strengthening constraints to let the model find the optimal solution faster.

In Example 2, the optimal solution is found in 17,433 seconds (4.8 hours) and the objective function value is equal to \$731,710 with an optimality gap of 0.009%. Details of the selected campaigns with their production time and frequencies are shown in Table 4.5.

| $c$ | $k$ | $t_{ck}(\text{hour})$ | $N_k$ | $N_k t_{ck}$ | $N_k S_c$ | Description   |
|-----|-----|-----------------------|-------|--------------|-----------|---------------|
| 15  | 2   | 10.46                 | 13    | 135.93       | 12.56     | Every 4 weeks |
| 19  | 1   | 17.10                 | 26    | 444.65       | 30.62     | Every 2 weeks |
| 37  | 3   | 4.60                  | 6.5   | 29.90        | 5.26      | Every 8 weeks |
| 70  | 3   | 1.54                  | 6.5   | 9.99         | 3.80      | Every 8 weeks |
| 76  | 3   | 9.56                  | 6.5   | 62.14        | 7.40      | Every 8 weeks |
| 87  | 3   | 10.21                 | 6.5   | 66.36        | 9.06      | Every 8 weeks |
| 90  | 3   | 3.31                  | 6.5   | 21.50        | 7.44      | Every 8 weeks |
| 106 | 2   | 8.32                  | 13    | 108.19       | 7.85      | Every 4 weeks |
| 110 | 1   | 11.85                 | 26    | 308.06       | 17.70     | Every 2 weeks |
| 114 | 2   | 4.88                  | 13    | 63.44        | 9.68      | Every 4 weeks |
| 118 | 2   | 6.90                  | 13    | 89.73        | 7.66      | Every 4 weeks |
| 126 | 2   | 26.41                 | 13    | 343.33       | 17.75     | Every 4 weeks |

Table 4.5: PoT solution, Example 2

12 campaigns are selected from 126 campaigns. Total production time is 1683.22 hours ( $\sum_c \sum_k N_k t_{ck}$ ) and total setup time is 136.77 hours ( $\sum_c \sum_k N_k S_c$ ). Thus total capacity utilization is 99.9%.

The total approximate cycle stock inventory of each product is demonstrated in figure 4.7. The details are provided in table 4.6.

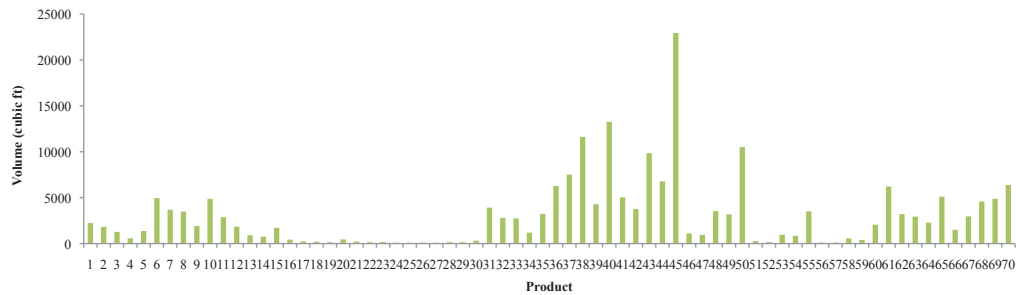


Figure 4.7: Total approximate cycle stock inventory, Example 2

| Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ | Product | $I_p(ft^3)$ |
|---------|-------------|---------|-------------|---------|-------------|---------|-------------|---------|-------------|
| 1       | 2236.29     | 15      | 1708.84     | 29      | 157.79      | 43      | 9844.15     | 57      | 110.37      |
| 2       | 1816.41     | 16      | 433.69      | 30      | 311.27      | 44      | 6783.11     | 58      | 570.59      |
| 3       | 1274.44     | 17      | 242.91      | 31      | 3919.40     | 45      | 22934.51    | 59      | 400.87      |
| 4       | 572.02      | 18      | 206.00      | 32      | 2791.59     | 46      | 1110.08     | 60      | 2054.55     |
| 5       | 1368.73     | 19      | 156.64      | 33      | 2748.76     | 47      | 944.07      | 61      | 6222.31     |
| 6       | 4950.10     | 20      | 454.15      | 34      | 1189.00     | 48      | 3549.99     | 62      | 3207.97     |
| 7       | 3683.55     | 21      | 218.39      | 35      | 3237.94     | 49      | 3183.47     | 63      | 2932.43     |
| 8       | 3484.81     | 22      | 154.95      | 36      | 6263.06     | 50      | 10515.43    | 64      | 2275.59     |
| 9       | 1912.64     | 23      | 171.56      | 37      | 7503.83     | 51      | 265.92      | 65      | 5111.71     |
| 10      | 4872.20     | 24      | 97.42       | 38      | 11624.72    | 52      | 163.51      | 66      | 1504.88     |
| 11      | 2882.19     | 25      | 101.49      | 39      | 4293.89     | 53      | 958.70      | 67      | 2964.23     |
| 12      | 1844.14     | 26      | 111.01      | 40      | 13249.57    | 54      | 838.78      | 68      | 4589.52     |
| 13      | 911.73      | 27      | 92.55       | 41      | 5038.99     | 55      | 3506.53     | 69      | 4870.94     |
| 14      | 752.21      | 28      | 155.39      | 42      | 3771.54     | 56      | 111.16      | 70      | 6375.68     |

Table 4.6: Total approximate cycle stock inventory of each product, Example 2

Figure 4.8 provides information about the amount of each product produced by the PoT optimization (supply) and compares them with demand of products.

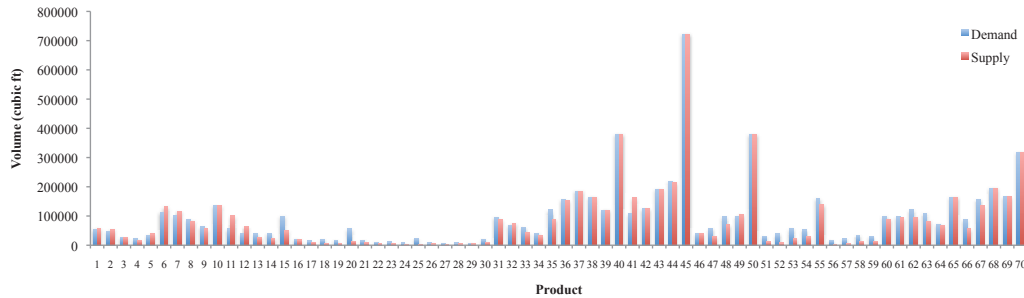


Figure 4.8: Demand vs. supply, Example 2

The three high-demanded products are 45 (2x6x16), 50 (2x8x16) and 40 (2x4x16).

There are several considerations that need to be taken into account:

- All the high-demand products are with length 16 ( $ft$ ). Thus, running campaigns which emphasize 16 ( $ft$ ) lumber pieces was expected. Campaign 126 is a reasonable choice as it consists of the largest logs with lengths between 16 and 18 ( $ft$ ) and price list 20 which emphasizes 16 ( $ft$ ) lumber pieces.
- Campaign 110 emphasizes the larger thicknesses and mainly produces the three products mentioned plus a few others such as 55 (2x10x16) and 60 (2x12x16). As this campaign is generated from the logs with lengths between 16 and 18 ( $ft$ ) its main products are with length 16 ( $ft$ ).
- Demand of products 61-70 (4x4s and 6x6s), are mainly satisfied through campaigns 19 and 90 which are from log classes with price list 3 (focusing

on lumber pieces with larger widths).

- The only products with width 4 (*in*) are 4x4s. Campaign 114 focuses on producing such products.
- Campaign 76 is from log class 5 in which logs are sorted based on length between 12 (*ft*) and 14 (*ft*) and price list 7 (focusing on lumber pieces with width 2 (*in*)). This campaign produces more products with width 2 (*in*) and length 12 (*ft*).
- As can be seen in figure 4.8 by running each campaign, there is excess production of a few products. This results in unnecessary inventory of some products such as 11 and 41.

We can see campaigns from different log classes, and different price lists each focusing on a particular sets of lumber pieces. The frequency and running time of each campaign depend upon various factors, such as its production rate, fractional outputs and setup time. Therefore, the PoT attempts to find an implementable plan that responds well to the problem objectives within the PoT context. In this example, 12 campaigns are selected among 126 campaigns that can fill 90% of the demand requirements.

Various combinations of campaigns may be selected according to different model settings. In all the cases the objective is to have a minimum level of inventory for each product, while meeting its demand requirements under capacity (available time) constraint. The two examples presented here show how the model works for an assumed sawmill with quite different demand requirements.

The model does not produce an EPEI solution. Instead, it chooses several campaigns with different cycle times based on a powers of two multipliers of a basic period. The key feature of our model is that it is applied to campaigns instead of products. Thus, it minimizes setup time and inventory cost over all products.

In general, the PoT model enables sawmill management to manage several campaigns based on desirable outputs and use them to fulfill demand of different products over a planning horizon. If the designed campaigns were not enough to fill all the demand requirements, one can generate more campaigns based on other price lists. This can be done through inputting shadow prices of the demand constraint (4.5) to a column generation approach. However, in the work reported in this thesis we do not take this approach into account.

The main contribution of this model is that we can calculate the minimum possible inventory levels that correspond to a constant annual demand for a complex mix of products produced with the multiple product campaigns typical of a sawmill. The model indicates the types of campaigns necessary to achieve this inventory level and how frequently they should be run. However, the model does not directly provide a production control strategy. The main thing that is missing is the recognition of the initial inventory for all products. As we discussed in Chapter 2, any cyclic economic lot sizing strategy requires the right initial inventory. Secondly, the model is based on constant demand for all products. Again, as discussed in Chapter 2, this is a common assumption for all economic lot sizing approaches. If we want to use this approach as a control mechanism, we need to understand how it responds to uncertainty in demand.

In Chapter 5, the output of this model is used to examine how a few control approaches, that were designed based on the PoT results, respond in a system with stochastic demand.

# Chapter 5

## Control Approaches

The PoT model with annual deterministic demand provides the information about appropriate campaign lot sizes, their running durations (production time) and frequencies (cycle times) to both fulfill demand requirements and minimize inventory. However, it does not give any detailed schedule for running campaigns. As discussed in Chapter 2, all economic lot scheduling approaches have an issue of the effect due to the initial inventory. Also, the assumption of deterministic demand is necessary for the optimization problem. In reality, most sawmills must deal with stochastic demands which makes the campaign lot scheduling problem more complex.

In this chapter, we investigate how some of these issues might affect a sawmill. We examine a number of control approaches for campaign scheduling and simulate the behaviour of inventory based on the results of the PoT. These approaches are not optimization procedures, but present control mechanisms that simulate the stochastic and dynamic nature of the production environment and attempt to evaluate the performance of the PoT model in case of stochastic demands. If the PoT model does not cause supply of each product to equal its demand, then the control approaches can not do this either. Thus, in the remainder of this chapter, we work on the basis that the annual demand is exactly the amount calculated as supplied in the PoT approach.

Instead of constant rate of demand arrival times and sizes in the PoT model, demand may arrive stochastically in a real system. In the cases examined here, demand size and inter-arrival times are assumed to have uniform and exponential distribution functions, respectively; with the total product of expected demand size and expected demand arrival rate equaling the deterministic annual demand of the PoT model. However, the realized demand of a stochastic process is not equal to its expected value so that, over any time interval the actual demand can be at variance from the expected demand in that period.

The lean manufacturing approach of EPEI (see Chapter 2) attempts to control production with constant lot sizes of each product. We investigate here various approaches using the campaign lot sizes coming out of the PoT approach to see how they might perform.

The control step starts with an idle sawmill that has information about campaigns' production times and number of runs (from the PoT solution) and the future demands over a short horizon. The question is what campaign to run next and how much to run. However, we have decided to always use the PoT lot sizes (production time) so that the only control question is what campaign should be processed next.

This chapter begins with the description of some possible control approaches developed within a simulation environment and then evaluates their performance by applying them to Example 2 presented in Chapter 4, over two cases. In Case 1, demand has large arrival frequencies with less variety in sizes to simulate near-constant demand, while in Case 2, the batch order arrivals are “noisier” with varying demand arrivals and sizes (“bulk” arrivals).

## 5.1 Assumptions

In the approaches addressed here, it is assumed that the sales department always provides the production section with the exact dates in which upcoming orders should be filled. Moreover, it is assumed that demand needs to be satisfied exactly at the date specified through a JIT delivery process. In other words, when the production process faces a deadline, the inventory of related products will decrease by the amount requested. If enough inventory is available, the requested product will be delivered. Otherwise, we assume all the on-hand inventory is shipped to the customer and the rest creates negative inventory levels in the form of “back-orders”. When a back-order happens, customers will wait and receive their orders as soon as the required product is produced.

A sawmill cannot know future demand exactly, In the work reported here, we assume that orders that will be occurring within a certain time window of the present are known precisely. This is equivalent to assuming that there is a fixed lead time for each order. For orders outside this lead time window, we assume that the mill has to forecast this demand based upon past demands. In the work reported here, we used a simple exponential-smoothing forecasting method (Silver et al. [43]) to estimate expected future due dates. Exponential smoothing relies on the most recent observations and previous forecast level to estimate the next observation. The simplest formulation of this method is presented in equation 5.1.

$$s_1 = x_0, \quad s_t = \alpha x_{t-1} + (1 - \alpha)s_{t-1}, t > 1 \quad (5.1)$$

where  $s_t$  and  $x_t$  are forecasted output and real observation at time  $t$ , respectively.

Since the demand process is assumed to be intermittent (Silver et al. [43], Section 4.9.2), it is necessary to forecast both the size and delivery date of future orders. Equations 5.2 and 5.3 demonstrate the process for forecasting the time between arrivals and demand size, respectively.

$$Time_p^F = \alpha Time_p^A + (1 - \alpha) Time_p^{F'} \quad (5.2)$$

$$Size_p^F = \alpha Size_p^A + (1 - \alpha) Size_p^{F'} \quad (5.3)$$

where,  $Time_p^F$  is the current forecast for time between arrival.  $Time_p^A$  represents the most recently observed time between arrival and  $Time_p^{F'}$  is the previous forecasted time between arrival for product  $p$ . Also,  $Size_p^F$  is the new forecast of the demand size.  $Size_p^A$  represents the most recently observed order size, and  $Size_p^{F'}$  is the previous forecast for order size of product  $p$ .

For example, the initial values for  $Time_p^{F'}$  and  $Size_p^{F'}$  are set as 10 (days) and 100 (cubic feet) for a given product  $p$ . If the first demand arrives in day 12 ( $Time_p^A = 12$ ) with size 120 (Cubic feet) ( $Size_p^A = 120$ ), if  $\alpha = 0.1$ , the forecasts are updated as follows:

$$Time_p^F = 0.1 * 12 + (0.9) * 10 = 10.2 \text{ (day)}$$

$$Size_p^F = 0.1 * 120 + (0.9) * 100 = 102 \text{ (cubic feet)}$$

The calculations demonstrate the next demand is forecasted to arrive in 10.2 days ahead (in day 22.2) with size 102 (cubic feet). When the next actual demand arrives, the two forecasts will be updated and the two numbers (10.2 and 102) will become the old forecast values ( $Time_p^{F'}$  and  $Size_p^{F'}$ ) for the new forecast.

In developing out control approaches, we assume that the mill would have a lead time window of “m” time units. Thus, the mill would know the orders that are to occur within the next “m” time units and forecast the orders to arrive after that. The mill can then base its control decision on the known and anticipated orders. In an economic order quantity (EOQ) control setting, it is possible to have the EOQ vary with the demand forecast (see Silver et al. [43]). This is not so easy in the case considered here.

All the approaches use available lot sizes (production time of each campaign). Some may also use the frequency of each campaign. The challenge lies in deciding the next campaign to run based on information about available campaigns, lot-



sizes, upcoming demands and on-hand inventory based on the PoT results.

We use simulation to represent inventory behaviour of each approach. From simulation, information about each product's inventory is accessible and can be calculated based on equation 5.4.

$$Inv_t^p = Inv_{t-T_c}^p + R_c \alpha_{cp} T_c - D_{T_c}^p \quad (5.4)$$

where:

$Inv_p^t$ : Inventory of product  $p$  at the end of time  $t$

$R_c$ : Production input rate

$\alpha_{cp}$ : Fractional output of product  $p$  from each unit of campaign  $c$

$T_c$ : Production time of campaign  $c$

$D_{T_c}^p$ : Actual demand of product  $p$  simulated in time interval  $T_c$

This equation calculates the inventory of each product  $p$  based on its inventory at the beginning of the campaign run for campaign  $c$  plus the amount produced over this time minus the demand.

All control strategies assume that the campaign selection decision is repeatedly made at the end of each campaign run. A campaign selection criterion which varies among the control approaches, determines what campaign should be run next. Henceforth, when we say “campaigns”, we mean the eligible ones that have been selected by the PoT model.

In simulation terms, there are two events. One is a demand arrival, which reduces inventory of the relevant product  $p$  and also updates the forecasts of demand size and demand rate (inter-arrival time). The other is the completion of a campaign, which increases all relevant inventories and then chooses the next campaign. We consider 5 control approaches.

Approach 1 attempts to find a campaign that produces maximum amount of

the product that has minimum runout time over all the products.

Approach 2 chooses the best campaign based on minimum expected inventory values (positive and negative estimated costs) at the end of the campaign production run. Neither of the approaches considers allowable campaign frequencies but uses the campaign production times (from the PoT model).

Approach 3 considers the campaign frequencies and their production times without paying attention to the inventory positions and upcoming demands. It schedules campaigns according to the frequencies determined by the PoT model.

Approach 4 is a combination of Approaches 2 and 3. The inventory estimation process (Approach 2) provides a list of three ranked campaigns, based on minimum expected future inventory values, while another list of three candidate campaigns is generated using the campaign frequency requirements (Approach 3). The lists are updated and compared at the end of each production run and the campaign existing in both, is chosen. Each element of list 2 (i.e. more urgent campaigns in terms of the required number of runs) is compared to the elements of list 1 (i.e. campaigns that create less inventory value). Once a common campaign is found in both lists, it is chosen. If no common campaign exists, the first ranked campaign of list 2 will be chosen.

Approach 5, is another combination of Approaches 2 and 3 in which 2 lists are created (same as Approach 4). Each element of list 1 is compared to list 2, and if a common one appears, it will be chosen. In the case of no common campaigns, the first element of list 1 (causing the least future inventory value) will be chosen.

For example list 1 suggests campaigns 3, 4 and 2 in the ascending order of their estimated inventory values ( $[3,4,2]$ ) and list 2 contains campaigns 2, 1, and 3 as the three most urgent campaigns ( $[2,1,3]$ ). Campaigns 2 and 3 are common to both lists, so are eligible to be chosen. Approach 4 chooses campaign 2 (the most urgent) while Approach 5 picks up campaign 3 (causing the least future inventory value).

Simulation is done using Python programming language and related code is

presented in Appendix M. The following sections detail the proposed approaches.

## 5.2 Approach 1

In the first approach, the campaign selection criterion is based on the runout time of products. The demands of “ $m$ ” time units ahead are considered at the end of each production run and the product that we are going to runout of first, is selected. Then the campaign that is capable of producing more of the selected product is chosen and processed. Therefore, a specific product is picked each time and the appropriate campaign is processed. The algorithm is as following:

**Step 1:** At the end of a production run, we look at the inventory of all products. Some products may currently have a zero or negative inventory. Thus their run out time is zero. Other products may have an inventory less than the demands in the next “ $m$ ” time units. Their run out time is the time that the sum of the demand sizes exceeds the inventory. In other cases that the current inventory exceeds the sum of demands in the next “ $m$ ” time units, the run out time of each product is computed as formulation 5.5.

$$RO_t^p = m + \frac{(Inv_t^p - \sum_t^{t+m} D_t^p)}{\text{(forecasted demand per unit time)}} \quad (5.5)$$

where  $RO_t^p$  is the run out time of product  $p$  calculated at time  $t$  (i.e. end of a production run).  $Inv_t^p$  is the current inventory of product  $p$  while we are at time  $t$ . The summation of demand from time  $t$  to the end of “ $m$ ” time units ahead is presented as  $\sum_t^{t+m} D_t^p$ .

Based on the status of current inventory, we calculate the amount of time we need for each product to consume what we have in terms of inventory on-hand.

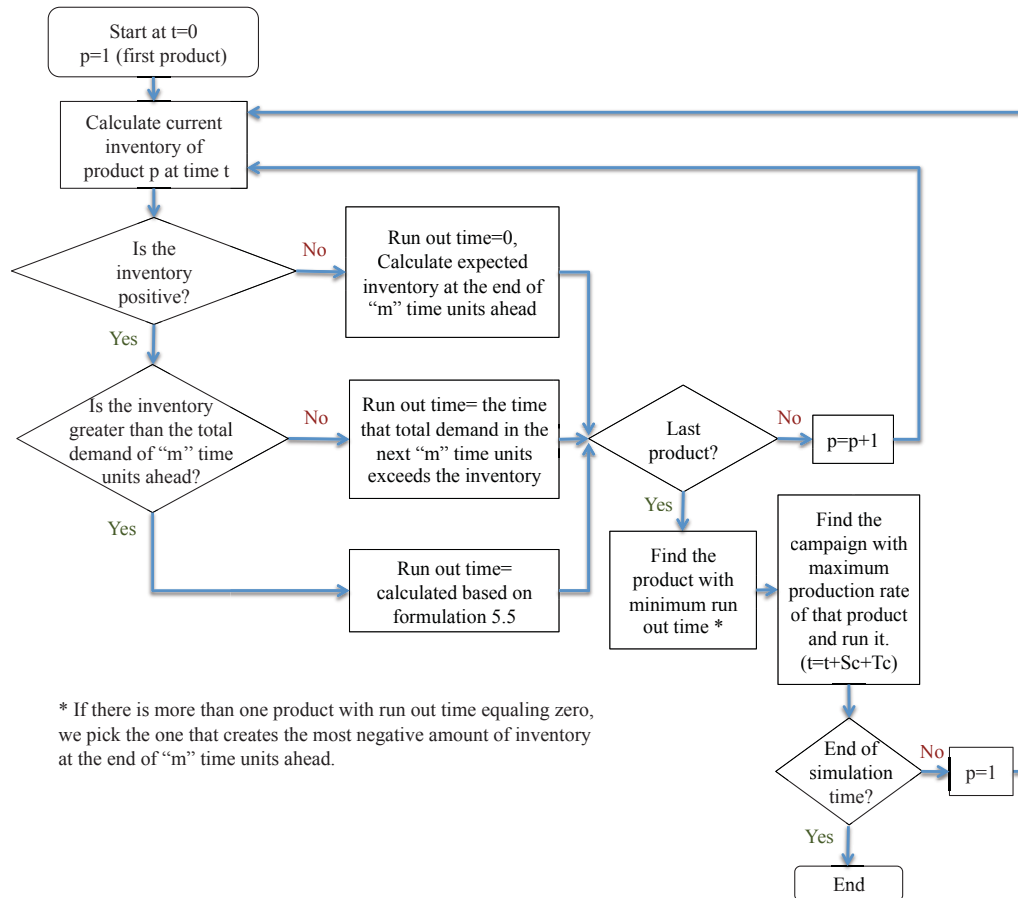
**Step 2:** We find a product that has minimum runout time (the most urgent

product). If there is more than one product with run out time equaling zero, we pick the one that creates the most negative amount of inventory at the end of “m” time units ahead. This product is the one that, if not produced, will result in a large negative inventory in future.

**Step 3:** We select the campaign with maximum production rate ( $R_c\alpha_{cp}$ ) of the given product and process it for the related production time ( $T_c$ ).

**Step 4:** At the end of the production run (time:  $t + S_c + T_c$ ) new due times are available for “m” more time units. So, the due times that were within ( $t + S_c + T_c$ ,  $t + m$ ) are not changed. Again, we go to step 1 and do the calculations. This procedure continues to the end of simulation time.

Figure 5.1 is a simple flowchart illustrating the steps of Approach 1.



**Figure 5.1:** Approach 1 flowchart

This approach does not consider the required campaign frequencies determined by the PoT results as well as estimated inventory values. It only emphasizes the product that will run out first considering the available inventory and demand over time span of  $m$  time units.

### 5.3 Approach 2

This approach chooses the campaign that has minimum future inventory value at the end of its production time. The campaigns are ranked based on their future total inventory values at the end of their production runs and the one with minimum value is selected and processed for the associated amount of time determined by the PoT model ( $T_c$ ). At the end of each production run, a new decision is made and the procedure continues to the end of the simulation time. The algorithm is described in the following steps:

**Step 1:** At time  $t$  (current time), for each campaign  $c$  (from the PoT model) we calculate the final inventory of every product at the end of its production time ( $T_c$ ) while knowing the order arrivals of products over the campaign production time. This can be done using equation 5.6.

$$INV_t^{cp} = Inv_t^p + R_c \alpha_{cp} T_c - \sum_t^{t+S_c+T_c} D_t^p \quad (5.6)$$

where  $INV_t^{cp}$  is the future inventory of product  $p$  at the end of campaign  $c$  production run (i.e. time  $t + S_c + T_c$ ) when the decision is made at time  $t$ .  $T_c$  is the running duration (production time) of campaign  $c$  resulting from the PoT model.  $D_t^p$  is the actual demand of product  $p$  at time  $t$  and  $\sum_t^{t+S_c+T_c} D_t^p$  represents the summation of product  $p$  demands over campaign  $c$  setup and production time (time interval  $(t, t + S_c + T_c)$ ). If the production time of any campaign is longer than “ $m$ ”, we need to use the forecasted demands for  $S_c + T_c - m$  amount of time.

**Step 2:** We consider a cost per unit of product  $p$  ( $V_p$  which is the inventory value of product  $p$ ) to calculate an estimated cost related to  $INV_t^{cp}$ . Formulations 5.7 and 5.8 are the two possibilities.

$$Cost^+ INV_t^{cp} = INV_t^{cp} * V_p \quad \text{if } INV_t^{cp} \geq 0 \quad (5.7)$$

$$Cost^- INV_t^{cp} = -INV_t^{cp} * V_p \quad \text{if } INV_t^{cp} \leq 0 \quad (5.8)$$

**Step 3:** For each campaign  $c$ , we calculate the total estimated inventory value for all the products using equation 5.9. Then we find the campaign with minimum total estimated inventory value and process it for its related amount of production time ( $T_c$ ).

$$Total Cost_t^c = \sum_p (Cost^+ INV_t^{cp} * y_t^{cp} + Cost^- INV_t^{cp} * (1 - y_t^{cp})) \quad (5.9)$$

where  $y_t^{cp}$  is a binary variable which equals to 1, if calculated  $INV_t^{cp}$  is positive. Otherwise, it is zero.

**Step 4:** At the end of production run (time:  $t + S_c + T_c$ ) of campaign  $c$ , new due times, inventories and forecasts are available for “m” time units ahead. Again, we go to step 1 and do the calculations. This procedure continues to the end of the simulation time.

Figure 5.2 is a simple flowchart illustrating the steps of Approach 2.

The campaign selection criterion for this approach is the campaign with minimum estimated inventory value calculated at the end of each production run. This approach also does not consider the allowable frequencies of campaign runs resulting from the PoT. It tries to eliminate extensive positive and negative inventories over a short horizon.

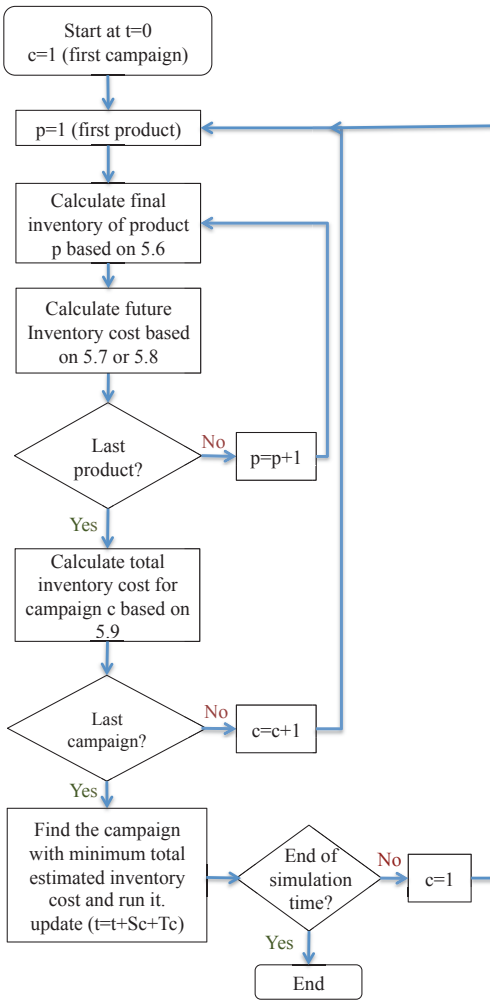


Figure 5.2: Approach 2 flowchart

## 5.4 Approach 3

This approach uses the outputs of the PoT model in terms of eligible campaigns, their production times and frequencies. At the end of each production run, the algorithm searches for the most urgent campaign based on its specified frequency (from the PoT). Thus, it ensures at the end of the year all the campaigns have been processed for the amount of time determined by the PoT model. The algorithm can be described in the following steps:

**Step 1:** At current simulation time  $t$  (i.e. the end of a campaign production),

we calculate the required number of runs up to that point of time for each campaign  $c$ , based on equation 5.10.

$$Required\ Runs_t^c = \frac{F_c}{Total\ Time} * (t - t_0) \quad (5.10)$$

where  $F_c$  is the number of runs (frequency) of campaign  $c$  resulting from the PoT model over a year and  $Total\ Time$  is the time horizon of that model (e.g. one year).  $t_0$  is the simulation start time.  $(t - t_0)$  and  $Total\ Time$  have the same time units (e.g. hours).

**Step 2:** We calculate the difference between required and actual number of runs up to the current time of the simulation ( $t$ ) and find the campaign that has maximum difference based on equation 5.11 and process it for the associated amount of production time ( $T_c$ ).

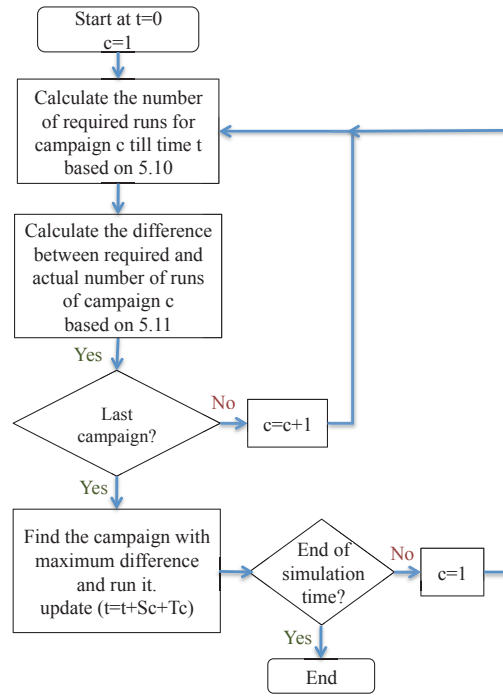
$$Difference_t^c = Required\ Runs_t^c - Actual\ Runs_t^c \quad (5.11)$$

**Step 3:** At the end of the production run (time:  $t + S_c + T_c$ ) we go to step 1 and do the calculations again. This procedure continues to the end of the simulation time.

Figure 5.3 is a simple flowchart illustrating the steps of Approach 3.

In this approach, we only rely on the results of the PoT model and try to implement them without consideration for inventory positions and demand occurrence. This resembles a “push system” in which the production runs are dictated by a set of predefined schedules, which in this case are initiated by the given number of campaign runs and their production durations .





**Figure 5.3:** Approach 3 flowchart

## 5.5 Approach 4

As briefly described earlier, approach 4 combines the criteria of Approaches 2 and 3. The campaign selection criterion is based on both the campaign frequency requirements and inventory value estimation by giving more priority to the former. This approach is addressed in the following steps:

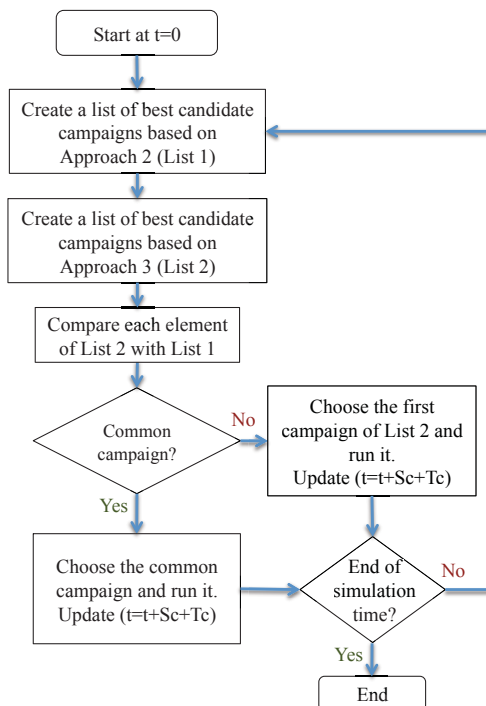
**Step 1:** At time  $t$  (the end of a production run), for each eligible campaign  $c$  we estimate the final inventory of each product at the end of its production run resulted from the PoT (based on Approach 2). This can be done through equation 5.6. Then the associated inventory value is calculated based on equations 5.7 and 5.8 for each product  $p$  from each campaign  $c$ . Also, the total estimated inventory value resulting from each campaign  $c$  is calculated according to formulation 5.9. A list of the 3 best campaigns is created based on ascending order of the estimated future inventory costs (list 1).

**Step 2:** At the same time as Step 1, for each campaign  $c$ , we calculate the required number of runs up to that point of time based on equation 5.10 (Approach 3). Then we calculate the difference between the required and actual number of runs for each campaign using equation 5.11 and prepare a list of the 3 most urgent campaigns (list 2).

**Step 3:** For each campaign of list 2 starting from the first element, we check if it exists in list 1. Once a common campaign is found, it is chosen. Otherwise, the first element of list 2 (the most urgent campaign) will be selected and processed.

**Step 4:** At the end of the production run (time:  $t + S_c + T_c$ ) new due times, inventories and forecasts are available for “m” time units ahead. Again, we go to step 1 and find the next campaign. This procedure continues to the end of the simulation time.

Figure 5.4 is a simple flowchart illustrating the steps of Approach 4. In this



**Figure 5.4:** Approach 4 flowchart

approach, we use the two criteria including minimum estimated inventory value and meeting the campaign frequency requirement, by paying more attention to the latter criterion. If a campaign is urgent (based on PoT frequencies) and also it is one of the three best campaigns that creates less estimated inventory value (regardless of its rank in list 1), it has the highest priority. If none of the three urgent campaigns is one of the first three best campaigns (based on the future inventory value), the campaign which needs to be processed first based on the PoT campaign frequencies (ranked first in list 2) is chosen.

## 5.6 Approach 5

This approach is another combination of Approaches 2 and 3. The steps are the same as Approach 4, with a difference in Step 3 where the campaign checking is started from list 1, if any of the campaigns in list 1 also exists in list 2 (regardless of its rank in list 2), it will be chosen. In the case of none common campaigns, the first element of list 1 (causing the least future inventory value) will be selected.

Figure 5.5 is a simple flowchart illustrating the steps of Approach 5.

In this approach more weight is given to campaigns with less future estimated inventory values, while simultaneously we consider the required campaign frequencies. However, at the end of simulation, the frequency of campaign runs may not be equal to the frequencies determined by the PoT model. It is because, more priority is given to the campaigns with less future inventory values. This may result in processing certain campaigns more or less than the times determined by the PoT model.

The following section demonstrates an example to show the performance of implementing these approaches in two stochastic environments.

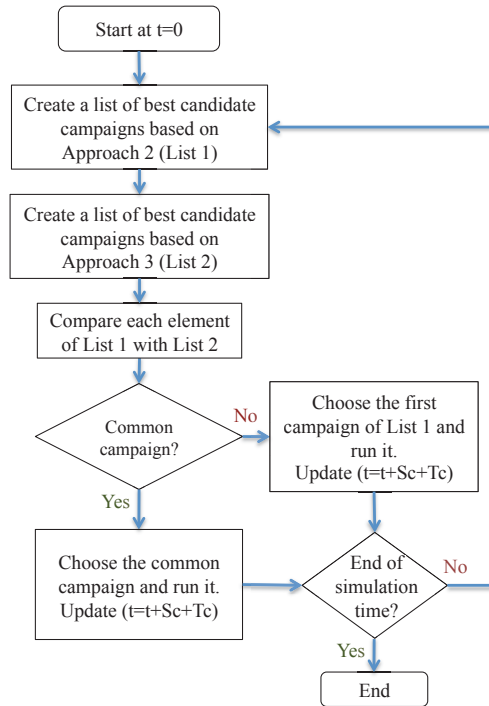


Figure 5.5: Approach 5 flowchart

## 5.7 Example

This section demonstrates the application of the mentioned approaches through two cases based on the data of Example 2 in chapter 4. In the first case, demand enters the system in small batches and high frequencies. This procedure creates near-constant demand over time, while in the second case, demands are assumed to be more random, with different arrival rates and batch sizes (“bulk” arrivals). In each case, the performances of control approaches are compared using simulation.

### 5.7.1 Data

70 products are produced from running 12 campaigns with the production time and frequencies listed in Table 4.5. To create stochastic demand, we assume that demand arrivals are based on specific rates over a year (number of arrivals per year). The time between arrivals is exponentially distributed with mean  $1/\lambda_p$  as shown in equation 5.15. It is assumed that demand sizes are uniformly distributed

within certain ranges (equation 5.12).

$$Size_p^A \sim U(D_p^{min}, D_p^{max}) \quad (5.12)$$

$$D_p^{min} = 0.75 * \frac{D_p}{Rate_p}, \quad D_p^{max} = 1.25 * \frac{D_p}{Rate_p} \quad (5.13)$$

$$Time_p^A \sim exp(\lambda_p) \quad (5.14)$$

$$\lambda_p = \frac{Total\ time}{Rate_p}, \quad Mean_p = \frac{1}{\lambda_p} \quad (5.15)$$

where  $D_p^{min}$  and  $D_p^{max}$  are the minimum and the maximum amounts for the demand size of product  $p$  at its arrival. Therefore when a demand arrives, it has a size that fits within this range. If the number of orders of product  $p$  over a year is  $Rate_p$ , the average demand size is calculated as  $\frac{D_p}{Rate_p}$  in which  $D_p$  is the total deterministic demand of product  $p$  resulting from the PoT model. The amount by which each product's demand size can vary from its mean, is assumed to be a quarter of average demand size.  $\lambda_p$  is the average time between arrival and is calculated as the total available time per year divided by the demand rate. It is assumed that the time between arrivals has exponential distribution function (equation 5.14) and the associated mean is represented by  $Mean_p$ . *Total time* is total available time per year.

Two cases are considered for demand arrivals. In the first case, demands arrive in small batches (sizes) and high frequencies (1000 times per year) to create near-constant demands. While in the second case, bulk arrivals create more stochastic demand. Tables 5.1 and 5.2 present the data of deterministic demand (the amount supplied by the PoT model), arrival rate, average time between arrival and the mean for the exponential distribution function for each product in Case 1 and Case 2, respectively. For each case, the average size of each product's demand per order is demonstrated in figures 5.6 and 5.7. As can be seen, the average time between arrivals for Case 1 is 1.82 (hour) for all the products and the order sizes are less than 722 (cubic ft). In Case 2, average time between arrivals are between

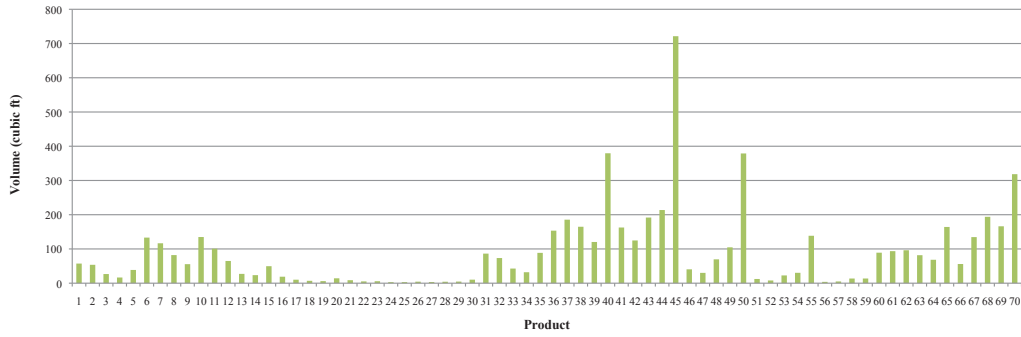
38.72 and 303.33 (hours) and the maximum average order size is 42,165 (cubic ft).

| Product | $D_p$<br>( $ft^3$ ) | $Rate_p$<br>(#/year) | $\lambda_p$<br>(hour) | $Mean_p$<br>(1/hour) | Product | $D_p$<br>( $ft^3$ ) | $Rate_p$<br>(#/year) | $\lambda_p$<br>(hour) | $Mean_p$<br>(1/hour) |
|---------|---------------------|----------------------|-----------------------|----------------------|---------|---------------------|----------------------|-----------------------|----------------------|
| 1       | 57300.1             | 1000                 | 1.82                  | 0.549                | 36      | 153438.0            | 1000                 | 1.82                  | 0.549                |
| 2       | 53713.8             | 1000                 | 1.82                  | 0.549                | 37      | 185509.8            | 1000                 | 1.82                  | 0.549                |
| 3       | 26721.9             | 1000                 | 1.82                  | 0.549                | 38      | 164844.9            | 1000                 | 1.82                  | 0.549                |
| 4       | 16616.5             | 1000                 | 1.82                  | 0.549                | 39      | 120403.1            | 1000                 | 1.82                  | 0.549                |
| 5       | 38665.5             | 1000                 | 1.82                  | 0.549                | 40      | 379480.2            | 1000                 | 1.82                  | 0.549                |
| 6       | 133152.8            | 1000                 | 1.82                  | 0.549                | 41      | 162619.8            | 1000                 | 1.82                  | 0.549                |
| 7       | 116572.5            | 1000                 | 1.82                  | 0.549                | 42      | 124790.6            | 1000                 | 1.82                  | 0.549                |
| 8       | 82081.0             | 1000                 | 1.82                  | 0.549                | 43      | 191731.7            | 1000                 | 1.82                  | 0.549                |
| 9       | 55560.6             | 1000                 | 1.82                  | 0.549                | 44      | 213670.7            | 1000                 | 1.82                  | 0.549                |
| 10      | 134831.8            | 1000                 | 1.82                  | 0.549                | 45      | 721344.9            | 1000                 | 1.82                  | 0.549                |
| 11      | 100768.6            | 1000                 | 1.82                  | 0.549                | 46      | 40472.8             | 1000                 | 1.82                  | 0.549                |
| 12      | 64906.1             | 1000                 | 1.82                  | 0.549                | 47      | 30156.3             | 1000                 | 1.82                  | 0.549                |
| 13      | 27187.0             | 1000                 | 1.82                  | 0.549                | 48      | 69652.5             | 1000                 | 1.82                  | 0.549                |
| 14      | 23478.7             | 1000                 | 1.82                  | 0.549                | 49      | 104758.7            | 1000                 | 1.82                  | 0.549                |
| 15      | 49623.5             | 1000                 | 1.82                  | 0.549                | 50      | 378920.5            | 1000                 | 1.82                  | 0.549                |
| 16      | 18938.4             | 1000                 | 1.82                  | 0.549                | 51      | 12302.2             | 1000                 | 1.82                  | 0.549                |
| 17      | 10191.5             | 1000                 | 1.82                  | 0.549                | 52      | 7918.9              | 1000                 | 1.82                  | 0.549                |
| 18      | 6955.8              | 1000                 | 1.82                  | 0.549                | 53      | 22650.0             | 1000                 | 1.82                  | 0.549                |
| 19      | 6042.5              | 1000                 | 1.82                  | 0.549                | 54      | 30350.0             | 1000                 | 1.82                  | 0.549                |
| 20      | 14248.9             | 1000                 | 1.82                  | 0.549                | 55      | 138511.8            | 1000                 | 1.82                  | 0.549                |
| 21      | 8733.3              | 1000                 | 1.82                  | 0.549                | 56      | 3873.3              | 1000                 | 1.82                  | 0.549                |
| 22      | 5225.0              | 1000                 | 1.82                  | 0.549                | 57      | 5041.3              | 1000                 | 1.82                  | 0.549                |
| 23      | 5732.4              | 1000                 | 1.82                  | 0.549                | 58      | 13606.5             | 1000                 | 1.82                  | 0.549                |
| 24      | 3022.8              | 1000                 | 1.82                  | 0.549                | 59      | 13674.3             | 1000                 | 1.82                  | 0.549                |
| 25      | 3121.8              | 1000                 | 1.82                  | 0.549                | 60      | 89097.6             | 1000                 | 1.82                  | 0.549                |
| 26      | 4430.0              | 1000                 | 1.82                  | 0.549                | 61      | 93567.2             | 1000                 | 1.82                  | 0.549                |
| 27      | 3361.3              | 1000                 | 1.82                  | 0.549                | 62      | 96203.9             | 1000                 | 1.82                  | 0.549                |
| 28      | 4248.7              | 1000                 | 1.82                  | 0.549                | 63      | 81762.5             | 1000                 | 1.82                  | 0.549                |
| 29      | 4700.5              | 1000                 | 1.82                  | 0.549                | 64      | 68409.3             | 1000                 | 1.82                  | 0.549                |
| 30      | 10293.6             | 1000                 | 1.82                  | 0.549                | 65      | 164235.0            | 1000                 | 1.82                  | 0.549                |
| 31      | 86363.2             | 1000                 | 1.82                  | 0.549                | 66      | 56057.1             | 1000                 | 1.82                  | 0.549                |
| 32      | 73446.2             | 1000                 | 1.82                  | 0.549                | 67      | 134597.3            | 1000                 | 1.82                  | 0.549                |
| 33      | 42712.4             | 1000                 | 1.82                  | 0.549                | 68      | 193888.5            | 1000                 | 1.82                  | 0.549                |
| 34      | 31998.6             | 1000                 | 1.82                  | 0.549                | 69      | 166173.7            | 1000                 | 1.82                  | 0.549                |
| 35      | 88579.7             | 1000                 | 1.82                  | 0.549                | 70      | 318409.0            | 1000                 | 1.82                  | 0.549                |

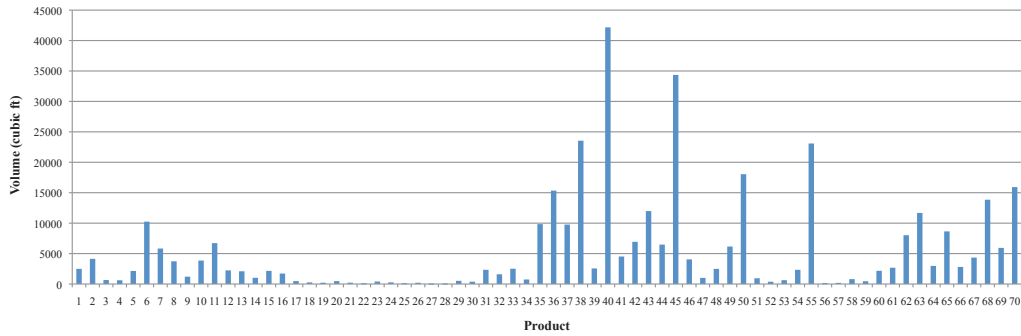
Table 5.1: Case 1: Basic data of stochastic demand generation process

| Product | $D_p$<br>( $ft^3$ ) | $Rate_p$<br>(#/year) | $\lambda_p$<br>(hour) | $Mean_p$<br>(1/hour) | Product | $D_p$<br>( $ft^3$ ) | $Rate_p$<br>(#/year) | $\lambda_p$<br>(hour) | $Mean_p$<br>(1/hour) |
|---------|---------------------|----------------------|-----------------------|----------------------|---------|---------------------|----------------------|-----------------------|----------------------|
| 1       | 57300.1             | 23                   | 79.13                 | 0.013                | 36      | 153438.0            | 10                   | 182.00                | 0.005                |
| 2       | 53713.8             | 13                   | 140.00                | 0.007                | 37      | 185509.8            | 19                   | 95.79                 | 0.010                |
| 3       | 26721.9             | 41                   | 44.39                 | 0.023                | 38      | 164844.9            | 7                    | 260.00                | 0.004                |
| 4       | 16616.5             | 27                   | 67.41                 | 0.015                | 39      | 120403.1            | 47                   | 38.72                 | 0.026                |
| 5       | 38665.5             | 18                   | 101.11                | 0.010                | 40      | 379480.2            | 9                    | 202.22                | 0.005                |
| 6       | 133152.8            | 13                   | 140.00                | 0.007                | 41      | 162619.8            | 36                   | 50.56                 | 0.020                |
| 7       | 116572.5            | 20                   | 91.00                 | 0.011                | 42      | 124790.6            | 18                   | 101.11                | 0.010                |
| 8       | 82081.0             | 22                   | 82.73                 | 0.012                | 43      | 191731.7            | 16                   | 113.75                | 0.009                |
| 9       | 55560.6             | 46                   | 39.57                 | 0.025                | 44      | 213670.7            | 33                   | 55.15                 | 0.018                |
| 10      | 134831.8            | 35                   | 52.00                 | 0.019                | 45      | 721344.9            | 21                   | 86.67                 | 0.012                |
| 11      | 100768.6            | 15                   | 121.33                | 0.008                | 46      | 40472.8             | 10                   | 182.00                | 0.005                |
| 12      | 64906.1             | 29                   | 62.76                 | 0.016                | 47      | 30156.3             | 30                   | 60.67                 | 0.016                |
| 13      | 27187.0             | 13                   | 140.00                | 0.007                | 48      | 69652.5             | 28                   | 65.00                 | 0.015                |
| 14      | 23478.7             | 23                   | 79.13                 | 0.013                | 49      | 104758.7            | 17                   | 107.06                | 0.009                |
| 15      | 49623.5             | 23                   | 79.13                 | 0.013                | 50      | 378920.5            | 21                   | 86.67                 | 0.012                |
| 16      | 18938.4             | 11                   | 165.45                | 0.006                | 51      | 12302.2             | 13                   | 140.00                | 0.007                |
| 17      | 10191.5             | 21                   | 86.67                 | 0.012                | 52      | 7918.9              | 21                   | 86.67                 | 0.012                |
| 18      | 6955.8              | 26                   | 70.00                 | 0.014                | 53      | 22650.0             | 36                   | 50.56                 | 0.020                |
| 19      | 6042.5              | 28                   | 65.00                 | 0.015                | 54      | 30350.0             | 13                   | 140.00                | 0.007                |
| 20      | 14248.9             | 30                   | 60.67                 | 0.016                | 55      | 138511.8            | 6                    | 303.33                | 0.003                |
| 21      | 8733.3              | 39                   | 46.67                 | 0.021                | 56      | 3873.3              | 26                   | 70.00                 | 0.014                |
| 22      | 5225.0              | 41                   | 44.39                 | 0.023                | 57      | 5041.3              | 27                   | 67.41                 | 0.015                |
| 23      | 5732.4              | 14                   | 130.00                | 0.008                | 58      | 13606.5             | 17                   | 107.06                | 0.009                |
| 24      | 3022.8              | 11                   | 165.45                | 0.006                | 59      | 13674.3             | 30                   | 60.67                 | 0.016                |
| 25      | 3121.8              | 22                   | 82.73                 | 0.012                | 60      | 89097.6             | 41                   | 44.39                 | 0.023                |
| 26      | 4430.0              | 21                   | 86.67                 | 0.012                | 61      | 93567.2             | 35                   | 52.00                 | 0.019                |
| 27      | 3361.3              | 46                   | 39.57                 | 0.025                | 62      | 96203.9             | 12                   | 151.67                | 0.007                |
| 28      | 4248.7              | 40                   | 45.50                 | 0.022                | 63      | 81762.5             | 7                    | 260.00                | 0.004                |
| 29      | 4700.5              | 9                    | 202.22                | 0.005                | 64      | 68409.3             | 23                   | 79.13                 | 0.013                |
| 30      | 10293.6             | 27                   | 67.41                 | 0.015                | 65      | 164235.0            | 19                   | 95.79                 | 0.010                |
| 31      | 86363.2             | 37                   | 49.19                 | 0.020                | 66      | 56057.1             | 20                   | 91.00                 | 0.011                |
| 32      | 73446.2             | 46                   | 39.57                 | 0.025                | 67      | 134597.3            | 31                   | 58.71                 | 0.017                |
| 33      | 42712.4             | 17                   | 107.06                | 0.009                | 68      | 193888.5            | 14                   | 130.00                | 0.008                |
| 34      | 31998.6             | 43                   | 42.33                 | 0.024                | 69      | 166173.7            | 28                   | 65.00                 | 0.015                |
| 35      | 88579.7             | 9                    | 202.22                | 0.005                | 70      | 318409.0            | 20                   | 91.00                 | 0.011                |

Table 5.2: Case 2: Basic data of stochastic demand generation process



**Figure 5.6:** Case 1: Average demand size per order ( $\frac{D_p}{Rate_p}$ )



**Figure 5.7:** Case 2: Average demand size per order ( $\frac{D_p}{Rate_p}$ )

For simplicity, We present the inventory behaviour of just 7 products (products 10, 20, 30, 40, 50, 60 and 70). We use a lead time window of 4 weeks (“m” equals 4 weeks or 140 hours). As mentioned earlier, we assume actual demand inter-arrival times and sizes are generated based on exponential and uniform distribution functions respectively.

The forecasting procedure is based on equations 5.2 and 5.3 in which  $\alpha = 0.4$ . The procedure we used to simulate our demand is that once a demand arrival time and size is available (within  $m=4$  weeks), the next forecasted demand arrival time and size are triggered. At the end of each production run, new demands and new forecasts are available. Initial inventory of each product is assumed to be zero.

### 5.7.2 Results and Discussion

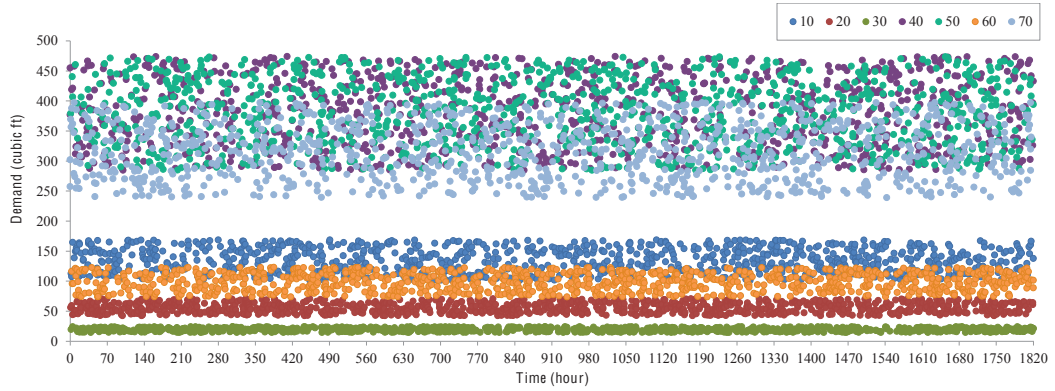
For each case (Case 1 and Case 2), we implement simulation over 4 years. In each case, stochastic demand is generated and fed into the system and all approaches are examined. For comparison purposes, all approaches use the same demand data during a simulation run. To demonstrate the behaviour of the system, figures in this section present the data and results related to just 7 products (10, 20, 30, 40, 50, 60 and 70) over a simulation run.

Figures 5.8 and 5.9 illustrate the demands of the mentioned products over the first year of the simulation for Case 1 and Case 2, respectively. As mentioned earlier, demands in Case 1 are more frequent but with a correspondingly smaller size than Case 2. Thus, the difference between realized demand and the average demand for each product in a given time interval is less in Case 1. This will result in obtaining stochastic demand that is near-constant and more consistent with the demand obtained from the PoT model. Case 1 is used to examine the control approaches when the system faces near-constant demand arrivals.

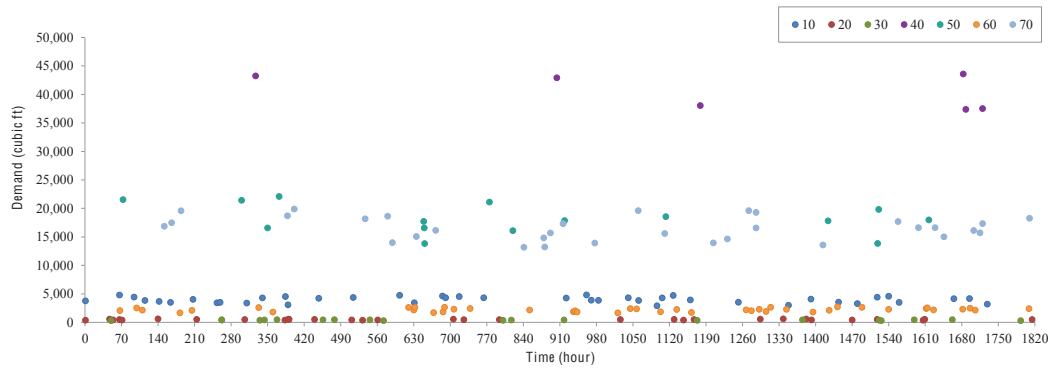
As opposed to Case 1, Case 2 represents a stochastic situation in which orders arrive with different rates and more varying sizes (bulk arrivals). Case 2 poses more difficulties because with bulk arrivals, the difference between realized demand and its expected value increases. For example, in figure 5.9 we expected the demand of product 40 to happen over 9 orders (Table 5.2) while we see 6 orders happened in the actual process. These differences, however are unavoidable when we deal with the stochastic environment.

Through the simulation, we can monitor the inventory status of each product by using different control approaches. All the figures in this section present the inventory of 7 particular products (10, 20, 30, 40, 50, 60, 70) over 4 years of one simulation run.





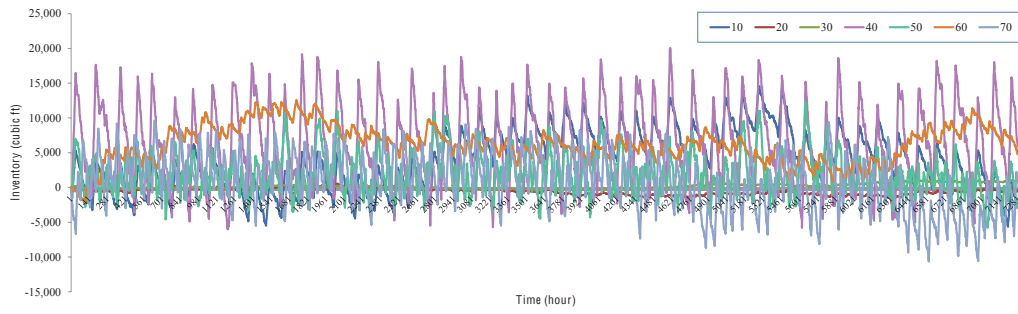
**Figure 5.8:** Case 1: Demands of 7 products over the first year



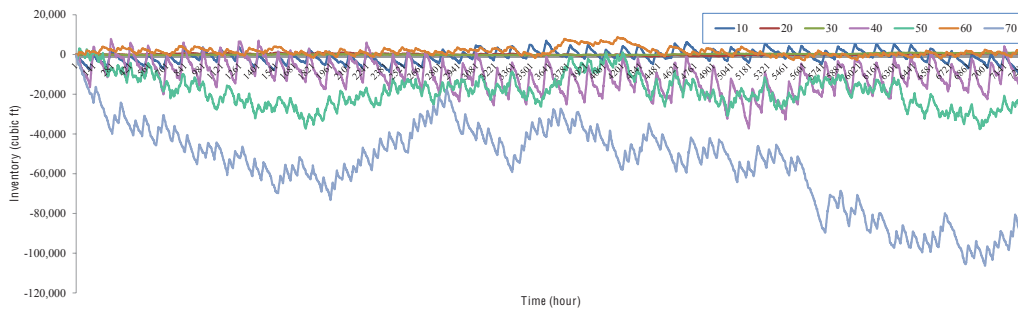
**Figure 5.9:** Case 2: Demands of 7 products over the first year

Figures 5.10- 5.14 show the inventory of each of the 7 products using Approach 1 to Approach 5 for Case 1.

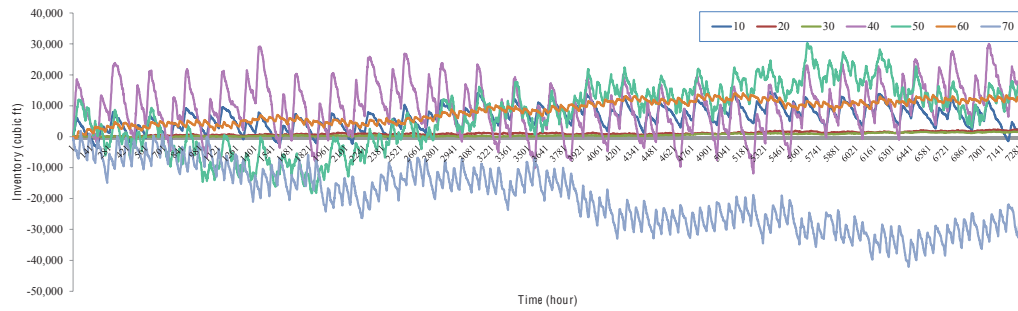
Figure 5.15 shows the schedules created by each approach for the first 16 weeks of the simulation in Case 1.



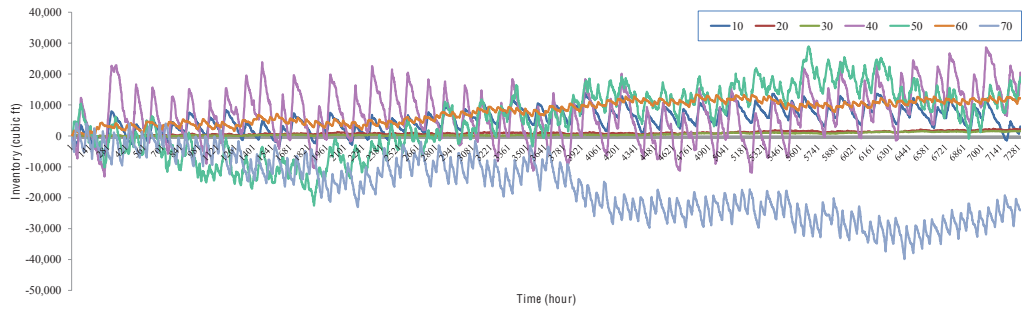
**Figure 5.10:** Case 1: Inventory status of 7 products using Approach 1



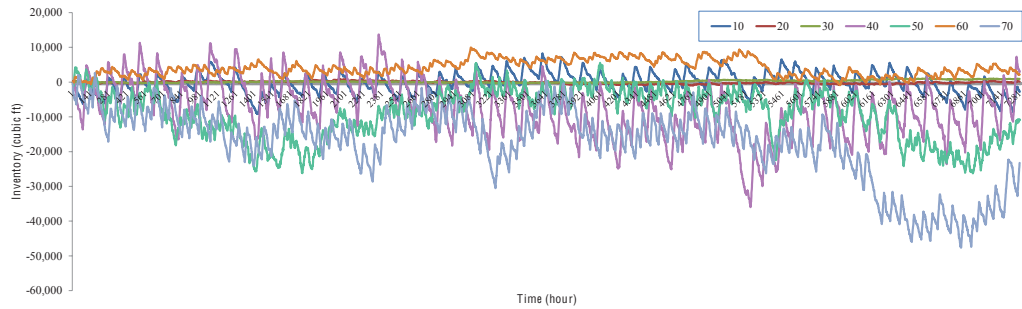
**Figure 5.11:** Case 1: Inventory status of 7 products using Approach 2



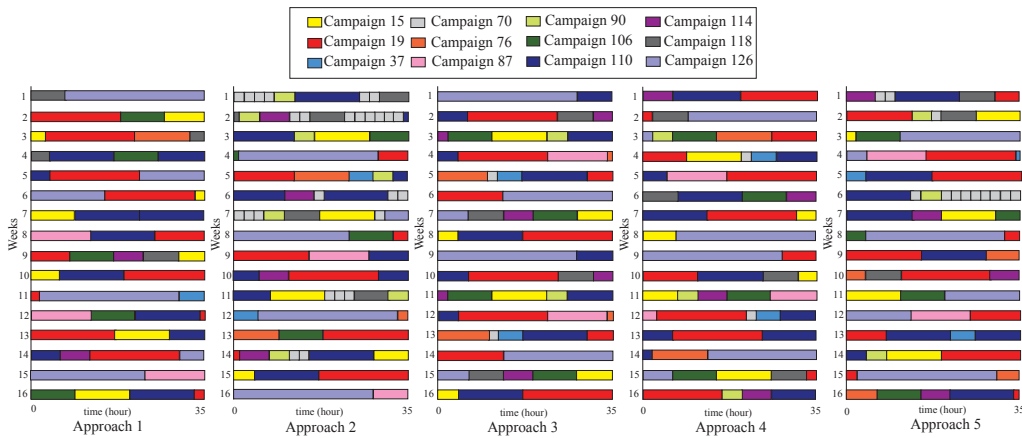
**Figure 5.12:** Case 1: Inventory status of 7 products using Approach 3



**Figure 5.13:** Case 1: Inventory status of 7 products using Approach 4



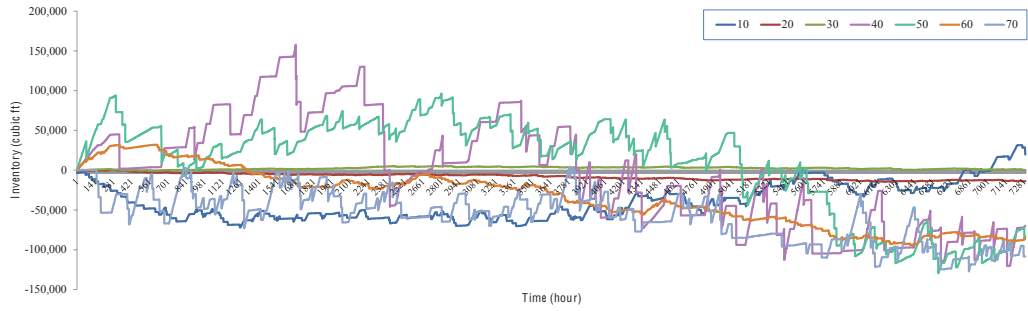
**Figure 5.14:** Case 1: Inventory status of 7 products using Approach 5



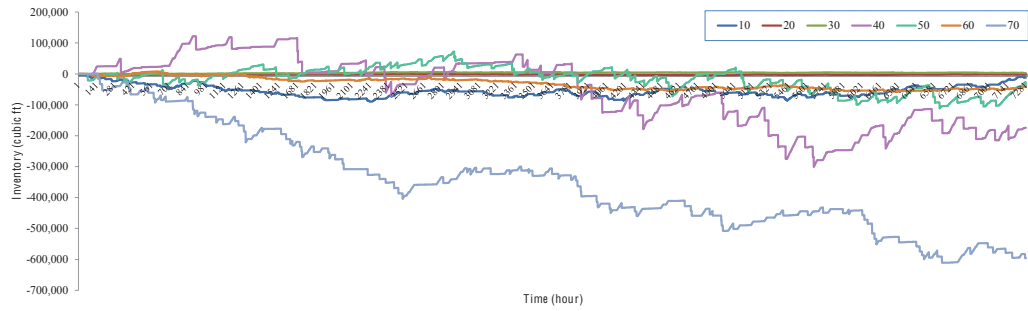
**Figure 5.15:** Case 1: Weekly campaign schedules for the first 16 weeks of 5 Approaches

Figures 5.16- 5.20 demonstrate the inventory of each of the 7 products using Approach 1 to Approach 5 over 4 years of a simulation run for Case 2.

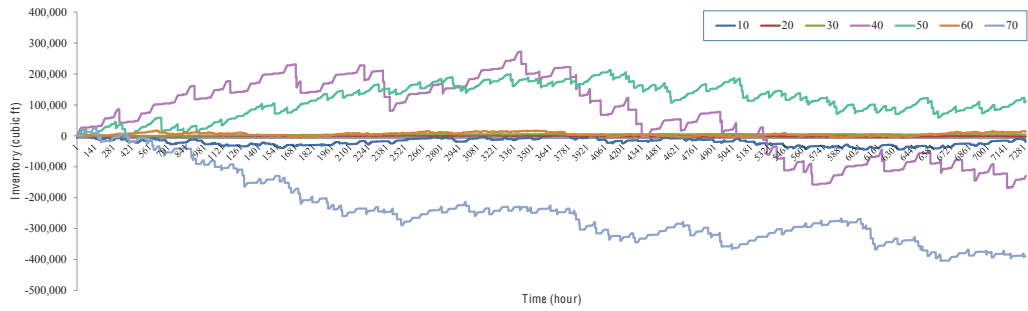
Figure 5.21 shows the schedules created from each approach for the first 16 weeks of the simulation in Case 2.



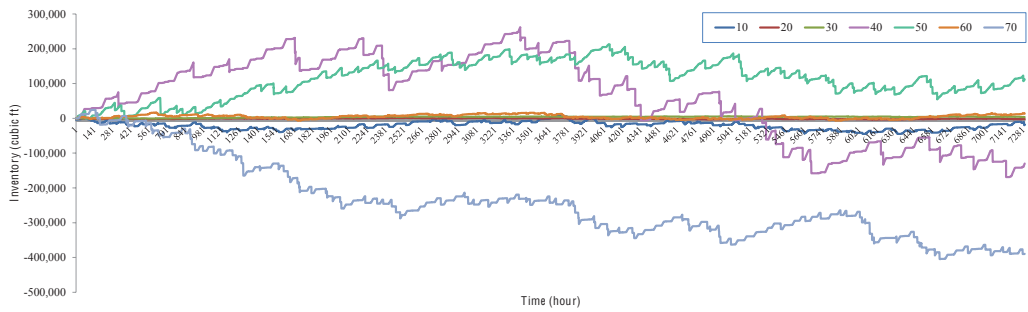
**Figure 5.16:** Case 2: Inventory status of 7 products using Approach 1



**Figure 5.17:** Case 2: Inventory status of 7 products using Approach 2



**Figure 5.18:** Case 2: Inventory status of 7 products using Approach 3



**Figure 5.19:** Case 2: Inventory status of 7 products using Approach 4

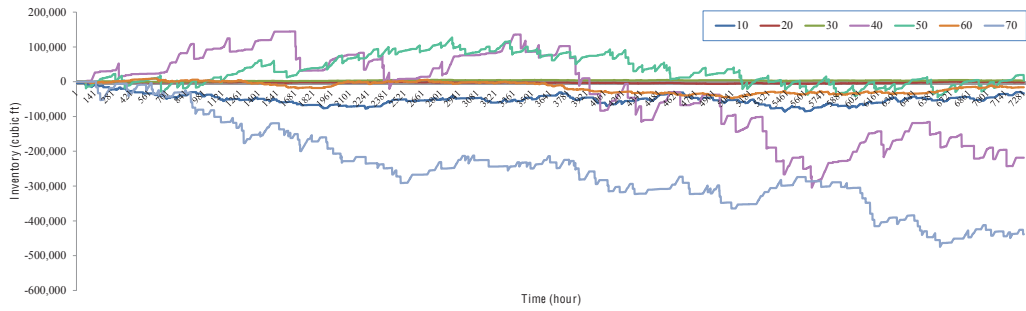


Figure 5.20: Case 2: Inventory status of 7 products using Approach 5

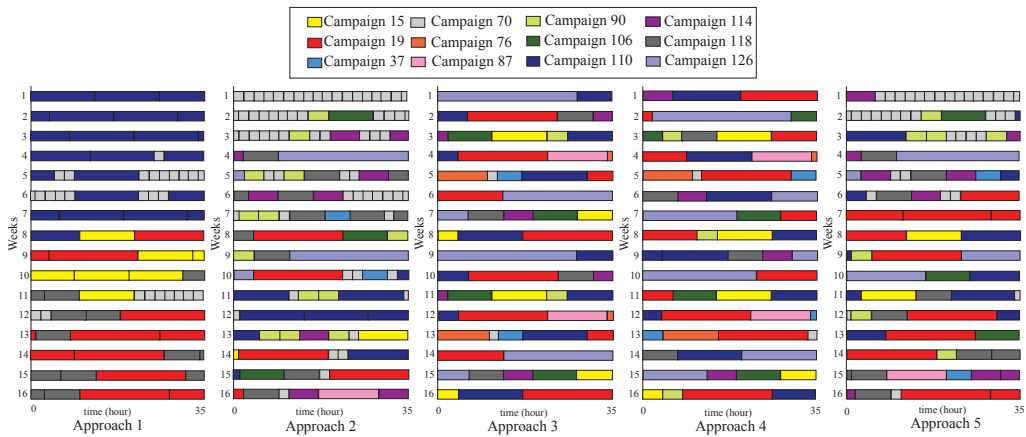


Figure 5.21: Case 2: Weekly campaign schedules for the first 16 weeks of 5 Approaches

Figures 5.22 and 5.23 show the number of runs of each campaign over the first year simulation in Case 1 and Case 2. Each figure compares the results of each approach with the number of runs proposed by the PoT model.

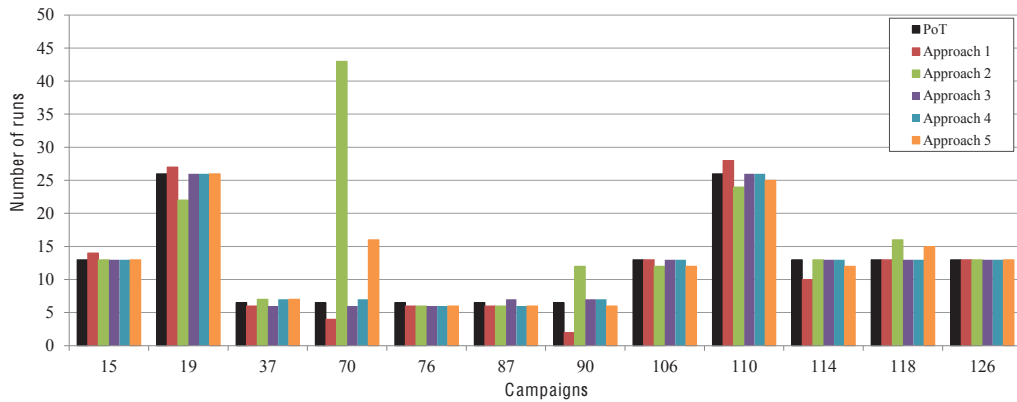
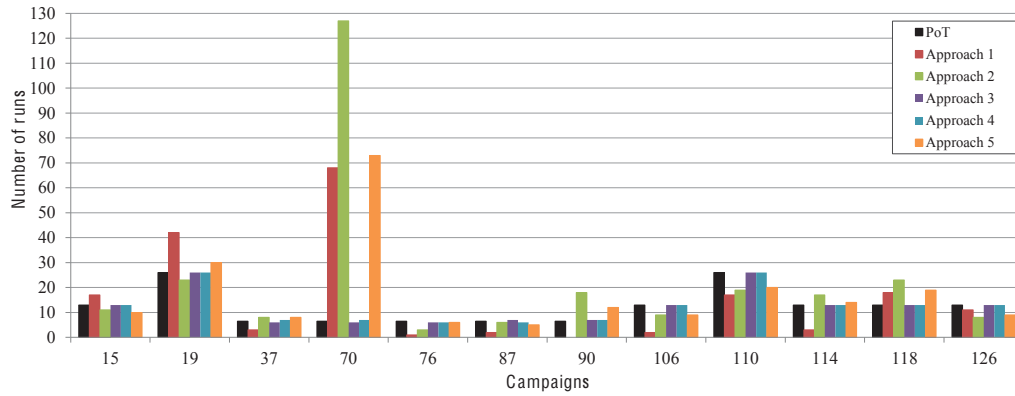


Figure 5.22: Case 1: Number of runs of each campaign over the first year of simulation



**Figure 5.23:** Case 2: Number of runs of each campaign over the first year of simulation

Comparing two figures 5.10 and 5.16, we see when demand arrivals have high frequencies (Case 1), Approach 1 can better control the inventory levels, so that we do not see very large positive or negative inventories lasting for long periods over the simulation time in Figure 5.10. As orders arrive with short inter-arrival times and small sizes in Case 1, successive runs of a certain campaign to fulfill the demand of a specific product is rarely required (Approach 1 of Figure 5.15). In this case, the number of campaign runs over a year is close to the ones determined by the PoT problem. This can be seen by comparing black and red bars in Figure 5.22. However, in Case 2, where demands have bulk arrivals (less frequent with large order size), we usually need successive runs of a specific campaign to fill the demand of a required product. This poses difficulties, because long runs of a chosen campaign not only produces the most required product (with minimum runout time) but it may simultaneously create significant amounts of other products which already have high inventory levels. For example, Figure 5.21 illustrates successive runs of campaign 110 over the first three weeks. Referring to Appendix I, this campaign creates large inventories of products 45, 50, 55, 60 and 40 at the same time. Thus, Approach 1 can perform more efficiently in the case of near-constant demand arrivals.

Figures 5.11 and 5.17 provide the results of Approach 2 for Case 1 and Case 2, respectively. In both cases, Approach 2 creates negative inventory of few products

such as products 70. Referring to Figures 5.15 and 5.21, we see successive runs of campaign 70. Moreover, the number of times that campaign 70 has been processed over a year (i.e. 43 times in Case 1, and 127 times in Case 2) is much greater than the one imposed by the PoT model (i.e. 6.5 times)(Figures 5.22 and 5.23). Referring to Section 5.3, the selection criterion of Approach 2 is the campaign with minimum total future inventory cost at the end of its production run. The reason of having many runs of campaign 70 is that this campaign has the shortest production time among other campaigns (i.e. 1.54 hours). This short production time imposes less production and most of the time less excess inventory of unnecessary products at the end of the campaign's production time. Thus the chance of choosing campaign 70 is high at the decision making time as it usually creates less inventory cost. However, this campaign does not produce some products such as 50 and 70. Considering zero initial inventory for these products, processing campaign 70 for a large number of times, results in more negative inventory of such products. Although this approach seems to produce large positive or negative inventory levels for some products, there are only 3 products in Case 1, with large positive and negative inventories over all 70 products during 4 years. In Case 2, the number of products with uncontrolled inventory levels is 6.

Approach 3 only uses the required campaign frequencies (number of times a campaign is processed) determined by the PoT model, as its campaign selection criterion, regardless of the inventory positions or order arrivals. As with the other approaches, every time a campaign is chosen, it is processed for the amount of time resulted from the PoT model.

Figures 5.12 and 5.18 show the inventory behaviour of the 7 products when Approach 3 is applied to Case 1 and Case 2. As can be seen, in the near-constant case (Case 1), the inventories of some products gradually increase (e.g. product 60) or decrease (e.g. product 70). However, in Case 2, we see large positive and negative inventories lasting for long periods of time.

In both cases, a part of this problem arises from the stochastic nature of the de-

mand arrivals which causes the realized demand to be different than the expected value (deterministic demand). For instance, in Case 1, the annual deterministic demand of product 60 over the first year is 89,097 (cubic feet). However the total realized demand over this period is 86,912 (cubic feet). The actual demand is 2,185 (cubic feet) less than what is expected. Thus using the same PoT requirements create excess inventory of this product over the first year. On the other side, the expected demand of product 70 in Case 1, over the first year is 318,409 (cubic feet), but the total actual demand ends up with 324,372 (cubic feet). The 5,606 (cubic feet) difference, results in negative inventory of this product when we use the same campaigns and frequencies imposed the PoT model.

When demand has bulk arrivals (Case 2), the difference between actual demand and its expected value increases. This results in large positive and negative inventories of some products when Approach 3 is used. For example, the total actual demand of product 40 is 182,642 (cubic feet) less than its deterministic value in the first 2 years. As shown in figure 5.18, this causes large positive inventory of this product over this period. However, the total actual demand over the third year is 309,635 (cubic feet) greater than the deterministic demand of the PoT. This results in descending trend of the inventory of this product. Since Approach 3 does not violate the number of campaign runs determined by the PoT results, it can not effectively control the process in the case of bulk demand arrival (Case 2).

Another issue underlying Approach 3 is the initial inventory. We always consider zero initial inventory and constant demand for the PoT model. However, due to the stochastic nature of demand, we may end up with positive or negative inventories at the end of a year. If we do not adjust the new initial inventories (which in this case we did not) for the new horizon, Approach 3, can not do it either.

The resulting schedules of Approach 3 are shown in figures 5.15 and 5.21. The number of each campaign run is based on the PoT campaign frequency require-



ments (see figure 5.23). For instance, campaign 19 (red) is processed every 2 weeks (26 times per year) and campaign 90 (light green) is run every 8 weeks (6.5 times per year). Thus, the number of campaign runs are the same as the PoT model (figures 5.22 and 5.23).

Approach 4 (shown in figures 5.13 for Case 1 and 5.19 for Case 2), as a combination of Approaches 2 and 3 with more emphasis on Approach 3, creates very similar graphs and schedules to Approach 3. In figures 5.15 and 5.21 we see that only the order of some campaigns in Approach 4 is different from Approach 3. The two problems including the stochastic nature of demand and the initial inventory (as mentioned for Approach 3) exist here because of the assumption underlying this approach that forces to process campaigns for the same number of times as determined by the PoT model. Figures 5.22 and 5.23 imply that Approach 4 has the same number of campaign runs as the PoT model.

Figures 5.14 and 5.20 present the results of implementing Approach 5 in which the two criteria of Approaches 2 and 3 are combined with more focus on Approach 2 (the campaign with minimum future inventory cost). The inventory behaviour is similar to the situation when Approach 2 is applied. However, in both cases, this approach can improve the inventory level of some products such as 70 (from -100,000 to -50,000 (cubic feet) in Case 1 and from -600,000 to -500,000 (cubic feet) in Case 2).

Similar to Approach 2, long runs of a specific campaign is possible (such as campaign 70) but the number of each campaign run is influenced by the required frequencies posed by the PoT model. Although, at the end of the year, the number of each campaign run is different than the the PoT's, it will cause more diverse runs than Approach 2 (figures 5.22 and 5.23). For example the number of runs of campaign 70 in Case 2, is reduced from 127 in Approach 2 to 73 in this approach.

The main objective was to examine how the campaign lot sizes developed through the powers of two model might perform. In order to do so, we had to develop some concept of how we might actually implement the PoT lot sizes. The

5 approaches discussed here are some examples. However, what rapidly becomes evident is that even if the average demand remains constant, the stochastic process behind it matters. If the demand arrives as a poisson process with a high arrival rate but small demand size and variance at each arrival, then variance of total demand over time intervals such as a month can be small enough to be regarded as effectively constant. However, if the same average demand arrives as a poisson process with a slow arrival rate and a large mean demand size and variance for each arrival, then the actual demand in a period can be quite different from the expected demand. Over that period the lot sizes and frequencies computed by the powers of two model will not perform well.

The PoT model does not consider initial inventory at the beginning of the time horizon. Thus if a product ends up with large positive or negative inventory at the end of a year, it poses difficulties in the next year's plan. From our simulations, it is obvious that such a situation is entirely possible, even given the constant average demand that we have used. One possible approach might be to take the initial excess (shortage) of inventory and subtract (add) it to the anticipated annual demand for the coming year and re-calculate campaign lot sizes. However it is dealt with, it appears that a process of periodic re-planning is necessary.

# Chapter 6

## Conclusion

To better control the lumber production process through inventory minimization, a 3-stage control mechanism was proposed.

At the first stage, a quick model was designed which would work as a simple lumber value optimizer. This algorithm creates a large number of cutting patterns once, evaluates the potential patterns for each log within a class of logs, and chooses the pattern which would create an optimal breakdown and thus an optimal value yield on each log based on a specified price list. A combination of a log class and a price list defines a set of lumber outputs in the form of a “campaign”. Thus lumber output proportions per unit volume of a campaign can be quickly calculated through this process. The campaign concept gives a useful way of dealing with the joint production process.

Although, in this study we generated 126 campaigns, this efficient model, coded in Python, is capable of creating a wide range of campaigns through inputting new classes of logs and price lists. We showed simple ways of generating price lists in Chapter 3. For the most part these are adequate in allowing us to meet demand. As we will discuss, there are other options that we have not yet experimented with.

The second stage involves planning the campaigns in a sawmill. Due to the divergent production nature of the sawing process, we needed an algorithm to ex-

plot lot-scheduling principles. In the classical setting, the economic lot-scheduling takes into account holding costs and setup times to find a production strategy that minimizes inventory while respecting capacity constraints. We used a Powers-of-Two approach implemented as an MIP with the novelty of our approach being that, instead of focusing on product lot sizes, we focus on campaign lot sizes. This is appropriate for a sawmill because setup times occur for campaigns, not products and because it is impossible to separate the production of individual products.

The model was developed to provide an assessment of the inventory levels required to deal with a given set of annual demands in the simplest case when demands are constant. The model uses the campaigns developed in Chapter 3 and operates at a realistic scale with a reasonably large number of potential campaigns and a complex product mix of 70 products. The model allows the calculation of campaign lot sizes and the frequency with which campaigns would be run.

The campaigns that we defined and used appear to be adequate. However it is possible to generate more campaigns in different ways. One is by using different price lists. One way to obtain such prices is to solve the model as a linear program without insisting on integer solutions. Then the shadow prices (see Williams [53] on the demand constraints, Chapter 4, equations 4.5) provide a price list that could be used in the campaign generation procedure of Chapter 3. An alternative way is to first solve the MIP model of chapter 4 for an optimal or approximately optimal solution. If the binary variables resulting from this solution are fixed and the model solved again as a linear program, once again, shadow prices will be available on the constraint 4.5 which can be used as price list in campaign generation. It would be useful as further research to see if improved campaign generation procedures would affect inventory levels.

The final stage speaks to scheduling campaigns and dealing with the stochastic nature of demand. The Powers-of-Two model calculates campaign frequencies and production levels. It does not address scheduling, although the history of powers of two approaches, discussed in Chapter 2, suggest that scheduling is facilitated

by a powers of two lot size. We did not look scheduling specifics in this section. Instead, we were interested in how the scheduling might respond to different types of stochastic demand. Through the PoT model, appropriate campaigns, associated frequencies and running durations were available. Using a simulation environment, we created five control approaches that would use the outputs of the PoT model to examine the performance of the PoT model and schedule campaigns over short time periods in the case of stochastic demand.

The stochastic demand was simulated through two cases. In the first case, demand had high arrival frequencies and small demand batch sizes. This simulated a near-constant demand. In the second case, demand had bulk arrivals with varying sizes. Through the simulation, the performances of these approaches were evaluated in the cases. The PoT based control approaches were reasonably effective, when demand was near constant. Even here, there were some problems due to the initial inventories not being as required and due to the demand realizations not corresponding exactly to the expected constant demand levels. Bulk arrivals of demand posed even more difficulties in the control processes. A part of these difficulties was caused by the stochastic nature of demand arrivals. When demand has bulk arrivals, the difference between realized demand and its expected value can be quite substantial over quite long periods of time. Thus, using the requirements imposed by the PoT model did not control the inventory levels efficiently.

A traditional lean manufacturing approach to a capacity constrained manufacturing facility is the use of a cyclic EPEI production strategy. Even with constant demands and individually produced products, it isn't clear that an EPEI strategy makes sense because the demand for and value of specific products can be quite variable. In this thesis we showed how to create campaigns to deal with the joint production of many products. Using the campaign concept, the PoT model developed in Chapter 4 allows us to calculate lot sizes and frequencies for each product that can meet annual demands at minimal cycle inventory. Even with the limited set of price lists we used in our campaign generation, we were

able to come quite close to meeting demand for a large number of products using appropriate lot sizes for a fairly small number of campaigns. In Chapter 5, we investigated if these campaigns can be implemented in an automatic fashion that can cope with demand that is constant in expected value but with significant variance. The simulations indicate that such an automatic pull process is not likely to work well. This implies that a more activist production planning process with frequent re-planning is necessary. We have not carried out such a process in this thesis. An example of such a process can be found in Saadatyar [41].

There are many different possible areas of future research suggested by this thesis.

Some of these involve looking at more and different types of campaigns. One issue is multi-species issues. For example if we process both spruce and pine logs then a given piece size in two different species may be two different products. At one level, this makes little mathematical difference to the models of chapter 4. However, when running a pine campaign, no spruce products are produced. This means that planning and scheduling these campaigns may be more important. Another involves different sorting strategies. We looked at sorting by length in Chapter 3. It is also possible to sort by diameter. This may give greater control of the size profile of the products. Finally, we can develop new price lists in order to develop more appropriate campaigns. As discussed above one way to do this is through the use of shadow prices.

A question we have not looked at is log buying strategies. Finished products are not the only inventory in the sawmill. If we know what campaigns we are running and when, this suggests which logs need to be bought and the directions given to the harvesters in terms of log lengths and diameters. How campaign processing affects log procurement and conversely how log procurement affects campaign production needs study.

The main issue raised by this thesis is the nature of production planning. Unless demand for products are constant then a pure pull process is unlikely to

work for the reasons that we have seen above. Saadatyar [41] may provide some answers.

# Bibliography

- [1] Hardware ACE. Lumber estimating price guide. <http://www.acehardware.net/estimate/>, 2011.
- [2] P. P. Alvarez and J. R. Vera. Application of robust optimization to the sawmill planning problem. *Annals of Operations Research*, pages 1–19, 2011.
- [3] S. M. Bhandarkar, X. Luo, R. F. Daniels, and E. W. Tollner. Automated planning and optimization of lumber production using machine vision and computed tomography. *Automation Science and Engineering, IEEE Transactions on*, 5(4):677–695, 2008.
- [4] J. Bicheno, M. Holweg, and J. Niessmann. Constraint batch sizing in a lean environment. *International Journal of Production Economics*, 73(1):41 – 49, 2001.
- [5] R. Björheden. Raw material procurement in sawmills’ business level strategy—a contingency perspective. *International Journal of Forest Engineering*, 16(2), 2005.
- [6] E. E. Bomberger. A dynamic programming approach to a lot size scheduling problem. *Management Science*, 12(11):778–784, 1966.
- [7] C. Carnieri and G. A. Mendoza. A fractional algorithm for optimal cutting of lumber into dimension parts. *Annals of Operations Research*, 95(1):83–92, 2000.



- [8] C. Carnieri, G. A. Mendoza, and W. G. Luppold. Optimal cutting of dimension parts from lumber with a defect: a heuristic solution procedure. *Forest Products Journal*, 43(9):66–72, 1993.
- [9] C. Carnieri, G. A. Mendoza, and L. G. Gavinho. Solution procedures for cutting lumber into furniture parts. *European Journal of Operational Research*, 73(3):495–501, 1994.
- [10] F. F. Cid-Yanez, J. M. Frayret, and F. Léger. Evaluation of push and pull strategies in lumber production: An agent-based approach. *International Journal of Production Research*, 47(22):6295–6319.
- [11] F. F. Cid-Yanez, J. M. Frayret, A. Rousseau, and F. Leger. Simulation of lumber production planning using software agents: a case study. In *51st International Convention of Society of Wood Science and Technology*, Chile, Concepcin, 2008.
- [12] D. F. Cook and M. L. Wolfe. Genetic algorithm approach to a lumber cutting optimization problem. *Cybernetics and Systems*, 22(3):357–365, 1991.
- [13] S. Eilon. Scheduling for batch production. *Journal of Institute of Production Engineering*, 36(9):549–570, 1957.
- [14] Salah E. Elmaghraby. The economic lot scheduling problem (elsp): Review and extensions. *Management Science*, 24(6):pp. 587–598, 1978. ISSN 00251909.
- [15] A. Federgruen and Y. S. Zheng. Optimal power-of-two replenishment strategies in capacitated general production/ distribution networks. *Management Science*, 39(6), 1993.
- [16] Jan C. Fransoo and Werner G. M. M. Rutten. A typology of production control situations in process industries. *International Journal of Operations and Production Management*, 14(12):47–57, January 1994. ISSN 0144-3577.

- [17] J. Gaudreault, J. M. Frayret, A. Rousseau, and S. D'Amours. Combined planning and scheduling in a divergent production system with a co-production. Technical report, Working Paper, 2008.
- [18] J. Gaudreault, P. Forget, J. M. Frayret, A. Rousseau, and S. D'Amours. Distributed operations planning in the lumber supply chain: Models and coordination. Technical report, CIRRELT Working Paper CIRRELT-2009, 2009.
- [19] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, pages 849–859, 1961.
- [20] R. W. Haessler. An improved extended basic period procedure for solving the economic lot scheduling problem. *AIIE transactions*, 11(4):336–340, 1979.
- [21] H. Hallock, A. R. Stern, and D. W. Lewis. Is there a 'Best' Sawing method. Technical report, DTIC Document, 1976.
- [22] F. Hanssmann. *Operations research in production and inventory control*. Wiley, 1962.
- [23] Harry Freeman and Son Ltd. Nominal, target and actual dimensions. Supplied by e-mail to Eldon Gunn. Personal communication by Richard Freeman, 2012.
- [24] S. S. How, H. S. Sik, and I. Ahmad. Review on six types of log cutting methods in various applications: part 1. *Forest Research Institute Malaysia*, (45), 2007.
- [25] Gurobi Optimization Inc. Gurobi optimizer reference manual. <http://www.gurobi.com>, 2012.
- [26] P. Jackson, W. Maxwell, and J. Muckstadt. The joint replenishment problem with a powers-of-two restriction. *IIE transactions*, 17(1):25–32, 1985.

- [27] M. Kazemi Zanjani, D. Ait-Kadi, and M. Nourelfath. Robust production planning in a manufacturing environment with random yield: A case in sawmill production planning. *European Journal of Operational Research*, 201(3):882–891, 2010.
- [28] B. Kerber and B.J. Dreckshage. *Lean supply chain management essentials: A framework for materials managers*. CRC Press, 2011.
- [29] D. W. Lewis. *Sawmill simulation and the best opening face system: a user's guide*. United States Department of Agriculture, 1985.
- [30] A. Makhorin. GLPK GNU linear programming kit. <http://www.gnu.org/software/glpk>, 2008.
- [31] Thomas C. Maness and Scott E. Norton. Multiple period combined optimization approach to forest production planning. *Scandinavian Journal of Forest Research*, 17(5):460–471, 2002.
- [32] S. Maturana, E. Pizani, and J. Vera. Scheduling production for a sawmill: A comparison of a mathematical model versus a heuristic. *Computers and Industrial Engineering*, 59(4):667–674, 2010.
- [33] W.L. Maxwell and J.A. Muckstadt. Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research*, 33(6):1316–1341, 1985.
- [34] G. A. Mendoza, R. J. Meimban, W. G. Luppold, and P.A. Araman. Combining simulation and optimization models for hardwood lumber production. In *Proceedings: The 1991 SAP National Convention*, pages 4–7, 1991.
- [35] M. A. Narro Lopez and B. G. Kingsman. The economic lot scheduling problem: theory and practice. *International Journal of Production Economics*, 23(1):147–164, 1991.

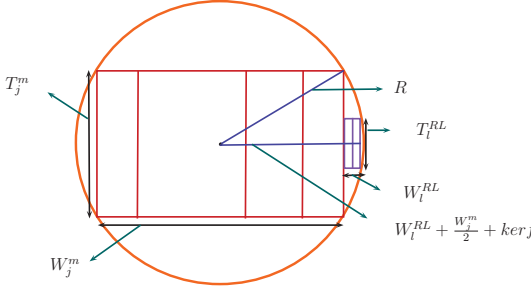
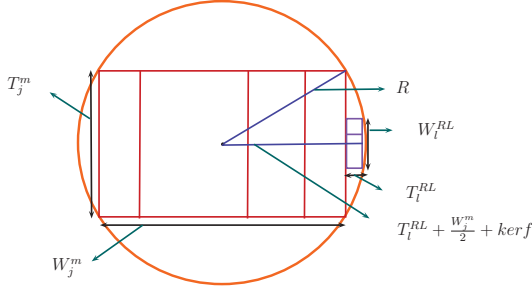
- [36] C. D. Ray, X. Zuo, J. H. Michael, and J. K. Wiedenbeck. The lean index: Operational “lean” metrics for the wood products industry. *Wood and Fiber Science*, 38(2):238–255, 2006.
- [37] M. P. Reinders and T. H. B. Hendriks. Lumberproduction optimization. *European Journal of Operational Research*, 42(3):243–253, 1989.
- [38] A. Rinnhofer, A. Petutschnigg, and J. P. Andreu. Internal log scanning for optimizing breakdown. *Computers and Electronics in Agriculture*, 41(1):7–21, 2003.
- [39] J. Rogers. A computational approach to the economic lot scheduling problem. *Management Science*, 4(3):264–291, 1958.
- [40] G. Rossum. *Python reference manual*. CWI (Centre for Mathematics and Computer Science), 1995.
- [41] S. Saadatyar. *Medium term production planning and campaign scheduling for sawmill*. Thesis, Dalhousie University, December 2012. Graduation date: 2013.
- [42] Bowater Mersey Oakhill Sawmill. Scanned logs data. Supplied to Eldon Gunn by Hans Peterson, 2012.
- [43] E. A. Silver, D. F. Pyke, R. Peterson, et al. *Inventory management and production planning and scheduling*. Wiley New York, 3 edition, 1998.
- [44] M. Singmin and National Timber Research Institute (South Africa). *SIM-SAW: A simulation program to evaluate the effect of sawing patterns on log recovery*. National Timber Research Institute, Council for Scientific and Industrial Research, 1978.
- [45] P.H. Steele. *Factors determining lumber recovery in sawmilling*. US Department of Agriculture, Forest Service, Forest Products Laboratory, 1984.

- [46] H. Sun, H.C. Huang, and W. Jaruphongsa. A genetic algorithm for the economic lot scheduling problem under the extended basic period and power-of-two policy. *CIRP Journal of Manufacturing Science and Technology*, 2(1): 29–34, 2009.
- [47] H. Sun, H. C. Huang, and W. Jaruphongsa. The economic lot scheduling problem under extended basic period and power-of-two policy. *Optimization Letters*, 4(2):157–172, 2010.
- [48] S. Tejavibulya. *Dynamic programming sawing models for optimizing lumber recovery*. College of Forest Resources, University of Washington, 1981.
- [49] C. Todoroki and M. Ronnqvist. Dynamic control of timber production at a sawmill with log sawing optimization. *Scandinavian Journal of Forest Research*, 17(1):79–89, 2002.
- [50] C. L. Todoroki and E. M. Ronnqvist. Secondary log breakdown optimization with dynamic programming. *Journal of the Operational Research Society*, 48 (5):471–478, 1997.
- [51] C. L. Todoroki and E. M. Ronnqvist. Combined primary and secondary log breakdown optimisation. *The Journal of the Operational Research Society*, 50(3):219–229, 1999. ISSN 0160-5682.
- [52] C. L. Todoroki et al. AUTOSAW system for sawing simulation. *New Zealand Journal of Forestry Science*, 20(3):332–348, 1990.
- [53] H.P. Williams. *Model building in mathematical programming*. Wiley-Interscience publication. Wiley, 1999.
- [54] M. J. Yao and S. E. Elmaghraby. The economic lot scheduling problem under power-of-two policy. *Computers and Mathematics with Applications*, 41(10-11):1379–1393, 2001.

- [55] Y. Zeng. *Log breakdown using dynamic programming and 3-D log shape*. Thesis, Oregon State University, May 1991. Graduation date: 1991.
- [56] Y. Zeng. *Integration of an expert system and dynamic programming approach to optimize log breakdown using 3-dimensional log and internal defect shape information*. Thesis, Oregon State University, August 1995. Graduation date: 1996.

# Appendix A

## Formulas for Right-left Cuts

| Vertical right-left sub-cuts  | Horizontal right-left sub-cuts  |
|---|---|
| For each $T_l^{RL}$ where:  | For each $T_l^{RL}$ where:  |
| $T_l^{RL} \leq T_j^m \quad (A.1)$   | $T_l^{RL} \leq R - \left(\frac{W_j^m}{2} + kerf\right) \quad (A.2)$                         |
| We calculate all the combinations of possible widths and find $W_l^{RL}$ .                          | We calculate all the combinations of possible widths and find $W_l^{RL}$ .                  |
| $W_l^{RL} = \left(\sum_i Z_{i,l} * (W_i + aW_i + kerf)\right) - kerf \quad (A.3)$                   | $W_l^{RL} = \left(\sum_i Z_{i,l} * (W_i + aW_i + kerf)\right) - kerf \quad (A.4)$           |
| Where:  | Where:  |
| $W_l^{RL} \leq \sqrt{R^2 - \left(\frac{T_l^{RL}}{2}\right)^2} - \frac{W_j^m}{2} - kerf \quad (A.5)$ | $W_l^{RL} \leq 2 * \sqrt{R^2 - \left(T_{RL} + \frac{W_j^m}{2} + kerf\right)^2} \quad (A.6)$ |
|                  |         |

# Appendix B

## Market Price for Doug Fir Lumber

| Lumber dimension | Market price(\$) | Lumber dimension | Market price(\$) | Lumber dimension | Market price(\$) |
|------------------|------------------|------------------|------------------|------------------|------------------|
| 2x4x8            | 2.45             | 2x10x8           | 6.77             | 4x6x8            | 10.69            |
| 2x4x10           | 3.07             | 2x10x10          | 8.46             | 4x6x10           | 13.37            |
| 2x4x12           | 3.68             | 2x10x12          | 10.74            | 4x6x12           | 16.04            |
| 2x4x14           | 4.24             | 2x10x14          | 12.53            | 4x6x14           | 18.71            |
| 2x4x16           | 5.34             | 2x10x16          | 14.32            | 4x6x16           | 23.11            |
| 2x4x18           | 4.97             | 2x10x18          | 16.11            | 4x6x18           | 24.30            |
| 2x4x20           | 5.52             | 2x10x20          | 17.90            | 4x6x20           | 24.30            |
| 2x6x8            | 3.87             | 2x12x8           | 8.50             | 6x6x8            | 17.66            |
| 2x6x10           | 4.83             | 2x12x10          | 10.62            | 6x6x10           | 22.48            |
| 2x6x12           | 5.80             | 2x12x12          | 12.74            | 6x6x12           | 27.95            |
| 2x6x14           | 6.77             | 2x12x14          | 15.36            | 6x6x14           | 29.20            |
| 2x6x16           | 7.46             | 2x12x16          | 18.69            | 6x6x16           | 35.96            |
| 2x6x18           | 8.30             | 2x12x18          | 21.03            | 6x6x18           | 39.73            |
| 2x6x20           | 8.30             | 2x12x20          | 23.36            | 6x6x20           | 44.96            |
| 2x8x8            | 5.29             | 4x4x8            | 6.41             | 6x8x8            | 23.54            |
| 2x8x10           | 6.61             | 4x4x10           | 8.01             | 6x8x10           | 29.16            |
| 2x8x12           | 7.93             | 4x4x12           | 10.53            | 6x8x12           | 35.96            |
| 2x8x14           | 10.03            | 4x4x14           | 12.28            | 6x8x14           | 41.20            |
| 2x8x16           | 11.46            | 4x4x16           | 15.92            | 6x8x16           | 46.22            |
| 2x8x18           | 12.89            | 4x4x18           | 17.75            | 6x8x18           | 52.97            |
| 2x8x20           | 14.32            | 4x4x20           | 19.90            | 6x8x20           | 59.94            |



# Appendix C

## Length of Sub-cuts for Horizontal Above-below

$Seq_j^{AB}$ : Set of sub-cuts widths of the above-below cut with  $Y_{i,j}$  number of  $W_i$  for each  $T_j^{AB}$ . For each thickness  $T_j^{AB}$  when main cut with thickness  $T_f^m$  and width  $W_f^m$  is fixed:

Set  $Y_{ijk} = 0$

**Step 1:** Find  $W_{min} = Min[Seq_j^{AB}]$

**Step 2:** Keep  $i$  if  $W_{min} = W_i$

**Step 3:** Calculate  $rWaneUD, rWaneSide$  and  $rWane$

$$rWaneSide = \sqrt{\left(\frac{T_f^m}{2} + kerf + W_j^{AB}\right)^2 + \left(\frac{1 - wanePrSide}{2} * T_j^{AB}\right)^2} \quad (C.1)$$

$$rWaneUD = \sqrt{\left(\frac{T_f^m}{2} + kerf + W_j^{AB} - (wanePrUD * W_{min})\right)^2 + \left(\frac{T_j^{AB}}{2}\right)^2} \quad (C.2)$$

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (C.3)$$

**Step 4:** Calculate  $Lwane$

$$Lwane = \lfloor Min\{L, L * (1 - \frac{rWane - R_S}{R_L - R_S})\} / 2 \rfloor * 2 \quad (C.4)$$

**Step 5:** If  $Lwane = L$ , set  $L$  as the length of remaining cuts ( $Y_{ijk} = Y_{ijk} + Y_{i,j}$  where  $L_k = L$ ) and STOP.

Otherwise go to step 6.

**Step 6:** If  $Lwane = L_k$ , set  $Y_{ijk} = Y_{ijk} + 1$  and  $Y_{i,j} = Y_{i,j} - 1$

**Step 7:** Let  $W_j^{AB} = W_j^{AB} - (W_{min} + \ker f)$

**Step 8:** Update the set of width,  $[Seq_j^{AB}] = [Seq_j^{AB}] - [W_{min}]$

**Step 9:** If  $[Seq_j^{AB}] \neq \emptyset$  go to step 1.

Otherwise STOP.

# Appendix D

## Length of Sub-cuts for Vertical Above-below

For each thickness  $T_j^{AB}$  when main cut with thickness  $T_f^m$  and width  $W_f^m$  is fixed:

Set  $W_R = W_L \frac{W_j^{AB}}{2}$ ,  $B_R = 1$ ,  $B_L = 0$  (Starting from the right half),  $Y_{ijk} = 0$

**Step 1:** Find  $W_{min} = Min[Seq_j^{AB}]$

**Step 2:** Keep  $i$  if  $W_{min} = W_i$

**Step 3:** Calculate  $rWaneUD$ ,  $rWaneSide$  and  $rWane$

$$rWaneSide = \sqrt{((W_R * B_R) + (W_L * B_L))^2 + \left(\frac{1 - wanePrSide}{2}\right) * T_j^{AB} + kerf + \frac{T_f^m}{2}}^2 \quad (D.1)$$

$$rWaneUD = \sqrt{((W_R * B_R) + (W_L * B_L) - (wanePrUD * W_{min}))^2 + \left(\frac{T_j^m}{2} + kerf + T_j^{AB}\right)}^2 \quad (D.2)$$

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (D.3)$$

**Step 4:** Calculate  $Lwane$

$$Lwane = \lfloor Min\{L, L * (1 - \frac{rWane - R_S}{R_L - R_S})\} / 2 \rfloor * 2 \quad (D.4)$$

**Step 5:** If  $Lwane = L$ , set  $L$  as the length of remaining cuts ( $Y_{ijk} = Y_{ijk} + Y_{i,j}$  where  $L_k = L$ ) and STOP.

Otherwise go to step 6.

**Step 6:** If  $Lwane = L_k$ , set  $Y_{ijk} = Y_{ijk} + 1$  and  $Y_{i,j} = Y_{i,j} - 1$

**Step 7:** Let  $W_R = W_R - (W_{min} + kerf) * B_R$  and  $W_L = W_L - (W_{min} + kerf) * B_L$

**Step 8:** Update the set of width,  $[Seq_j^{AB}] = [Seq_j^{AB}] - [W_{min}]$  and let  $B = B_R, B_R = B_L, B_L = B$  (switch between  $W_R$  and  $W_L$ )

**Step 9:** If  $[Seq_j^{AB}] \neq \emptyset$  go to step 1.

Otherwise STOP.

# Appendix E

## Length of Sub-cuts for Vertical Right-left

$Seq_j^{RL}$ : Set of sub-cuts widths of the right-left cut with  $Z_{i,j}$  number of  $W_i$  for each  $T_j^{RL}$ . For each thickness  $T_j^{RL}$  when main cut with thickness  $T_f^m$  and width  $W_f^m$  is fixed:

Set  $Z_{ijk} = 0$

**Step 1:** Find  $W_{min} = Min[Seq_j^{RL}]$

**Step 2:** Keep  $i$  if  $W_{min} = W_i$

**Step 3:** Calculate  $rWaneUD, rWaneSide$  and  $rWane$

$$rWaneSide = \sqrt{\left(\frac{W_f^m}{2} + kerf + W_j^{RL}\right)^2 + \left(\frac{1 - wanePrSide}{2} * T_j^{RL}\right)^2} \quad (E.1)$$

$$rWaneUD = \sqrt{\left(\frac{W_f^m}{2} + kerf + W_j^{RL} - (wanePrUD * W_{min})\right)^2 + \left(\frac{T_j^{RL}}{2}\right)^2} \quad (E.2)$$

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (E.3)$$

**Step 4:** Calculate  $Lwane$

$$Lwane = \lfloor Min\{L, L * (1 - \frac{rWane - R_S}{R_L - R_S})\} / 2 \rfloor * 2 \quad (E.4)$$

**Step 5:** If  $Lwane = L$ , set  $L$  as the length of remaining cuts ( $Z_{ijk} = Z_{ijk} + Z_{i,j}$  where  $L_k = L$ ) and STOP.

Otherwise go to step 6.

**Step 6:** If  $Lwane = L_k$ , set  $Z_{ijk} = Z_{ijk} + 1$  and  $Z_{i,j} = Z_{i,j} - 1$

**Step 7:** Let  $W_j^{RL} = W_j^{RL} - (W_{min} + \ker f)$

**Step 8:** Update the set of width,  $[Seq_j^{RL}] = [Seq_j^{RL}] - [W_{min}]$

**Step 9:** If  $[Seq_j^{RL}] \neq \emptyset$  go to step 1.

Otherwise STOP.

# Appendix F

## Length of Sub-cuts for Horizontal Right-left

For each thickness  $T_j^{RL}$  when main cut with thickness  $T_f^m$  and width  $W_f^m$  is fixed:

Set  $W_R = W_L \frac{W_j^{RL}}{2}$ ,  $B_R = 1$ ,  $B_L = 0$  (Starting from the right half),  $Z_{ijk} = 0$

**Step 1:** Find  $W_{min} = Min[Seq_j^{RL}]$

**Step 2:** Keep  $i$  if  $W_{min} = W_i$

**Step 3:** Calculate  $rWaneUD$ ,  $rWaneSide$  and  $rWane$

$$rWaneSide = \sqrt{((W_R * B_R) + (W_L * B_L))^2 + \left(\frac{1 - wanePrSide}{2}\right) * T_j^{RL} + kerf + \frac{W_f^m}{2}}^2 \quad (F.1)$$

$$rWaneUD = \sqrt{((W_R * B_R) + (W_L * B_L) - (wanePrUD * W_{min}))^2 + \left(\frac{T_j^m}{2} + kerf + T_j^{RL}\right)^2} \quad (F.2)$$

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (F.3)$$

**Step 4:** Calculate  $Lwane$

$$Lwane = \lfloor Min\{L, L * (1 - \frac{rWane - R_S}{R_L - R_S})\} / 2 \rfloor * 2 \quad (F.4)$$

**Step 5:** If  $Lwane = L$ , set  $L$  as the length of remaining cuts ( $Z_{ijk} = Z_{ijk} + Z_{i,j}$  where  $L_k = L$ ) and STOP.

Otherwise go to step 6.

**Step 6:** If  $Lwane = L_k$ , set  $Z_{ijk} = Z_{ijk} + 1$  and  $Z_{i,j} = Z_{i,j} - 1$

**Step 7:** Let  $W_R = W_R - (W_{min} + kerf) * B_R$  and  $W_L = W_L - (W_{min} + kerf) * B_L$

**Step 8:** Update the set of width,  $[Seq_j^{RL}] = [Seq_j^{RL}] - [W_{min}]$  and let  $B = B_R, B_R = B_L, B_L = B$  (switch between  $W_R$  and  $W_L$ )

**Step 9:** If  $[Seq_j^{RL}] \neq \emptyset$  go to step 1.

Otherwise STOP.



# Appendix G

## Campaign Definitions

Log classes are categorized in three types: small, large and mix.

Small log class include logs with small radius in range (2-3) (*in*) and length in range (8-14) (*ft*).

A large log class has small radius with lognormal(1.198,0.323) (*in*) distribution and length in range (8-18) (*ft*).

The mix classes combine both small and large classes and sort the logs according to specific lengths as shown in following figure.

| Campaign | Log class          | Price list             | Campaign | Log class          | Price list             |
|----------|--------------------|------------------------|----------|--------------------|------------------------|
| 1        | Small              | 1 (actual price)       | 64       | 10 < Length < 12ft | 12 (thickness=6 in)    |
| 2        | Small              | 2 (volume)             | 65       | 10 < Length < 12ft | 13 (thickness=8 in)    |
| 3        | Small              | 3 (larger widths)      | 66       | 10 < Length < 12ft | 14 (thickness=10 in)   |
| 4        | Small              | 4 (larger thicknesses) | 67       | 10 < Length < 12ft | 15 (thickness=12 in)   |
| 5        | Small              | 5 (larger lengths)     | 68       | 10 < Length < 12ft | 16 (length=8 ft)       |
| 6        | Small              | 6 (width=1 in)         | 69       | 10 < Length < 12ft | 17 (length=10ft)       |
| 7        | Small              | 7 (width=2 in)         |          |                    |                        |
| 8        | Small              | 8 (width=4 in)         | 70       | 12 < Length < 14ft | 1 (actual price)       |
| 9        | Small              | 9 (width=6 in)         | 71       | 12 < Length < 14ft | 2 (volume)             |
| 10       | Small              | 10 (thickness=3 in)    | 72       | 12 < Length < 14ft | 3 (larger widths)      |
| 11       | Small              | 11 (thickness=4 in)    | 73       | 12 < Length < 14ft | 4 (larger thicknesses) |
| 12       | Small              | 12 (thickness=6 in)    | 74       | 12 < Length < 14ft | 5 (larger lengths)     |
| 13       | Small              | 13 (thickness=8 in)    | 75       | 12 < Length < 14ft | 6 (width=1 in)         |
| 14       | Small              | 16 (length=8 ft)       | 76       | 12 < Length < 14ft | 7 (width=2 in)         |
| 15       | Small              | 17 (length=10ft)       | 77       | 12 < Length < 14ft | 8 (width=4 in)         |
| 16       | Small              | 18 (length=12 ft)      | 78       | 12 < Length < 14ft | 9 (width=6 in)         |
|          |                    |                        | 79       | 12 < Length < 14ft | 10 (thickness=3 in)    |
| 17       | Large              | 1 (actual price)       | 80       | 12 < Length < 14ft | 11 (thickness=4 in)    |
| 18       | Large              | 2 (volume)             | 81       | 12 < Length < 14ft | 12 (thickness=6 in)    |
| 19       | Large              | 3 (larger widths)      | 82       | 12 < Length < 14ft | 13 (thickness=8 in)    |
| 20       | Large              | 4 (larger thicknesses) | 83       | 12 < Length < 14ft | 14 (thickness=10 in)   |
| 21       | Large              | 5 (larger lengths)     | 84       | 12 < Length < 14ft | 15 (thickness=12 in)   |
| 22       | Large              | 6 (width=1 in)         | 85       | 12 < Length < 14ft | 16 (length=8 ft)       |
| 23       | Large              | 7 (width=2 in)         | 86       | 12 < Length < 14ft | 17 (length=10ft)       |
| 24       | Large              | 8 (width=4 in)         | 87       | 12 < Length < 14ft | 18 (length=12 ft)      |
| 25       | Large              | 9 (width=6 in)         |          |                    |                        |
| 26       | Large              | 10 (thickness=3 in)    | 88       | 14 < Length < 16ft | 1 (actual price)       |
| 27       | Large              | 11 (thickness=4 in)    | 89       | 14 < Length < 16ft | 2 (volume)             |
| 28       | Large              | 12 (thickness=6 in)    | 90       | 14 < Length < 16ft | 3 (larger widths)      |
| 29       | Large              | 13 (thickness=8 in)    | 91       | 14 < Length < 16ft | 4 (larger thicknesses) |
| 30       | Large              | 14 (thickness=10 in)   | 92       | 14 < Length < 16ft | 5 (larger lengths)     |
| 31       | Large              | 15 (thickness=12 in)   | 93       | 14 < Length < 16ft | 6 (width=1 in)         |
| 32       | Large              | 16 (length=8 ft)       | 94       | 14 < Length < 16ft | 7 (width=2 in)         |
| 33       | Large              | 17 (length=10ft)       | 95       | 14 < Length < 16ft | 8 (width=4 in)         |
| 34       | Large              | 18 (length=12 ft)      | 96       | 14 < Length < 16ft | 9 (width=6 in)         |
| 35       | Large              | 19 (length=14 ft)      | 97       | 14 < Length < 16ft | 10 (thickness=3 in)    |
| 36       | Large              | 20 (length=16 ft)      | 98       | 14 < Length < 16ft | 11 (thickness=4 in)    |
|          |                    |                        | 99       | 14 < Length < 16ft | 12 (thickness=6 in)    |
| 37       | Length < 10ft      | 1 (actual price)       | 100      | 14 < Length < 16ft | 13 (thickness=8 in)    |
| 38       | Length < 10ft      | 2 (volume)             | 101      | 14 < Length < 16ft | 14 (thickness=10 in)   |
| 39       | Length < 10ft      | 3 (larger widths)      | 102      | 14 < Length < 16ft | 15 (thickness=12 in)   |
| 40       | Length < 10ft      | 4 (larger thicknesses) | 103      | 14 < Length < 16ft | 16 (length=8 ft)       |
| 41       | Length < 10ft      | 5 (larger lengths)     | 104      | 14 < Length < 16ft | 17 (length=10ft)       |
| 42       | Length < 10ft      | 6 (width=1 in)         | 105      | 14 < Length < 16ft | 18 (length=12 ft)      |
| 43       | Length < 10ft      | 7 (width=2 in)         | 106      | 14 < Length < 16ft | 19 (length=14 ft)      |
| 44       | Length < 10ft      | 8 (width=4 in)         |          |                    |                        |
| 45       | Length < 10ft      | 9 (width=6 in)         | 107      | 16 < Length < 18ft | 1 (actual price)       |
| 46       | Length < 10ft      | 10 (thickness=3 in)    | 108      | 16 < Length < 18ft | 2 (volume)             |
| 47       | Length < 10ft      | 11 (thickness=4 in)    | 109      | 16 < Length < 18ft | 3 (larger widths)      |
| 48       | Length < 10ft      | 12 (thickness=6 in)    | 110      | 16 < Length < 18ft | 4 (larger thicknesses) |
| 49       | Length < 10ft      | 13 (thickness=8 in)    | 111      | 16 < Length < 18ft | 5 (larger lengths)     |
| 50       | Length < 10ft      | 14 (thickness=10 in)   | 112      | 16 < Length < 18ft | 6 (width=1 in)         |
| 51       | Length < 10ft      | 15 (thickness=12 in)   | 113      | 16 < Length < 18ft | 7 (width=2 in)         |
| 52       | Length < 10ft      | 16 (length=8 ft)       | 114      | 16 < Length < 18ft | 8 (width=4 in)         |
|          |                    |                        | 115      | 16 < Length < 18ft | 9 (width=6 in)         |
| 53       | 10 < Length < 12ft | 1 (actual price)       | 116      | 16 < Length < 18ft | 10 (thickness=3 in)    |
| 54       | 10 < Length < 12ft | 2 (volume)             | 117      | 16 < Length < 18ft | 11 (thickness=4 in)    |
| 55       | 10 < Length < 12ft | 3 (larger widths)      | 118      | 16 < Length < 18ft | 12 (thickness=6 in)    |
| 56       | 10 < Length < 12ft | 4 (larger thicknesses) | 119      | 16 < Length < 18ft | 13 (thickness=8 in)    |
| 57       | 10 < Length < 12ft | 5 (larger lengths)     | 120      | 16 < Length < 18ft | 14 (thickness=10 in)   |
| 58       | 10 < Length < 12ft | 6 (width=1 in)         | 121      | 16 < Length < 18ft | 15 (thickness=12 in)   |
| 59       | 10 < Length < 12ft | 7 (width=2 in)         | 122      | 16 < Length < 18ft | 16 (length=8 ft)       |
| 60       | 10 < Length < 12ft | 8 (width=4 in)         | 123      | 16 < Length < 18ft | 17 (length=10ft)       |
| 61       | 10 < Length < 12ft | 9 (width=6 in)         | 124      | 16 < Length < 18ft | 18 (length=12 ft)      |
| 62       | 10 < Length < 12ft | 10 (thickness=3 in)    | 125      | 16 < Length < 18ft | 19 (length=14 ft)      |
| 63       | 10 < Length < 12ft | 11 (thickness=4 in)    | 126      | 16 < Length < 18ft | 20 (length=16 ft)      |

# Appendix H

## Campaign Data

| Campaign | $R_c(ft^3/year)$ | $S_c(year)$ | Campaign | $R_c(ft^3/year)$ | $S_c(year)$ | Campaign | $R_c(ft^3/year)$ | $S_c(year)$ |
|----------|------------------|-------------|----------|------------------|-------------|----------|------------------|-------------|
| 1        | 13045027.35      | 0.00080     | 43       | 12363715.01      | 0.00077     | 85       | 14329959         | 0.00039     |
| 2        | 11895443.61      | 0.00044     | 44       | 13428144.01      | 0.00052     | 86       | 13129040.8       | 0.00064     |
| 3        | 13149412.31      | 0.00082     | 45       | 12356046.52      | 0.00045     | 87       | 11432021.76      | 0.00077     |
| 4        | 12282394.24      | 0.00050     | 46       | 13998591.49      | 0.00045     | 88       | 11150603.74      | 0.00072     |
| 5        | 11951978.19      | 0.00047     | 47       | 13110858.55      | 0.00058     | 89       | 10172540.02      | 0.00061     |
| 6        | 13793080.76      | 0.00065     | 48       | 12959586.71      | 0.00043     | 90       | 11351201         | 0.00063     |
| 7        | 12370160.62      | 0.00058     | 49       | 12343090.74      | 0.00039     | 91       | 10541679.01      | 0.00079     |
| 8        | 13080949.4       | 0.00052     | 50       | 12336734.21      | 0.00054     | 92       | 10251669.47      | 0.00039     |
| 9        | 11996318.01      | 0.00045     | 51       | 12325277.24      | 0.00038     | 93       | 11259971.41      | 0.00067     |
| 10       | 13402767.38      | 0.00040     | 52       | 12316152.29      | 0.00076     | 94       | 10593096.97      | 0.00052     |
| 11       | 12408411.73      | 0.00076     | 53       | 12349549.24      | 0.00079     | 95       | 14498238.05      | 0.00070     |
| 12       | 13108847.6       | 0.00045     | 54       | 11244859.77      | 0.00035     | 96       | 11257343.81      | 0.00077     |
| 13       | 11907748.84      | 0.00056     | 55       | 12367801.94      | 0.00075     | 97       | 15286744.19      | 0.00038     |
| 14       | 13873641.39      | 0.00073     | 56       | 11773136.04      | 0.00045     | 98       | 13071015.14      | 0.00067     |
| 15       | 12864146.31      | 0.00053     | 57       | 11318475.9       | 0.00041     | 99       | 10866256.17      | 0.00073     |
| 16       | 12178323.95      | 0.00050     | 58       | 12901852.34      | 0.00032     | 100      | 11025857.14      | 0.00060     |
| 17       | 11150860.83      | 0.00080     | 59       | 11798619.74      | 0.00044     | 101      | 10923315.23      | 0.00071     |
| 18       | 10169367.33      | 0.00064     | 60       | 13252778.15      | 0.00062     | 102      | 10551548.32      | 0.00040     |
| 19       | 11298743.89      | 0.00065     | 61       | 11538586.35      | 0.00035     | 103      | 14724133.65      | 0.00029     |
| 20       | 10526248.76      | 0.00049     | 62       | 13853553.96      | 0.00075     | 104      | 13449712.83      | 0.00040     |
| 21       | 10250687.77      | 0.00059     | 63       | 12624364.49      | 0.00057     | 105      | 12475337.69      | 0.00032     |
| 22       | 11270405.54      | 0.00080     | 64       | 12180843.98      | 0.00073     | 106      | 10966737.27      | 0.00033     |
| 23       | 10547434.69      | 0.00071     | 65       | 11439393.17      | 0.00073     | 107      | 11058466.89      | 0.00036     |
| 24       | 14241369.53      | 0.00051     | 66       | 11402476.29      | 0.00070     | 108      | 10098381.81      | 0.00048     |
| 25       | 11253560.25      | 0.00081     | 67       | 11315538.42      | 0.00067     | 109      | 11231955.38      | 0.00076     |
| 26       | 15155627.68      | 0.00050     | 68       | 13924484.65      | 0.00057     | 110      | 10414007.61      | 0.00037     |
| 27       | 13090149.22      | 0.00044     | 69       | 11904103.57      | 0.00038     | 111      | 10195366.81      | 0.00037     |
| 28       | 10853134.6       | 0.00047     | 70       | 11804213.77      | 0.00032     | 112      | 11133989.9       | 0.00057     |
| 29       | 11047897.87      | 0.00040     | 71       | 10639857.1       | 0.00074     | 113      | 10439598.59      | 0.00032     |
| 30       | 10997428.3       | 0.00028     | 72       | 11904905.97      | 0.00073     | 114      | 14418333.5       | 0.00041     |
| 31       | 10609038.06      | 0.00055     | 73       | 11065372.64      | 0.00034     | 115      | 11283374.27      | 0.00081     |
| 32       | 14622230.88      | 0.00034     | 74       | 10692462.98      | 0.00081     | 116      | 15048129.08      | 0.00050     |
| 33       | 13317132.55      | 0.00071     | 75       | 11852102.41      | 0.00046     | 117      | 13110277.32      | 0.00047     |
| 34       | 12218625.69      | 0.00075     | 76       | 11158270.75      | 0.00063     | 118      | 10785180.95      | 0.00032     |
| 35       | 11448484.87      | 0.00055     | 77       | 13468372.12      | 0.00041     | 119      | 11116206.84      | 0.00075     |
| 36       | 10767660.36      | 0.00060     | 78       | 11410290.27      | 0.00028     | 120      | 11112374.85      | 0.00064     |
| 37       | 13136651.65      | 0.00044     | 79       | 14582336.18      | 0.00039     | 121      | 10656139.96      | 0.00058     |
| 38       | 12316152.29      | 0.00079     | 80       | 12584746.56      | 0.00070     | 122      | 15103480.14      | 0.00080     |
| 39       | 13322584.8       | 0.00077     | 81       | 11655862.8       | 0.00052     | 123      | 13939628.69      | 0.00057     |
| 40       | 12452713         | 0.00028     | 82       | 11116016.33      | 0.00065     | 124      | 12967443.42      | 0.00057     |
| 41       | 12316152.29      | 0.00076     | 83       | 11008588.67      | 0.00034     | 125      | 12191957.54      | 0.00077     |
| 42       | 14908973.01      | 0.00069     | 84       | 10810221.89      | 0.00069     | 126      | 11135430.64      | 0.00075     |

# Appendix I

## Fractional Outputs ( $\alpha_{cp}$ )

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 1        | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      |
| 1         | 0.02485  | 0.01095 | 0.00082 | 0.00000 | 0.00000 | 0.02802 | 0.00721 | 0.00056 | 0.00000 | 0.00000 |
| 2         | 0.03419  | 0.02019 | 0.00405 | 0.00000 | 0.00000 | 0.04293 | 0.01501 | 0.00805 | 0.00000 | 0.00000 |
| 3         | 0.02312  | 0.01040 | 0.00076 | 0.00000 | 0.00000 | 0.02812 | 0.00720 | 0.00056 | 0.00000 | 0.00000 |
| 4         | 0.02189  | 0.00868 | 0.00125 | 0.00000 | 0.00000 | 0.04602 | 0.01686 | 0.00898 | 0.00000 | 0.00000 |
| 5         | 0.03373  | 0.02836 | 0.00368 | 0.00000 | 0.00000 | 0.03928 | 0.02055 | 0.01689 | 0.00000 | 0.00000 |
| 6         | 0.03991  | 0.03202 | 0.00409 | 0.00000 | 0.00000 | 0.18252 | 0.13059 | 0.08369 | 0.00000 | 0.00000 |
| 7         | 0.02042  | 0.00038 | 0.00022 | 0.00000 | 0.00000 | 0.02447 | 0.00415 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.02393  | 0.01016 | 0.00062 | 0.00000 | 0.00000 | 0.03128 | 0.01071 | 0.00226 | 0.00000 | 0.00000 |
| 9         | 0.03421  | 0.02019 | 0.00405 | 0.00000 | 0.00000 | 0.04083 | 0.01181 | 0.00689 | 0.00000 | 0.00000 |
| 10        | 0.05758  | 0.05235 | 0.02303 | 0.00000 | 0.00000 | 0.00025 | 0.00007 | 0.00007 | 0.00000 | 0.00000 |
| 11        | 0.00815  | 0.00016 | 0.00000 | 0.00000 | 0.00000 | 0.07445 | 0.01623 | 0.01183 | 0.00000 | 0.00000 |
| 12        | 0.01494  | 0.02270 | 0.00319 | 0.00000 | 0.00000 | 0.02946 | 0.02685 | 0.00684 | 0.00000 | 0.00000 |
| 13        | 0.03426  | 0.02019 | 0.00405 | 0.00000 | 0.00000 | 0.04266 | 0.01506 | 0.00850 | 0.00000 | 0.00000 |
| 14        | 0.04052  | 0.00805 | 0.00247 | 0.00000 | 0.00000 | 0.02579 | 0.00434 | 0.00247 | 0.00000 | 0.00000 |
| 15        | 0.02184  | 0.02713 | 0.00263 | 0.00000 | 0.00000 | 0.02156 | 0.01110 | 0.00099 | 0.00000 | 0.00000 |
| 16        | 0.03315  | 0.01483 | 0.01016 | 0.00000 | 0.00000 | 0.03605 | 0.01016 | 0.00892 | 0.00000 | 0.00000 |
| 17        | 0.00442  | 0.00404 | 0.00248 | 0.00064 | 0.00077 | 0.01140 | 0.00987 | 0.00535 | 0.00226 | 0.00233 |
| 18        | 0.00429  | 0.00387 | 0.00239 | 0.00146 | 0.00111 | 0.02056 | 0.01974 | 0.01263 | 0.00734 | 0.00985 |
| 19        | 0.00436  | 0.00402 | 0.00181 | 0.00060 | 0.00084 | 0.01032 | 0.00869 | 0.00475 | 0.00126 | 0.00115 |
| 20        | 0.00604  | 0.00474 | 0.00295 | 0.00131 | 0.00129 | 0.01779 | 0.01813 | 0.01004 | 0.00440 | 0.00441 |
| 21        | 0.00434  | 0.00632 | 0.00230 | 0.00170 | 0.00203 | 0.01614 | 0.02107 | 0.01886 | 0.01448 | 0.01403 |
| 22        | 0.00550  | 0.00619 | 0.00511 | 0.00254 | 0.00252 | 0.02437 | 0.03498 | 0.04437 | 0.02864 | 0.05327 |
| 23        | 0.00141  | 0.00030 | 0.00020 | 0.00014 | 0.00019 | 0.01279 | 0.00928 | 0.00246 | 0.00088 | 0.00047 |
| 24        | 0.00316  | 0.00432 | 0.00253 | 0.00064 | 0.00053 | 0.00929 | 0.01037 | 0.00908 | 0.00526 | 0.00588 |
| 25        | 0.00453  | 0.00337 | 0.00199 | 0.00138 | 0.00232 | 0.01470 | 0.01116 | 0.00685 | 0.00374 | 0.00576 |
| 26        | 0.01103  | 0.01707 | 0.01860 | 0.01818 | 0.01888 | 0.00005 | 0.00005 | 0.00013 | 0.00004 | 0.00004 |
| 27        | 0.00084  | 0.00013 | 0.00000 | 0.00000 | 0.00000 | 0.01927 | 0.02159 | 0.02721 | 0.01894 | 0.02746 |
| 28        | 0.00335  | 0.00508 | 0.00323 | 0.00124 | 0.00077 | 0.00562 | 0.00720 | 0.00318 | 0.00131 | 0.00090 |
| 29        | 0.00589  | 0.00509 | 0.00257 | 0.00108 | 0.00077 | 0.01402 | 0.02238 | 0.01714 | 0.00800 | 0.00877 |
| 30        | 0.00332  | 0.00338 | 0.00201 | 0.00102 | 0.00078 | 0.02096 | 0.02198 | 0.01212 | 0.00807 | 0.00932 |
| 31        | 0.00391  | 0.00369 | 0.00227 | 0.00139 | 0.00108 | 0.02047 | 0.01941 | 0.01236 | 0.00701 | 0.00953 |
| 32        | 0.00665  | 0.00208 | 0.00200 | 0.00045 | 0.00030 | 0.00914 | 0.00498 | 0.00236 | 0.00180 | 0.00218 |
| 33        | 0.00282  | 0.00090 | 0.00262 | 0.00172 | 0.00053 | 0.00483 | 0.01316 | 0.00352 | 0.00140 | 0.00246 |
| 34        | 0.00332  | 0.00156 | 0.01061 | 0.00141 | 0.00189 | 0.01076 | 0.00823 | 0.01923 | 0.00145 | 0.00211 |
| 35        | 0.00356  | 0.00296 | 0.00155 | 0.00906 | 0.00242 | 0.01538 | 0.01475 | 0.00839 | 0.01388 | 0.00255 |
| 36        | 0.00344  | 0.00392 | 0.00200 | 0.00054 | 0.00885 | 0.01497 | 0.01871 | 0.01281 | 0.00312 | 0.03180 |
| 37        | 0.01737  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.02709 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.04940  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03555 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.01224  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.02699 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.03070  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.05280 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.04944  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03505 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.05638  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.35492 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.03502  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01284 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.01459  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.02981 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.04965  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03412 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.12957  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00056 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00908  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.07036 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.02375  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.06411 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.04960  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03522 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.04929  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03533 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.04938  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03550 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.04942  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03519 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.02338  | 0.01055 | 0.00000 | 0.00000 | 0.00000 | 0.02088 | 0.01293 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.02313  | 0.02636 | 0.00000 | 0.00000 | 0.00000 | 0.03806 | 0.02685 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.02307  | 0.01049 | 0.00000 | 0.00000 | 0.00000 | 0.02100 | 0.01275 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.01387  | 0.01668 | 0.00000 | 0.00000 | 0.00000 | 0.02925 | 0.02596 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.02316  | 0.03563 | 0.00000 | 0.00000 | 0.00000 | 0.02861 | 0.03505 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.02859  | 0.04782 | 0.00000 | 0.00000 | 0.00000 | 0.08411 | 0.24283 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.01011  | 0.00123 | 0.00000 | 0.00000 | 0.00000 | 0.02284 | 0.00723 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.02287  | 0.01016 | 0.00000 | 0.00000 | 0.00000 | 0.02105 | 0.01741 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.02266  | 0.02600 | 0.00000 | 0.00000 | 0.00000 | 0.03490 | 0.01895 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.02710  | 0.08892 | 0.00000 | 0.00000 | 0.00000 | 0.00012 | 0.00027 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00526  | 0.00013 | 0.00000 | 0.00000 | 0.00000 | 0.05749 | 0.03213 | 0.00000 | 0.00000 | 0.00000 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 1        | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      |
| 64        | 0.00956  | 0.02788 | 0.00000 | 0.00000 | 0.00000 | 0.01480 | 0.04005 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.02361  | 0.02717 | 0.00000 | 0.00000 | 0.00000 | 0.03463 | 0.02967 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.02236  | 0.02604 | 0.00000 | 0.00000 | 0.00000 | 0.03762 | 0.02659 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.02296  | 0.02632 | 0.00000 | 0.00000 | 0.00000 | 0.03806 | 0.02614 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.02826  | 0.00987 | 0.00000 | 0.00000 | 0.00000 | 0.02093 | 0.01115 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00489  | 0.03600 | 0.00000 | 0.00000 | 0.00000 | 0.01169 | 0.02757 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.01200  | 0.01108 | 0.00347 | 0.00000 | 0.00000 | 0.02061 | 0.00850 | 0.00571 | 0.00000 | 0.00000 |
| 71        | 0.01106  | 0.00983 | 0.00875 | 0.00000 | 0.00000 | 0.03599 | 0.02497 | 0.02764 | 0.00000 | 0.00000 |
| 72        | 0.01203  | 0.01039 | 0.00344 | 0.00000 | 0.00000 | 0.02028 | 0.00849 | 0.00521 | 0.00000 | 0.00000 |
| 73        | 0.01244  | 0.00605 | 0.00666 | 0.00000 | 0.00000 | 0.02967 | 0.01742 | 0.02206 | 0.00000 | 0.00000 |
| 74        | 0.01024  | 0.01823 | 0.00889 | 0.00000 | 0.00000 | 0.03462 | 0.02699 | 0.04978 | 0.00000 | 0.00000 |
| 75        | 0.01260  | 0.01313 | 0.01400 | 0.00000 | 0.00000 | 0.05080 | 0.03615 | 0.20132 | 0.00000 | 0.00000 |
| 76        | 0.00564  | 0.00004 | 0.00090 | 0.00000 | 0.00000 | 0.02241 | 0.00793 | 0.00289 | 0.00000 | 0.00000 |
| 77        | 0.01090  | 0.01054 | 0.00305 | 0.00000 | 0.00000 | 0.02343 | 0.01614 | 0.01758 | 0.00000 | 0.00000 |
| 78        | 0.01082  | 0.00954 | 0.00803 | 0.00000 | 0.00000 | 0.02774 | 0.01355 | 0.01858 | 0.00000 | 0.00000 |
| 79        | 0.01059  | 0.02445 | 0.05736 | 0.00000 | 0.00000 | 0.00017 | 0.00001 | 0.00046 | 0.00000 | 0.00000 |
| 80        | 0.00371  | 0.00027 | 0.00001 | 0.00000 | 0.00000 | 0.04523 | 0.02779 | 0.05597 | 0.00000 | 0.00000 |
| 81        | 0.00647  | 0.01482 | 0.00801 | 0.00000 | 0.00000 | 0.00580 | 0.01187 | 0.01371 | 0.00000 | 0.00000 |
| 82        | 0.01140  | 0.01097 | 0.00885 | 0.00000 | 0.00000 | 0.02924 | 0.02560 | 0.03754 | 0.00000 | 0.00000 |
| 83        | 0.01002  | 0.00951 | 0.00833 | 0.00000 | 0.00000 | 0.03519 | 0.02414 | 0.02730 | 0.00000 | 0.00000 |
| 84        | 0.01080  | 0.00967 | 0.00867 | 0.00000 | 0.00000 | 0.03606 | 0.02471 | 0.02704 | 0.00000 | 0.00000 |
| 85        | 0.01732  | 0.00605 | 0.00558 | 0.00000 | 0.00000 | 0.01292 | 0.00312 | 0.00824 | 0.00000 | 0.00000 |
| 86        | 0.00669  | 0.02031 | 0.00773 | 0.00000 | 0.00000 | 0.00973 | 0.01467 | 0.00655 | 0.00000 | 0.00000 |
| 87        | 0.00826  | 0.00033 | 0.02371 | 0.00000 | 0.00000 | 0.01822 | 0.00564 | 0.05246 | 0.00000 | 0.00000 |
| 88        | 0.00439  | 0.00381 | 0.00162 | 0.00198 | 0.00000 | 0.01281 | 0.01032 | 0.00505 | 0.00664 | 0.00000 |
| 89        | 0.00446  | 0.00414 | 0.00221 | 0.00332 | 0.00000 | 0.02225 | 0.01894 | 0.01189 | 0.01935 | 0.00000 |
| 90        | 0.00413  | 0.00400 | 0.00163 | 0.00190 | 0.00000 | 0.01051 | 0.00876 | 0.00294 | 0.00371 | 0.00000 |
| 91        | 0.00656  | 0.00539 | 0.00237 | 0.00412 | 0.00000 | 0.01965 | 0.01914 | 0.00795 | 0.01025 | 0.00000 |
| 92        | 0.00432  | 0.00545 | 0.00229 | 0.00517 | 0.00000 | 0.01744 | 0.01941 | 0.01686 | 0.03334 | 0.00000 |
| 93        | 0.00501  | 0.00542 | 0.00464 | 0.00743 | 0.00000 | 0.02494 | 0.02294 | 0.02533 | 0.11852 | 0.00000 |
| 94        | 0.00127  | 0.00004 | 0.00002 | 0.00044 | 0.00000 | 0.01400 | 0.00907 | 0.00169 | 0.00124 | 0.00000 |
| 95        | 0.00303  | 0.00446 | 0.00182 | 0.00191 | 0.00000 | 0.00973 | 0.01091 | 0.00861 | 0.01461 | 0.00000 |
| 96        | 0.00509  | 0.00367 | 0.00269 | 0.00351 | 0.00000 | 0.01538 | 0.01096 | 0.00642 | 0.01206 | 0.00000 |
| 97        | 0.01045  | 0.01264 | 0.01820 | 0.05253 | 0.00000 | 0.00000 | 0.00013 | 0.00000 | 0.00034 | 0.00000 |
| 98        | 0.00073  | 0.00008 | 0.00000 | 0.00001 | 0.00000 | 0.02008 | 0.01883 | 0.02265 | 0.05111 | 0.00000 |
| 99        | 0.00353  | 0.00454 | 0.00385 | 0.00264 | 0.00000 | 0.00596 | 0.00647 | 0.00357 | 0.00322 | 0.00000 |
| 100       | 0.00629  | 0.00630 | 0.00244 | 0.00274 | 0.00000 | 0.01370 | 0.02184 | 0.01414 | 0.02288 | 0.00000 |
| 101       | 0.00351  | 0.00360 | 0.00192 | 0.00261 | 0.00000 | 0.02294 | 0.02002 | 0.01100 | 0.01997 | 0.00000 |
| 102       | 0.00400  | 0.00399 | 0.00207 | 0.00314 | 0.00000 | 0.02198 | 0.01885 | 0.01151 | 0.01862 | 0.00000 |
| 103       | 0.00674  | 0.00171 | 0.00243 | 0.00109 | 0.00000 | 0.00879 | 0.00430 | 0.00139 | 0.00566 | 0.00000 |
| 104       | 0.00307  | 0.00958 | 0.00251 | 0.00394 | 0.00000 | 0.00509 | 0.01001 | 0.00286 | 0.00534 | 0.00000 |
| 105       | 0.00297  | 0.00093 | 0.01254 | 0.00495 | 0.00000 | 0.00996 | 0.00634 | 0.01137 | 0.00445 | 0.00000 |
| 106       | 0.00249  | 0.00223 | 0.00011 | 0.01661 | 0.00000 | 0.01428 | 0.01091 | 0.00139 | 0.05592 | 0.00000 |
| 107       | 0.00295  | 0.00300 | 0.00264 | 0.00064 | 0.00130 | 0.00968 | 0.00964 | 0.00552 | 0.00238 | 0.00393 |
| 108       | 0.00251  | 0.00262 | 0.00169 | 0.00174 | 0.00187 | 0.01622 | 0.01577 | 0.01010 | 0.00821 | 0.01663 |
| 109       | 0.00299  | 0.00298 | 0.00149 | 0.00060 | 0.00142 | 0.00854 | 0.00804 | 0.00521 | 0.00133 | 0.00194 |
| 110       | 0.00433  | 0.00266 | 0.00199 | 0.00133 | 0.00217 | 0.01694 | 0.01805 | 0.00981 | 0.00521 | 0.00745 |
| 111       | 0.00273  | 0.00367 | 0.00124 | 0.00175 | 0.00342 | 0.01201 | 0.01705 | 0.01401 | 0.01723 | 0.02369 |
| 112       | 0.00361  | 0.00342 | 0.00326 | 0.00269 | 0.00425 | 0.01699 | 0.02059 | 0.02095 | 0.02271 | 0.08992 |
| 113       | 0.00067  | 0.00024 | 0.00003 | 0.00014 | 0.00031 | 0.01085 | 0.00957 | 0.00240 | 0.00122 | 0.00079 |
| 114       | 0.00182  | 0.00329 | 0.00277 | 0.00067 | 0.00090 | 0.00672 | 0.00702 | 0.00633 | 0.00572 | 0.00992 |
| 115       | 0.00304  | 0.00216 | 0.00125 | 0.00157 | 0.00392 | 0.01203 | 0.00963 | 0.00521 | 0.00370 | 0.00971 |
| 116       | 0.00948  | 0.01104 | 0.01318 | 0.01932 | 0.03186 | 0.00000 | 0.00000 | 0.00004 | 0.00000 | 0.00007 |
| 117       | 0.00046  | 0.00011 | 0.00000 | 0.00001 | 0.00000 | 0.01427 | 0.01628 | 0.02096 | 0.02092 | 0.04636 |
| 118       | 0.00226  | 0.00307 | 0.00261 | 0.00152 | 0.00129 | 0.00456 | 0.00545 | 0.00194 | 0.00151 | 0.00151 |
| 119       | 0.00448  | 0.00337 | 0.00190 | 0.00123 | 0.00129 | 0.01147 | 0.01827 | 0.01274 | 0.00856 | 0.01481 |
| 120       | 0.00189  | 0.00219 | 0.00131 | 0.00116 | 0.00131 | 0.01734 | 0.01981 | 0.00961 | 0.00930 | 0.01573 |
| 121       | 0.00217  | 0.00245 | 0.00155 | 0.00166 | 0.00182 | 0.01611 | 0.01564 | 0.01002 | 0.00781 | 0.01608 |
| 122       | 0.00472  | 0.00115 | 0.00160 | 0.00053 | 0.00050 | 0.00758 | 0.00429 | 0.00114 | 0.00182 | 0.00367 |
| 123       | 0.00217  | 0.00698 | 0.00168 | 0.00206 | 0.00090 | 0.00360 | 0.00902 | 0.00275 | 0.00120 | 0.00415 |
| 124       | 0.00194  | 0.00090 | 0.00966 | 0.00131 | 0.00319 | 0.00690 | 0.00556 | 0.00994 | 0.00148 | 0.00357 |
| 125       | 0.00171  | 0.00150 | 0.00072 | 0.01170 | 0.00409 | 0.00921 | 0.00908 | 0.00523 | 0.01133 | 0.00431 |
| 126       | 0.00107  | 0.00272 | 0.00103 | 0.00019 | 0.01493 | 0.00679 | 0.01403 | 0.01041 | 0.00108 | 0.05367 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 11       | 12      | 13      | 14      | 15      | 16      | 17      | 18      | 19      | 20      |
| 1         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 2         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 3         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 4         | 0.01236  | 0.00172 | 0.00021 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 5         | 0.00003  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 6         | 0.00789  | 0.00085 | 0.00086 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 9         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 10        | 0.00024  | 0.00010 | 0.00001 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12        | 0.02663  | 0.00244 | 0.00020 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00006 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14        | 0.00367  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15        | 0.00043  | 0.00087 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 16        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17        | 0.01381  | 0.00617 | 0.00220 | 0.00071 | 0.00071 | 0.00032 | 0.00019 | 0.00010 | 0.00007 | 0.00005 |
| 18        | 0.01482  | 0.00726 | 0.00345 | 0.00375 | 0.00338 | 0.00441 | 0.00105 | 0.00128 | 0.00066 | 0.00050 |
| 19        | 0.01148  | 0.00455 | 0.00155 | 0.00036 | 0.00057 | 0.00068 | 0.00018 | 0.00002 | 0.00000 | 0.00001 |
| 20        | 0.01655  | 0.01236 | 0.00670 | 0.00322 | 0.00296 | 0.00919 | 0.00384 | 0.00306 | 0.00169 | 0.00163 |
| 21        | 0.02065  | 0.00899 | 0.01007 | 0.00562 | 0.00639 | 0.00380 | 0.00152 | 0.00153 | 0.00078 | 0.00055 |
| 22        | 0.02763  | 0.02708 | 0.04587 | 0.03573 | 0.08338 | 0.01084 | 0.01128 | 0.02256 | 0.01647 | 0.05057 |
| 23        | 0.00246  | 0.00006 | 0.00001 | 0.00000 | 0.00000 | 0.00030 | 0.00003 | 0.00000 | 0.00000 | 0.00000 |
| 24        | 0.00022  | 0.00043 | 0.00033 | 0.00021 | 0.00034 | 0.00112 | 0.00063 | 0.00126 | 0.00102 | 0.00482 |
| 25        | 0.01652  | 0.00863 | 0.00579 | 0.00484 | 0.00480 | 0.00057 | 0.00026 | 0.00011 | 0.00011 | 0.00011 |
| 26        | 0.00017  | 0.00022 | 0.00028 | 0.00031 | 0.00030 | 0.00354 | 0.00196 | 0.00178 | 0.00103 | 0.00436 |
| 27        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00042 | 0.00033 | 0.00051 | 0.00032 | 0.00108 |
| 28        | 0.03366  | 0.01657 | 0.00823 | 0.00654 | 0.00787 | 0.00033 | 0.00035 | 0.00045 | 0.00048 | 0.00063 |
| 29        | 0.00150  | 0.00251 | 0.00092 | 0.00025 | 0.00013 | 0.02852 | 0.01063 | 0.00555 | 0.00325 | 0.00314 |
| 30        | 0.00829  | 0.00808 | 0.00396 | 0.00278 | 0.00271 | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 31        | 0.01239  | 0.00769 | 0.00400 | 0.00412 | 0.00341 | 0.00184 | 0.00229 | 0.00076 | 0.00048 | 0.00112 |
| 32        | 0.00919  | 0.00042 | 0.00033 | 0.00045 | 0.00078 | 0.00667 | 0.00026 | 0.00104 | 0.00058 | 0.00163 |
| 33        | 0.00266  | 0.01050 | 0.00039 | 0.00015 | 0.00104 | 0.00085 | 0.00556 | 0.00016 | 0.00049 | 0.00085 |
| 34        | 0.00547  | 0.00220 | 0.00914 | 0.00023 | 0.00090 | 0.00293 | 0.00041 | 0.00472 | 0.00013 | 0.00086 |
| 35        | 0.01081  | 0.00401 | 0.00219 | 0.00710 | 0.00089 | 0.00225 | 0.00164 | 0.00036 | 0.00310 | 0.00024 |
| 36        | 0.00921  | 0.00677 | 0.00267 | 0.00102 | 0.00899 | 0.00187 | 0.00074 | 0.00034 | 0.00015 | 0.00133 |
| 37        | 0.00048  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.00089  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00018 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.00059  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.00445  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00057 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.00088  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.02680  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00434 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.00028  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00004 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.00019  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00010 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.00095  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.00014  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00027 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00006 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.00890  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.00026  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00149 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.00094  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.00105  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00021 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.00090  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.00374  | 0.00143 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00006 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.00457  | 0.00308 | 0.00000 | 0.00000 | 0.00000 | 0.00099 | 0.00014 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.00315  | 0.00160 | 0.00000 | 0.00000 | 0.00000 | 0.00004 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.02039  | 0.00666 | 0.00000 | 0.00000 | 0.00000 | 0.00460 | 0.00143 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.00418  | 0.00372 | 0.00000 | 0.00000 | 0.00000 | 0.00104 | 0.00032 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.01351  | 0.05075 | 0.00000 | 0.00000 | 0.00000 | 0.00522 | 0.02842 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.00118  | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.00011  | 0.00063 | 0.00000 | 0.00000 | 0.00000 | 0.00023 | 0.00133 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.00429  | 0.00421 | 0.00000 | 0.00000 | 0.00000 | 0.00004 | 0.00004 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.00079  | 0.00045 | 0.00000 | 0.00000 | 0.00000 | 0.00164 | 0.00146 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00014 | 0.00061 | 0.00000 | 0.00000 | 0.00000 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 11       | 12      | 13      | 14      | 15      | 16      | 17      | 18      | 19      | 20      |
| 64        | 0.03235  | 0.00751 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00037 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.00076  | 0.00153 | 0.00000 | 0.00000 | 0.00000 | 0.00933 | 0.00424 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.00368  | 0.00425 | 0.00000 | 0.00000 | 0.00000 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.00455  | 0.00337 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00147 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.00589  | 0.00054 | 0.00000 | 0.00000 | 0.00000 | 0.00246 | 0.00117 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00067  | 0.00771 | 0.00000 | 0.00000 | 0.00000 | 0.00010 | 0.00115 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.00595  | 0.00237 | 0.00177 | 0.00000 | 0.00000 | 0.00004 | 0.00011 | 0.00011 | 0.00000 | 0.00000 |
| 71        | 0.00763  | 0.00310 | 0.00423 | 0.00000 | 0.00000 | 0.00229 | 0.00040 | 0.00050 | 0.00000 | 0.00000 |
| 72        | 0.00526  | 0.00153 | 0.00176 | 0.00000 | 0.00000 | 0.00019 | 0.00001 | 0.00001 | 0.00000 | 0.00000 |
| 73        | 0.01331  | 0.00868 | 0.00882 | 0.00000 | 0.00000 | 0.00684 | 0.00198 | 0.00319 | 0.00000 | 0.00000 |
| 74        | 0.01225  | 0.00400 | 0.00678 | 0.00000 | 0.00000 | 0.00293 | 0.00086 | 0.00092 | 0.00000 | 0.00000 |
| 75        | 0.02130  | 0.01200 | 0.10317 | 0.00000 | 0.00000 | 0.00768 | 0.00391 | 0.05371 | 0.00000 | 0.00000 |
| 76        | 0.00124  | 0.00009 | 0.00000 | 0.00000 | 0.00000 | 0.00012 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 77        | 0.00024  | 0.00034 | 0.00090 | 0.00000 | 0.00000 | 0.00062 | 0.00014 | 0.00281 | 0.00000 | 0.00000 |
| 78        | 0.00874  | 0.00295 | 0.00573 | 0.00000 | 0.00000 | 0.00031 | 0.00007 | 0.00012 | 0.00000 | 0.00000 |
| 79        | 0.00015  | 0.00038 | 0.00054 | 0.00000 | 0.00000 | 0.00384 | 0.00097 | 0.00265 | 0.00000 | 0.00000 |
| 80        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00020 | 0.00011 | 0.00094 | 0.00000 | 0.00000 |
| 81        | 0.03525  | 0.01007 | 0.00813 | 0.00000 | 0.00000 | 0.00017 | 0.00017 | 0.00047 | 0.00000 | 0.00000 |
| 82        | 0.00082  | 0.00152 | 0.00123 | 0.00000 | 0.00000 | 0.01755 | 0.00476 | 0.00532 | 0.00000 | 0.00000 |
| 83        | 0.00549  | 0.00418 | 0.00439 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 84        | 0.00702  | 0.00310 | 0.00476 | 0.00000 | 0.00000 | 0.00152 | 0.00115 | 0.00079 | 0.00000 | 0.00000 |
| 85        | 0.00979  | 0.00023 | 0.00095 | 0.00000 | 0.00000 | 0.00473 | 0.00000 | 0.00252 | 0.00000 | 0.00000 |
| 86        | 0.00270  | 0.00805 | 0.00069 | 0.00000 | 0.00000 | 0.00075 | 0.00345 | 0.00058 | 0.00000 | 0.00000 |
| 87        | 0.00307  | 0.00055 | 0.00885 | 0.00000 | 0.00000 | 0.00032 | 0.00006 | 0.00235 | 0.00000 | 0.00000 |
| 88        | 0.01555  | 0.00492 | 0.00233 | 0.00118 | 0.00000 | 0.00040 | 0.00014 | 0.00016 | 0.00014 | 0.00000 |
| 89        | 0.01938  | 0.00532 | 0.00328 | 0.00661 | 0.00000 | 0.00477 | 0.00069 | 0.00112 | 0.00097 | 0.00000 |
| 90        | 0.01210  | 0.00294 | 0.00067 | 0.00158 | 0.00000 | 0.00063 | 0.00005 | 0.00004 | 0.00002 | 0.00000 |
| 91        | 0.02053  | 0.00993 | 0.00579 | 0.00608 | 0.00000 | 0.01135 | 0.00387 | 0.00336 | 0.00284 | 0.00000 |
| 92        | 0.02076  | 0.00756 | 0.00557 | 0.01182 | 0.00000 | 0.00450 | 0.00140 | 0.00134 | 0.00126 | 0.00000 |
| 93        | 0.02657  | 0.01842 | 0.02549 | 0.15046 | 0.00000 | 0.00990 | 0.00614 | 0.01034 | 0.08150 | 0.00000 |
| 94        | 0.00228  | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00025 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 95        | 0.00031  | 0.00024 | 0.00031 | 0.00062 | 0.00000 | 0.00124 | 0.00021 | 0.00057 | 0.00513 | 0.00000 |
| 96        | 0.01770  | 0.00641 | 0.00428 | 0.00908 | 0.00000 | 0.00065 | 0.00009 | 0.00007 | 0.00025 | 0.00000 |
| 97        | 0.00011  | 0.00010 | 0.00045 | 0.00068 | 0.00000 | 0.00430 | 0.00148 | 0.00069 | 0.00491 | 0.00000 |
| 98        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00045 | 0.00010 | 0.00032 | 0.00096 | 0.00000 |
| 99        | 0.04035  | 0.01467 | 0.00698 | 0.01377 | 0.00000 | 0.00040 | 0.00027 | 0.00041 | 0.00104 | 0.00000 |
| 100       | 0.00125  | 0.00228 | 0.00074 | 0.00049 | 0.00000 | 0.03240 | 0.00914 | 0.00439 | 0.00530 | 0.00000 |
| 101       | 0.01111  | 0.00746 | 0.00416 | 0.00611 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 102       | 0.01631  | 0.00621 | 0.00324 | 0.00777 | 0.00000 | 0.00221 | 0.00205 | 0.00094 | 0.00160 | 0.00000 |
| 103       | 0.00945  | 0.00030 | 0.00009 | 0.00163 | 0.00000 | 0.00695 | 0.00000 | 0.00052 | 0.00286 | 0.00000 |
| 104       | 0.00274  | 0.00995 | 0.00027 | 0.00104 | 0.00000 | 0.00095 | 0.00556 | 0.00000 | 0.00139 | 0.00000 |
| 105       | 0.00561  | 0.00208 | 0.00876 | 0.00053 | 0.00000 | 0.00404 | 0.00065 | 0.00491 | 0.00099 | 0.00000 |
| 106       | 0.01153  | 0.00261 | 0.00037 | 0.01533 | 0.00000 | 0.00112 | 0.00021 | 0.00005 | 0.00211 | 0.00000 |
| 107       | 0.01559  | 0.00767 | 0.00236 | 0.00095 | 0.00120 | 0.00043 | 0.00021 | 0.00009 | 0.00009 | 0.00009 |
| 108       | 0.01526  | 0.00845 | 0.00308 | 0.00489 | 0.00570 | 0.00491 | 0.00138 | 0.00167 | 0.00090 | 0.00084 |
| 109       | 0.01294  | 0.00570 | 0.00163 | 0.00027 | 0.00097 | 0.00090 | 0.00028 | 0.00003 | 0.00000 | 0.00001 |
| 110       | 0.01571  | 0.01304 | 0.00593 | 0.00412 | 0.00500 | 0.00793 | 0.00416 | 0.00291 | 0.00223 | 0.00276 |
| 111       | 0.02276  | 0.01022 | 0.01253 | 0.00693 | 0.01078 | 0.00361 | 0.00174 | 0.00186 | 0.00104 | 0.00093 |
| 112       | 0.02500  | 0.01736 | 0.02287 | 0.02777 | 0.14073 | 0.00968 | 0.00515 | 0.01004 | 0.01017 | 0.08536 |
| 113       | 0.00257  | 0.00003 | 0.00002 | 0.00000 | 0.00000 | 0.00037 | 0.00005 | 0.00000 | 0.00000 | 0.00000 |
| 114       | 0.00011  | 0.00027 | 0.00006 | 0.00022 | 0.00057 | 0.00122 | 0.00046 | 0.00066 | 0.00061 | 0.00814 |
| 115       | 0.01807  | 0.01018 | 0.00609 | 0.00620 | 0.00810 | 0.00065 | 0.00037 | 0.00011 | 0.00013 | 0.00019 |
| 116       | 0.00000  | 0.00006 | 0.00012 | 0.00037 | 0.00051 | 0.00258 | 0.00197 | 0.00158 | 0.00067 | 0.00735 |
| 117       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00045 | 0.00025 | 0.00035 | 0.00034 | 0.00183 |
| 118       | 0.03262  | 0.01854 | 0.00858 | 0.00805 | 0.01328 | 0.00035 | 0.00031 | 0.00045 | 0.00059 | 0.00106 |
| 119       | 0.00153  | 0.00243 | 0.00080 | 0.00032 | 0.00022 | 0.02894 | 0.01209 | 0.00587 | 0.00435 | 0.00531 |
| 120       | 0.00738  | 0.00841 | 0.00367 | 0.00337 | 0.00458 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 121       | 0.01210  | 0.00888 | 0.00376 | 0.00527 | 0.00575 | 0.00161 | 0.00231 | 0.00071 | 0.00047 | 0.00190 |
| 122       | 0.00849  | 0.00033 | 0.00008 | 0.00040 | 0.00132 | 0.00653 | 0.00000 | 0.00044 | 0.00036 | 0.00274 |
| 123       | 0.00242  | 0.00930 | 0.00028 | 0.00003 | 0.00176 | 0.00079 | 0.00610 | 0.00000 | 0.00052 | 0.00144 |
| 124       | 0.00466  | 0.00185 | 0.00929 | 0.00027 | 0.00152 | 0.00350 | 0.00046 | 0.00577 | 0.00000 | 0.00146 |
| 125       | 0.01020  | 0.00356 | 0.00159 | 0.00867 | 0.00150 | 0.00205 | 0.00248 | 0.00036 | 0.00478 | 0.00041 |
| 126       | 0.00580  | 0.00763 | 0.00177 | 0.00029 | 0.01518 | 0.00061 | 0.00086 | 0.00008 | 0.00005 | 0.00224 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 21       | 22      | 23      | 24      | 25      | 26      | 27      | 28      | 29      | 30      |
| 1         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 2         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 3         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 4         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 5         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 6         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 9         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 10        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 16        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17        | 0.00057  | 0.00014 | 0.00012 | 0.00002 | 0.00001 | 0.00053 | 0.00038 | 0.00040 | 0.00031 | 0.00067 |
| 18        | 0.00099  | 0.00040 | 0.00048 | 0.00017 | 0.00010 | 0.00010 | 0.00018 | 0.00034 | 0.00035 | 0.00082 |
| 19        | 0.00018  | 0.00006 | 0.00001 | 0.00000 | 0.00000 | 0.00005 | 0.00013 | 0.00029 | 0.00029 | 0.00065 |
| 20        | 0.00326  | 0.00123 | 0.00130 | 0.00038 | 0.00032 | 0.00178 | 0.00090 | 0.00089 | 0.00058 | 0.00088 |
| 21        | 0.00069  | 0.00034 | 0.00039 | 0.00018 | 0.00007 | 0.00009 | 0.00016 | 0.00038 | 0.00033 | 0.00078 |
| 22        | 0.00340  | 0.00366 | 0.00786 | 0.00562 | 0.01995 | 0.00055 | 0.00109 | 0.00204 | 0.00135 | 0.00599 |
| 23        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00004 | 0.00012 | 0.00029 | 0.00029 | 0.00065 |
| 24        | 0.00512  | 0.00485 | 0.00379 | 0.00152 | 0.00041 | 0.00213 | 0.00255 | 0.00231 | 0.00193 | 0.00181 |
| 25        | 0.00050  | 0.00022 | 0.00023 | 0.00010 | 0.00005 | 0.00009 | 0.00018 | 0.00034 | 0.00035 | 0.00082 |
| 26        | 0.00407  | 0.00150 | 0.00047 | 0.00019 | 0.00014 | 0.00253 | 0.00235 | 0.00207 | 0.00157 | 0.00135 |
| 27        | 0.00096  | 0.00067 | 0.00043 | 0.00026 | 0.00118 | 0.00244 | 0.00271 | 0.00242 | 0.00228 | 0.00172 |
| 28        | 0.00007  | 0.00020 | 0.00018 | 0.00009 | 0.00020 | 0.00005 | 0.00029 | 0.00068 | 0.00041 | 0.00141 |
| 29        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00008 | 0.00024 | 0.00023 | 0.00063 |
| 30        | 0.01608  | 0.00489 | 0.00293 | 0.00125 | 0.00182 | 0.00001 | 0.00007 | 0.00023 | 0.00023 | 0.00062 |
| 31        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00756 | 0.00229 | 0.00095 | 0.00069 | 0.00102 |
| 32        | 0.00515  | 0.00015 | 0.00068 | 0.00024 | 0.00045 | 0.00355 | 0.00047 | 0.00122 | 0.00095 | 0.00172 |
| 33        | 0.00488  | 0.00461 | 0.00028 | 0.00042 | 0.00040 | 0.00233 | 0.00310 | 0.00073 | 0.00091 | 0.00193 |
| 34        | 0.00414  | 0.00431 | 0.00425 | 0.00012 | 0.00055 | 0.00272 | 0.00166 | 0.00240 | 0.00041 | 0.00214 |
| 35        | 0.00151  | 0.00291 | 0.00269 | 0.00260 | 0.00047 | 0.00086 | 0.00203 | 0.00136 | 0.00175 | 0.00149 |
| 36        | 0.00052  | 0.00016 | 0.00033 | 0.00015 | 0.00046 | 0.00008 | 0.00013 | 0.00029 | 0.00027 | 0.00087 |
| 37        | 0.00001  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00009 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.00010  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.00004  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.00010  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.00151  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00032 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.00023  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00014 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.00010  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.00026  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00013 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00026  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00018 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.00014  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00006 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.00104  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00006 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00022 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.00010  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.00003  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00033 | 0.00040 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.00031  | 0.00024 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00048 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.00001  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00037 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.00112  | 0.00036 | 0.00000 | 0.00000 | 0.00000 | 0.00076 | 0.00062 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.00037  | 0.00025 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00050 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.00148  | 0.00965 | 0.00000 | 0.00000 | 0.00000 | 0.00036 | 0.00365 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00037 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.00192  | 0.00116 | 0.00000 | 0.00000 | 0.00000 | 0.00078 | 0.00134 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.00021  | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00048 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.00176  | 0.00061 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00085 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00032  | 0.00056 | 0.00000 | 0.00000 | 0.00000 | 0.00121 | 0.00117 | 0.00000 | 0.00000 | 0.00000 |



| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 21       | 22      | 23      | 24      | 25      | 26      | 27      | 28      | 29      | 30      |
| 64        | 0.00020  | 0.00055 | 0.00000 | 0.00000 | 0.00000 | 0.00006 | 0.00105 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00035 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.00409  | 0.00102 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00032 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00221 | 0.00071 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.00173  | 0.00033 | 0.00000 | 0.00000 | 0.00000 | 0.00104 | 0.00099 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00009  | 0.00035 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00057 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.00022  | 0.00000 | 0.00003 | 0.00000 | 0.00000 | 0.00050 | 0.00016 | 0.00093 | 0.00000 | 0.00000 |
| 71        | 0.00103  | 0.00020 | 0.00093 | 0.00000 | 0.00000 | 0.00017 | 0.00005 | 0.00099 | 0.00000 | 0.00000 |
| 72        | 0.00011  | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00004 | 0.00087 | 0.00000 | 0.00000 |
| 73        | 0.00200  | 0.00070 | 0.00059 | 0.00000 | 0.00000 | 0.00110 | 0.00044 | 0.00128 | 0.00000 | 0.00000 |
| 74        | 0.00077  | 0.00021 | 0.00039 | 0.00000 | 0.00000 | 0.00020 | 0.00005 | 0.00110 | 0.00000 | 0.00000 |
| 75        | 0.00259  | 0.00109 | 0.02195 | 0.00000 | 0.00000 | 0.00042 | 0.00023 | 0.00570 | 0.00000 | 0.00000 |
| 76        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00004 | 0.00087 | 0.00000 | 0.00000 |
| 77        | 0.00381  | 0.00324 | 0.00214 | 0.00000 | 0.00000 | 0.00131 | 0.00142 | 0.00241 | 0.00000 | 0.00000 |
| 78        | 0.00045  | 0.00007 | 0.00051 | 0.00000 | 0.00000 | 0.00012 | 0.00005 | 0.00099 | 0.00000 | 0.00000 |
| 79        | 0.00298  | 0.00108 | 0.00069 | 0.00000 | 0.00000 | 0.00149 | 0.00111 | 0.00172 | 0.00000 | 0.00000 |
| 80        | 0.00061  | 0.00026 | 0.00117 | 0.00000 | 0.00000 | 0.00135 | 0.00206 | 0.00221 | 0.00000 | 0.00000 |
| 81        | 0.00004  | 0.00018 | 0.00040 | 0.00000 | 0.00000 | 0.00006 | 0.00013 | 0.00218 | 0.00000 | 0.00000 |
| 82        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00086 | 0.00000 | 0.00000 |
| 83        | 0.00924  | 0.00218 | 0.00372 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00080 | 0.00000 | 0.00000 |
| 84        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00366 | 0.00114 | 0.00118 | 0.00000 | 0.00000 |
| 85        | 0.00371  | 0.00002 | 0.00081 | 0.00000 | 0.00000 | 0.00210 | 0.00017 | 0.00228 | 0.00000 | 0.00000 |
| 86        | 0.00239  | 0.00295 | 0.00064 | 0.00000 | 0.00000 | 0.00120 | 0.00171 | 0.00208 | 0.00000 | 0.00000 |
| 87        | 0.00002  | 0.00013 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00119 | 0.00000 | 0.00000 |
| 88        | 0.00065  | 0.00008 | 0.00011 | 0.00000 | 0.00000 | 0.00072 | 0.00033 | 0.00014 | 0.00186 | 0.00000 |
| 89        | 0.00116  | 0.00028 | 0.00028 | 0.00026 | 0.00000 | 0.00011 | 0.00009 | 0.00009 | 0.00203 | 0.00000 |
| 90        | 0.00024  | 0.00004 | 0.00001 | 0.00000 | 0.00000 | 0.00003 | 0.00006 | 0.00006 | 0.00181 | 0.00000 |
| 91        | 0.00403  | 0.00124 | 0.00108 | 0.00042 | 0.00000 | 0.00186 | 0.00074 | 0.00050 | 0.00223 | 0.00000 |
| 92        | 0.00066  | 0.00030 | 0.00023 | 0.00024 | 0.00000 | 0.00010 | 0.00009 | 0.00009 | 0.00204 | 0.00000 |
| 93        | 0.00311  | 0.00166 | 0.00208 | 0.03135 | 0.00000 | 0.00040 | 0.00031 | 0.00042 | 0.00802 | 0.00000 |
| 94        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00004 | 0.00006 | 0.00181 | 0.00000 |
| 95        | 0.00537  | 0.00515 | 0.00422 | 0.00143 | 0.00000 | 0.00176 | 0.00218 | 0.00247 | 0.00422 | 0.00000 |
| 96        | 0.00047  | 0.00010 | 0.00012 | 0.00006 | 0.00000 | 0.00011 | 0.00009 | 0.00009 | 0.00203 | 0.00000 |
| 97        | 0.00448  | 0.00131 | 0.00041 | 0.00039 | 0.00000 | 0.00249 | 0.00218 | 0.00211 | 0.00310 | 0.00000 |
| 98        | 0.00126  | 0.00035 | 0.00018 | 0.00171 | 0.00000 | 0.00221 | 0.00231 | 0.00321 | 0.00402 | 0.00000 |
| 99        | 0.00001  | 0.00010 | 0.00009 | 0.00043 | 0.00000 | 0.00000 | 0.00006 | 0.00003 | 0.00261 | 0.00000 |
| 100       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00181 | 0.00000 |
| 101       | 0.01689  | 0.00444 | 0.00175 | 0.00225 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00181 | 0.00000 |
| 102       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00730 | 0.00200 | 0.00071 | 0.00227 | 0.00000 |
| 103       | 0.00531  | 0.00012 | 0.00049 | 0.00083 | 0.00000 | 0.00353 | 0.00027 | 0.00059 | 0.00350 | 0.00000 |
| 104       | 0.00566  | 0.00540 | 0.00009 | 0.00103 | 0.00000 | 0.00253 | 0.00342 | 0.00014 | 0.00372 | 0.00000 |
| 105       | 0.00505  | 0.00438 | 0.00512 | 0.00096 | 0.00000 | 0.00341 | 0.00190 | 0.00206 | 0.00263 | 0.00000 |
| 106       | 0.00000  | 0.00008 | 0.00018 | 0.00071 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00229 | 0.00000 |
| 107       | 0.00071  | 0.00021 | 0.00016 | 0.00003 | 0.00001 | 0.00035 | 0.00034 | 0.00020 | 0.00013 | 0.00113 |
| 108       | 0.00080  | 0.00042 | 0.00030 | 0.00024 | 0.00017 | 0.00004 | 0.00007 | 0.00009 | 0.00016 | 0.00139 |
| 109       | 0.00020  | 0.00008 | 0.00002 | 0.00000 | 0.00000 | 0.00003 | 0.00005 | 0.00007 | 0.00009 | 0.00109 |
| 110       | 0.00324  | 0.00133 | 0.00168 | 0.00055 | 0.00055 | 0.00175 | 0.00092 | 0.00079 | 0.00050 | 0.00148 |
| 111       | 0.00049  | 0.00032 | 0.00042 | 0.00024 | 0.00012 | 0.00000 | 0.00004 | 0.00009 | 0.00012 | 0.00132 |
| 112       | 0.00298  | 0.00166 | 0.00227 | 0.00271 | 0.03368 | 0.00045 | 0.00028 | 0.00062 | 0.00055 | 0.01011 |
| 113       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00004 | 0.00005 | 0.00009 | 0.00109 |
| 114       | 0.00488  | 0.00509 | 0.00446 | 0.00226 | 0.00069 | 0.00227 | 0.00264 | 0.00220 | 0.00234 | 0.00305 |
| 115       | 0.00043  | 0.00026 | 0.00012 | 0.00016 | 0.00008 | 0.00004 | 0.00007 | 0.00009 | 0.00016 | 0.00139 |
| 116       | 0.00376  | 0.00150 | 0.00038 | 0.00024 | 0.00024 | 0.00273 | 0.00265 | 0.00220 | 0.00198 | 0.00227 |
| 117       | 0.00088  | 0.00072 | 0.00012 | 0.00007 | 0.00199 | 0.00250 | 0.00265 | 0.00232 | 0.00298 | 0.00290 |
| 118       | 0.00000  | 0.00002 | 0.00009 | 0.00005 | 0.00034 | 0.00001 | 0.00002 | 0.00009 | 0.00013 | 0.00237 |
| 119       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00107 |
| 120       | 0.01730  | 0.00586 | 0.00278 | 0.00163 | 0.00307 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00105 |
| 121       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00855 | 0.00262 | 0.00088 | 0.00067 | 0.00172 |
| 122       | 0.00509  | 0.00010 | 0.00065 | 0.00023 | 0.00076 | 0.00382 | 0.00028 | 0.00084 | 0.00085 | 0.00291 |
| 123       | 0.00581  | 0.00506 | 0.00015 | 0.00049 | 0.00067 | 0.00279 | 0.00346 | 0.00021 | 0.00073 | 0.00325 |
| 124       | 0.00575  | 0.00617 | 0.00582 | 0.00000 | 0.00093 | 0.00382 | 0.00220 | 0.00304 | 0.00013 | 0.00361 |
| 125       | 0.00192  | 0.00471 | 0.00405 | 0.00423 | 0.00080 | 0.00135 | 0.00322 | 0.00182 | 0.00245 | 0.00252 |
| 126       | 0.00000  | 0.00002 | 0.00005 | 0.00020 | 0.00077 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00147 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 31       | 32      | 33      | 34      | 35      | 36      | 37      | 38      | 39      | 40      |
| 1         | 0.02429  | 0.00406 | 0.00111 | 0.00000 | 0.00000 | 0.02173 | 0.00384 | 0.00142 | 0.00000 | 0.00000 |
| 2         | 0.08729  | 0.03092 | 0.01522 | 0.00000 | 0.00000 | 0.12178 | 0.11636 | 0.05588 | 0.00000 | 0.00000 |
| 3         | 0.01796  | 0.00381 | 0.00053 | 0.00000 | 0.00000 | 0.02009 | 0.00372 | 0.00046 | 0.00000 | 0.00000 |
| 4         | 0.03315  | 0.00403 | 0.00103 | 0.00000 | 0.00000 | 0.14955 | 0.11909 | 0.05683 | 0.00000 | 0.00000 |
| 5         | 0.08426  | 0.04253 | 0.01341 | 0.00000 | 0.00000 | 0.09273 | 0.11303 | 0.05894 | 0.00000 | 0.00000 |
| 6         | 0.00034  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00056 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.04624  | 0.00506 | 0.00299 | 0.00000 | 0.00000 | 0.18658 | 0.16602 | 0.07219 | 0.00000 | 0.00000 |
| 8         | 0.01873  | 0.00359 | 0.00054 | 0.00000 | 0.00000 | 0.01668 | 0.00322 | 0.00022 | 0.00000 | 0.00000 |
| 9         | 0.08668  | 0.03092 | 0.01522 | 0.00000 | 0.00000 | 0.12059 | 0.11597 | 0.04701 | 0.00000 | 0.00000 |
| 10        | 0.15242  | 0.14119 | 0.06778 | 0.00000 | 0.00000 | 0.00042 | 0.00016 | 0.00007 | 0.00000 | 0.00000 |
| 11        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.18716 | 0.15410 | 0.07718 | 0.00000 | 0.00000 |
| 12        | 0.05671  | 0.01355 | 0.00711 | 0.00000 | 0.00000 | 0.08909 | 0.06095 | 0.01136 | 0.00000 | 0.00000 |
| 13        | 0.08685  | 0.03092 | 0.01542 | 0.00000 | 0.00000 | 0.12179 | 0.11636 | 0.05468 | 0.00000 | 0.00000 |
| 14        | 0.06950  | 0.00231 | 0.00107 | 0.00000 | 0.00000 | 0.14420 | 0.00581 | 0.00341 | 0.00000 | 0.00000 |
| 15        | 0.05415  | 0.06502 | 0.00120 | 0.00000 | 0.00000 | 0.08132 | 0.15151 | 0.00459 | 0.00000 | 0.00000 |
| 16        | 0.07677  | 0.02833 | 0.03265 | 0.00000 | 0.00000 | 0.11382 | 0.09902 | 0.06957 | 0.00000 | 0.00000 |
| 17        | 0.00280  | 0.00073 | 0.00046 | 0.00043 | 0.00040 | 0.00915 | 0.00346 | 0.00302 | 0.00134 | 0.00177 |
| 18        | 0.00843  | 0.00683 | 0.00532 | 0.00295 | 0.00375 | 0.01067 | 0.01564 | 0.02352 | 0.01962 | 0.04067 |
| 19        | 0.00331  | 0.00151 | 0.00206 | 0.00124 | 0.00277 | 0.00950 | 0.00426 | 0.00279 | 0.00150 | 0.00153 |
| 20        | 0.00063  | 0.00044 | 0.00041 | 0.00022 | 0.00038 | 0.00820 | 0.00922 | 0.01085 | 0.00993 | 0.02210 |
| 21        | 0.00398  | 0.00532 | 0.00549 | 0.00348 | 0.00545 | 0.00553 | 0.01386 | 0.02315 | 0.02066 | 0.05628 |
| 22        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 23        | 0.01161  | 0.00682 | 0.00381 | 0.00185 | 0.00266 | 0.01758 | 0.02286 | 0.03098 | 0.02250 | 0.04594 |
| 24        | 0.00243  | 0.00700 | 0.00606 | 0.00197 | 0.00122 | 0.00781 | 0.01129 | 0.00707 | 0.00220 | 0.00134 |
| 25        | 0.00432  | 0.00345 | 0.00492 | 0.00324 | 0.00456 | 0.00931 | 0.01045 | 0.01132 | 0.00838 | 0.01238 |
| 26        | 0.02466  | 0.04176 | 0.06266 | 0.04400 | 0.10819 | 0.00008 | 0.00026 | 0.00038 | 0.00012 | 0.00008 |
| 27        | 0.00004  | 0.00004 | 0.00002 | 0.00001 | 0.00000 | 0.01330 | 0.02849 | 0.05115 | 0.03988 | 0.12023 |
| 28        | 0.00223  | 0.00092 | 0.00145 | 0.00083 | 0.00124 | 0.00123 | 0.00286 | 0.00170 | 0.00126 | 0.00124 |
| 29        | 0.00563  | 0.00400 | 0.00448 | 0.00235 | 0.00321 | 0.00736 | 0.02223 | 0.01969 | 0.01444 | 0.02641 |
| 30        | 0.00828  | 0.00648 | 0.00478 | 0.00231 | 0.00288 | 0.00836 | 0.01720 | 0.02474 | 0.02231 | 0.04448 |
| 31        | 0.00835  | 0.00644 | 0.00482 | 0.00237 | 0.00302 | 0.01063 | 0.01538 | 0.02331 | 0.01898 | 0.04050 |
| 32        | 0.00911  | 0.00051 | 0.00080 | 0.00048 | 0.00077 | 0.02153 | 0.00081 | 0.00209 | 0.00167 | 0.00416 |
| 33        | 0.00046  | 0.01311 | 0.00069 | 0.00048 | 0.00081 | 0.00273 | 0.03425 | 0.00177 | 0.00146 | 0.00491 |
| 34        | 0.00198  | 0.00199 | 0.01609 | 0.00041 | 0.00082 | 0.00387 | 0.01024 | 0.05175 | 0.00110 | 0.00447 |
| 35        | 0.00526  | 0.00335 | 0.00340 | 0.01214 | 0.00092 | 0.00748 | 0.01384 | 0.01998 | 0.04101 | 0.00390 |
| 36        | 0.00632  | 0.00495 | 0.00390 | 0.00153 | 0.02190 | 0.00878 | 0.01461 | 0.02131 | 0.01137 | 0.08666 |
| 37        | 0.06920  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.04167 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.16546  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25345 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.05395  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.04147 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.09229  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.30340 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.16546  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25262 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.00104  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00176 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.13521  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.33465 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.05671  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.03515 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.16492  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25083 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.33728  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00163 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00010  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.39880 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.11891  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.21626 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.16524  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25311 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.16554  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25363 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.16546  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25336 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.16555  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.25348 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.00241  | 0.00745 | 0.00000 | 0.00000 | 0.00000 | 0.01308 | 0.01093 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.04179  | 0.05199 | 0.00000 | 0.00000 | 0.00000 | 0.05915 | 0.19752 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.00299  | 0.00936 | 0.00000 | 0.00000 | 0.00000 | 0.01294 | 0.01117 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.00526  | 0.00714 | 0.00000 | 0.00000 | 0.00000 | 0.06761 | 0.17187 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.04127  | 0.07278 | 0.00000 | 0.00000 | 0.00000 | 0.01512 | 0.20937 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.00425  | 0.01084 | 0.00000 | 0.00000 | 0.00000 | 0.09941 | 0.27123 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.00308  | 0.01195 | 0.00000 | 0.00000 | 0.00000 | 0.01284 | 0.01435 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.03970  | 0.05204 | 0.00000 | 0.00000 | 0.00000 | 0.05772 | 0.17950 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.06503  | 0.26258 | 0.00000 | 0.00000 | 0.00000 | 0.00019 | 0.00141 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00011  | 0.00009 | 0.00000 | 0.00000 | 0.00000 | 0.08571 | 0.28884 | 0.00000 | 0.00000 | 0.00000 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 31       | 32      | 33      | 34      | 35      | 36      | 37      | 38      | 39      | 40      |
| 64        | 0.02192  | 0.02241 | 0.00000 | 0.00000 | 0.00000 | 0.02944 | 0.08821 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.04017  | 0.05246 | 0.00000 | 0.00000 | 0.00000 | 0.05821 | 0.19366 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.04169  | 0.05149 | 0.00000 | 0.00000 | 0.00000 | 0.05853 | 0.20163 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.04175  | 0.05131 | 0.00000 | 0.00000 | 0.00000 | 0.05905 | 0.19682 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.02264  | 0.00441 | 0.00000 | 0.00000 | 0.00000 | 0.08866 | 0.01226 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00003  | 0.10812 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.23593 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.00375  | 0.00098 | 0.00256 | 0.00000 | 0.00000 | 0.01081 | 0.00201 | 0.00676 | 0.00000 | 0.00000 |
| 71        | 0.02241  | 0.00795 | 0.03017 | 0.00000 | 0.00000 | 0.01817 | 0.03297 | 0.13720 | 0.00000 | 0.00000 |
| 72        | 0.00183  | 0.00107 | 0.00533 | 0.00000 | 0.00000 | 0.00837 | 0.00215 | 0.00549 | 0.00000 | 0.00000 |
| 73        | 0.00040  | 0.00066 | 0.00238 | 0.00000 | 0.00000 | 0.01934 | 0.03449 | 0.09768 | 0.00000 | 0.00000 |
| 74        | 0.01334  | 0.00711 | 0.03076 | 0.00000 | 0.00000 | 0.00783 | 0.01709 | 0.14858 | 0.00000 | 0.00000 |
| 75        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 76        | 0.01212  | 0.00267 | 0.00730 | 0.00000 | 0.00000 | 0.04280 | 0.04427 | 0.16554 | 0.00000 | 0.00000 |
| 77        | 0.00184  | 0.00710 | 0.00522 | 0.00000 | 0.00000 | 0.01146 | 0.00726 | 0.00572 | 0.00000 | 0.00000 |
| 78        | 0.01700  | 0.00542 | 0.03234 | 0.00000 | 0.00000 | 0.01652 | 0.03057 | 0.08763 | 0.00000 | 0.00000 |
| 79        | 0.04188  | 0.04805 | 0.22007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00144 | 0.00000 | 0.00000 |
| 80        | 0.00002  | 0.00007 | 0.00005 | 0.00000 | 0.00000 | 0.02393 | 0.03895 | 0.23423 | 0.00000 | 0.00000 |
| 81        | 0.00882  | 0.00184 | 0.01295 | 0.00000 | 0.00000 | 0.00161 | 0.01370 | 0.01841 | 0.00000 | 0.00000 |
| 82        | 0.01880  | 0.00609 | 0.03046 | 0.00000 | 0.00000 | 0.01635 | 0.03922 | 0.11905 | 0.00000 | 0.00000 |
| 83        | 0.02228  | 0.00778 | 0.02906 | 0.00000 | 0.00000 | 0.01674 | 0.03465 | 0.13962 | 0.00000 | 0.00000 |
| 84        | 0.02235  | 0.00777 | 0.02920 | 0.00000 | 0.00000 | 0.01813 | 0.03257 | 0.13650 | 0.00000 | 0.00000 |
| 85        | 0.01428  | 0.00032 | 0.00297 | 0.00000 | 0.00000 | 0.03592 | 0.00014 | 0.01070 | 0.00000 | 0.00000 |
| 86        | 0.00008  | 0.01673 | 0.00290 | 0.00000 | 0.00000 | 0.00123 | 0.06422 | 0.01165 | 0.00000 | 0.00000 |
| 87        | 0.00017  | 0.00004 | 0.06856 | 0.00000 | 0.00000 | 0.00067 | 0.00000 | 0.19150 | 0.00000 | 0.00000 |
| 88        | 0.00304  | 0.00007 | 0.00058 | 0.00159 | 0.00000 | 0.00855 | 0.00401 | 0.00113 | 0.00399 | 0.00000 |
| 89        | 0.00857  | 0.00653 | 0.00336 | 0.01167 | 0.00000 | 0.00815 | 0.00349 | 0.01032 | 0.08869 | 0.00000 |
| 90        | 0.00352  | 0.00104 | 0.00125 | 0.00578 | 0.00000 | 0.00982 | 0.00304 | 0.00207 | 0.00428 | 0.00000 |
| 91        | 0.00040  | 0.00028 | 0.00024 | 0.00092 | 0.00000 | 0.00689 | 0.00344 | 0.00682 | 0.03937 | 0.00000 |
| 92        | 0.00158  | 0.00365 | 0.00362 | 0.01500 | 0.00000 | 0.00448 | 0.00117 | 0.00491 | 0.11169 | 0.00000 |
| 93        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 94        | 0.01209  | 0.00836 | 0.00270 | 0.00522 | 0.00000 | 0.01678 | 0.01209 | 0.02001 | 0.09783 | 0.00000 |
| 95        | 0.00146  | 0.00442 | 0.00333 | 0.00353 | 0.00000 | 0.00572 | 0.00799 | 0.00452 | 0.00472 | 0.00000 |
| 96        | 0.00405  | 0.00187 | 0.00245 | 0.01784 | 0.00000 | 0.00687 | 0.00168 | 0.00650 | 0.03458 | 0.00000 |
| 97        | 0.02573  | 0.03221 | 0.03606 | 0.17819 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00091 | 0.00000 |
| 98        | 0.00001  | 0.00002 | 0.00001 | 0.00001 | 0.00000 | 0.00967 | 0.00988 | 0.02908 | 0.20540 | 0.00000 |
| 99        | 0.00194  | 0.00009 | 0.00117 | 0.00369 | 0.00000 | 0.00039 | 0.00026 | 0.00103 | 0.00452 | 0.00000 |
| 100       | 0.00528  | 0.00285 | 0.00239 | 0.01235 | 0.00000 | 0.00449 | 0.01304 | 0.00961 | 0.06355 | 0.00000 |
| 101       | 0.00841  | 0.00628 | 0.00309 | 0.00937 | 0.00000 | 0.00527 | 0.00327 | 0.01260 | 0.09456 | 0.00000 |
| 102       | 0.00848  | 0.00625 | 0.00309 | 0.00988 | 0.00000 | 0.00801 | 0.00332 | 0.01009 | 0.08719 | 0.00000 |
| 103       | 0.00891  | 0.00039 | 0.00038 | 0.00210 | 0.00000 | 0.02033 | 0.00004 | 0.00038 | 0.00808 | 0.00000 |
| 104       | 0.00026  | 0.01215 | 0.00047 | 0.00180 | 0.00000 | 0.00198 | 0.02599 | 0.00007 | 0.00912 | 0.00000 |
| 105       | 0.00009  | 0.00007 | 0.01245 | 0.00187 | 0.00000 | 0.00114 | 0.00002 | 0.04070 | 0.00816 | 0.00000 |
| 106       | 0.00169  | 0.00016 | 0.00001 | 0.03954 | 0.00000 | 0.00167 | 0.00001 | 0.00000 | 0.15216 | 0.00000 |
| 107       | 0.00211  | 0.00028 | 0.00006 | 0.00038 | 0.00067 | 0.00803 | 0.00212 | 0.00242 | 0.00141 | 0.00298 |
| 108       | 0.00454  | 0.00455 | 0.00252 | 0.00245 | 0.00632 | 0.00695 | 0.00252 | 0.00377 | 0.01393 | 0.06865 |
| 109       | 0.00284  | 0.00049 | 0.00096 | 0.00085 | 0.00468 | 0.00866 | 0.00344 | 0.00188 | 0.00161 | 0.00259 |
| 110       | 0.00027  | 0.00001 | 0.00010 | 0.00016 | 0.00064 | 0.00576 | 0.00220 | 0.00267 | 0.00824 | 0.03730 |
| 111       | 0.00135  | 0.00189 | 0.00144 | 0.00263 | 0.00920 | 0.00403 | 0.00126 | 0.00061 | 0.01072 | 0.09499 |
| 112       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 113       | 0.01035  | 0.00726 | 0.00407 | 0.00200 | 0.00449 | 0.01159 | 0.00830 | 0.01005 | 0.01681 | 0.07754 |
| 114       | 0.00169  | 0.00502 | 0.00732 | 0.00256 | 0.00206 | 0.00450 | 0.01029 | 0.00834 | 0.00269 | 0.00227 |
| 115       | 0.00172  | 0.00105 | 0.00101 | 0.00161 | 0.00769 | 0.00608 | 0.00186 | 0.00270 | 0.00666 | 0.02090 |
| 116       | 0.01479  | 0.02299 | 0.03132 | 0.03573 | 0.18262 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00013 |
| 117       | 0.00000  | 0.00000 | 0.00001 | 0.00001 | 0.00000 | 0.00819 | 0.00734 | 0.01203 | 0.02289 | 0.20294 |
| 118       | 0.00079  | 0.00003 | 0.00006 | 0.00060 | 0.00210 | 0.00011 | 0.00011 | 0.00034 | 0.00114 | 0.00209 |
| 119       | 0.00265  | 0.00129 | 0.00129 | 0.00129 | 0.00542 | 0.00346 | 0.01003 | 0.00549 | 0.01063 | 0.04458 |
| 120       | 0.00440  | 0.00429 | 0.00220 | 0.00187 | 0.00487 | 0.00458 | 0.00286 | 0.00418 | 0.01720 | 0.07507 |
| 121       | 0.00447  | 0.00429 | 0.00220 | 0.00187 | 0.00510 | 0.00699 | 0.00258 | 0.00380 | 0.01318 | 0.06836 |
| 122       | 0.00646  | 0.00029 | 0.00046 | 0.00036 | 0.00129 | 0.01594 | 0.00003 | 0.00027 | 0.00106 | 0.00702 |
| 123       | 0.00023  | 0.00824 | 0.00037 | 0.00042 | 0.00137 | 0.00197 | 0.01986 | 0.00003 | 0.00049 | 0.00829 |
| 124       | 0.00009  | 0.00008 | 0.01035 | 0.00029 | 0.00138 | 0.00081 | 0.00001 | 0.02666 | 0.00009 | 0.00754 |
| 125       | 0.00067  | 0.00005 | 0.00002 | 0.01194 | 0.00155 | 0.00298 | 0.00024 | 0.00002 | 0.03630 | 0.00658 |
| 126       | 0.00097  | 0.00138 | 0.00014 | 0.00005 | 0.03696 | 0.00377 | 0.00080 | 0.00003 | 0.00000 | 0.14627 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 41       | 42      | 43      | 44      | 45      | 46      | 47      | 48      | 49      | 50      |
| 1         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 2         | 0.00073  | 0.00001 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 3         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 4         | 0.03465  | 0.01455 | 0.00781 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 5         | 0.00000  | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 6         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.00641  | 0.00048 | 0.00333 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 9         | 0.00072  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 10        | 0.00104  | 0.00017 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12        | 0.09783  | 0.02809 | 0.00822 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13        | 0.00072  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14        | 0.03147  | 0.00381 | 0.00170 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15        | 0.00000  | 0.01528 | 0.00065 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 16        | 0.00072  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17        | 0.01975  | 0.01738 | 0.01774 | 0.01660 | 0.02387 | 0.00194 | 0.00342 | 0.00350 | 0.00298 | 0.01062 |
| 18        | 0.02882  | 0.03058 | 0.04081 | 0.04035 | 0.11138 | 0.00879 | 0.01075 | 0.01832 | 0.01450 | 0.05103 |
| 19        | 0.02289  | 0.01766 | 0.01776 | 0.01608 | 0.02342 | 0.00170 | 0.00319 | 0.00533 | 0.00540 | 0.01655 |
| 20        | 0.02392  | 0.02606 | 0.03564 | 0.02778 | 0.06232 | 0.01877 | 0.01797 | 0.02888 | 0.02585 | 0.05557 |
| 21        | 0.01350  | 0.01640 | 0.04020 | 0.03561 | 0.11975 | 0.00319 | 0.00876 | 0.01466 | 0.01267 | 0.04398 |
| 22        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 23        | 0.04925  | 0.04715 | 0.05654 | 0.04182 | 0.10957 | 0.01327 | 0.01598 | 0.01896 | 0.01507 | 0.04719 |
| 24        | 0.00023  | 0.00053 | 0.00077 | 0.00031 | 0.00088 | 0.00073 | 0.00184 | 0.00385 | 0.00291 | 0.01182 |
| 25        | 0.01639  | 0.01158 | 0.01311 | 0.01501 | 0.02824 | 0.00061 | 0.00196 | 0.00282 | 0.00205 | 0.00698 |
| 26        | 0.00028  | 0.00162 | 0.00330 | 0.00230 | 0.00738 | 0.00145 | 0.00395 | 0.00454 | 0.00344 | 0.01079 |
| 27        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00060 | 0.00131 | 0.00128 | 0.00685 |
| 28        | 0.05939  | 0.06708 | 0.08687 | 0.06668 | 0.18181 | 0.00007 | 0.00098 | 0.00247 | 0.00211 | 0.00957 |
| 29        | 0.00003  | 0.00001 | 0.00051 | 0.00003 | 0.00009 | 0.06186 | 0.05050 | 0.06592 | 0.04814 | 0.11795 |
| 30        | 0.01274  | 0.02474 | 0.03112 | 0.02368 | 0.04515 | 0.00011 | 0.00006 | 0.00006 | 0.00004 | 0.00000 |
| 31        | 0.02747  | 0.03046 | 0.04009 | 0.03699 | 0.09911 | 0.00437 | 0.00700 | 0.01012 | 0.00823 | 0.01312 |
| 32        | 0.04310  | 0.00287 | 0.00641 | 0.00492 | 0.00940 | 0.03001 | 0.00199 | 0.00666 | 0.00562 | 0.01484 |
| 33        | 0.00238  | 0.06370 | 0.00463 | 0.00376 | 0.00951 | 0.00063 | 0.03460 | 0.00374 | 0.00408 | 0.01289 |
| 34        | 0.00701  | 0.01325 | 0.06896 | 0.00288 | 0.00858 | 0.00202 | 0.00631 | 0.03680 | 0.00196 | 0.01121 |
| 35        | 0.01244  | 0.02057 | 0.03146 | 0.05381 | 0.00756 | 0.00465 | 0.00815 | 0.01534 | 0.02756 | 0.00795 |
| 36        | 0.01634  | 0.01951 | 0.03316 | 0.02391 | 0.08668 | 0.00561 | 0.00843 | 0.01625 | 0.01057 | 0.04210 |
| 37        | 0.00305  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00102 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.00667  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00549 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.00316  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00112 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.02871  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00687 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.00700  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00529 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.00002  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.01639  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00375 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.00092  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00194 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.00290  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00120 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.00255  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00194 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.07512  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.00002  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01265 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.00590  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00056 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.00676  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00360 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.00706  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00455 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.00736  | 0.01748 | 0.00000 | 0.00000 | 0.00000 | 0.00155 | 0.00629 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.01260  | 0.05280 | 0.00000 | 0.00000 | 0.00000 | 0.00601 | 0.02824 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.00807  | 0.01692 | 0.00000 | 0.00000 | 0.00000 | 0.00170 | 0.00762 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.04262  | 0.05992 | 0.00000 | 0.00000 | 0.00000 | 0.01073 | 0.04180 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.00772  | 0.05441 | 0.00000 | 0.00000 | 0.00000 | 0.00257 | 0.03005 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.01775  | 0.07571 | 0.00000 | 0.00000 | 0.00000 | 0.00838 | 0.02648 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.00049  | 0.00149 | 0.00000 | 0.00000 | 0.00000 | 0.00065 | 0.00649 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.00838  | 0.01640 | 0.00000 | 0.00000 | 0.00000 | 0.00071 | 0.00437 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.00078  | 0.00664 | 0.00000 | 0.00000 | 0.00000 | 0.00344 | 0.01220 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00257 | 0.00000 | 0.00000 | 0.00000 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 41       | 42      | 43      | 44      | 45      | 46      | 47      | 48      | 49      | 50      |
| 64        | 0.10237  | 0.13765 | 0.00000 | 0.00000 | 0.00000 | 0.00009 | 0.00368 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.00120  | 0.00004 | 0.00000 | 0.00000 | 0.00000 | 0.02453 | 0.07603 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.00805  | 0.04247 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00026 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.01232  | 0.05522 | 0.00000 | 0.00000 | 0.00000 | 0.00429 | 0.01727 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.03780  | 0.00922 | 0.00000 | 0.00000 | 0.00000 | 0.01127 | 0.00689 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00000  | 0.06981 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.02970 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.01291  | 0.01312 | 0.02915 | 0.00000 | 0.00000 | 0.00043 | 0.00222 | 0.00986 | 0.00000 | 0.00000 |
| 71        | 0.01915  | 0.01885 | 0.10676 | 0.00000 | 0.00000 | 0.00785 | 0.00581 | 0.05387 | 0.00000 | 0.00000 |
| 72        | 0.01407  | 0.01280 | 0.02786 | 0.00000 | 0.00000 | 0.00086 | 0.00220 | 0.01547 | 0.00000 | 0.00000 |
| 73        | 0.02537  | 0.02394 | 0.09501 | 0.00000 | 0.00000 | 0.01268 | 0.01049 | 0.07388 | 0.00000 | 0.00000 |
| 74        | 0.01202  | 0.00671 | 0.12129 | 0.00000 | 0.00000 | 0.00200 | 0.00460 | 0.04775 | 0.00000 | 0.00000 |
| 75        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 76        | 0.03964  | 0.02113 | 0.13795 | 0.00000 | 0.00000 | 0.01241 | 0.01521 | 0.04282 | 0.00000 | 0.00000 |
| 77        | 0.00004  | 0.00060 | 0.00241 | 0.00000 | 0.00000 | 0.00056 | 0.00040 | 0.01266 | 0.00000 | 0.00000 |
| 78        | 0.01153  | 0.01081 | 0.02733 | 0.00000 | 0.00000 | 0.00031 | 0.00177 | 0.00923 | 0.00000 | 0.00000 |
| 79        | 0.00059  | 0.00055 | 0.01086 | 0.00000 | 0.00000 | 0.00071 | 0.00364 | 0.01359 | 0.00000 | 0.00000 |
| 80        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00003 | 0.00437 | 0.00000 | 0.00000 |
| 81        | 0.09002  | 0.06714 | 0.21985 | 0.00000 | 0.00000 | 0.00012 | 0.00049 | 0.00833 | 0.00000 | 0.00000 |
| 82        | 0.00000  | 0.00000 | 0.00179 | 0.00000 | 0.00000 | 0.04490 | 0.03057 | 0.13329 | 0.00000 | 0.00000 |
| 83        | 0.01036  | 0.01425 | 0.08149 | 0.00000 | 0.00000 | 0.00001 | 0.00001 | 0.00023 | 0.00000 | 0.00000 |
| 84        | 0.01854  | 0.01816 | 0.10598 | 0.00000 | 0.00000 | 0.00562 | 0.00454 | 0.03013 | 0.00000 | 0.00000 |
| 85        | 0.05492  | 0.00474 | 0.01691 | 0.00000 | 0.00000 | 0.02001 | 0.00042 | 0.01518 | 0.00000 | 0.00000 |
| 86        | 0.00115  | 0.05886 | 0.01301 | 0.00000 | 0.00000 | 0.00000 | 0.02125 | 0.01179 | 0.00000 | 0.00000 |
| 87        | 0.00012  | 0.00000 | 0.10523 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.05056 | 0.00000 | 0.00000 |
| 88        | 0.01979  | 0.01186 | 0.01628 | 0.04465 | 0.00000 | 0.00190 | 0.00130 | 0.00137 | 0.01634 | 0.00000 |
| 89        | 0.02918  | 0.01132 | 0.02137 | 0.18655 | 0.00000 | 0.00781 | 0.00341 | 0.00698 | 0.08252 | 0.00000 |
| 90        | 0.02301  | 0.01527 | 0.01540 | 0.04159 | 0.00000 | 0.00143 | 0.00181 | 0.00146 | 0.02672 | 0.00000 |
| 91        | 0.01952  | 0.01495 | 0.02695 | 0.10826 | 0.00000 | 0.01786 | 0.00555 | 0.01825 | 0.10669 | 0.00000 |
| 92        | 0.01322  | 0.00464 | 0.01150 | 0.19821 | 0.00000 | 0.00217 | 0.00022 | 0.00469 | 0.07773 | 0.00000 |
| 93        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 94        | 0.05148  | 0.03233 | 0.02683 | 0.19300 | 0.00000 | 0.00856 | 0.01090 | 0.01557 | 0.06787 | 0.00000 |
| 95        | 0.00000  | 0.00017 | 0.00018 | 0.00203 | 0.00000 | 0.00067 | 0.00032 | 0.00059 | 0.01911 | 0.00000 |
| 96        | 0.01650  | 0.00667 | 0.01137 | 0.04682 | 0.00000 | 0.00061 | 0.00065 | 0.00077 | 0.01109 | 0.00000 |
| 97        | 0.00000  | 0.00029 | 0.00060 | 0.01509 | 0.00000 | 0.00044 | 0.00039 | 0.00239 | 0.01646 | 0.00000 |
| 98        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00001 | 0.00013 | 0.00869 | 0.00000 |
| 99        | 0.05950  | 0.03190 | 0.05351 | 0.31423 | 0.00000 | 0.00001 | 0.00007 | 0.00056 | 0.01401 | 0.00000 |
| 100       | 0.00001  | 0.00000 | 0.00000 | 0.00023 | 0.00000 | 0.06083 | 0.02938 | 0.04592 | 0.20104 | 0.00000 |
| 101       | 0.01262  | 0.01193 | 0.01503 | 0.10396 | 0.00000 | 0.00004 | 0.00000 | 0.00000 | 0.00028 | 0.00000 |
| 102       | 0.02773  | 0.01156 | 0.02103 | 0.17080 | 0.00000 | 0.00495 | 0.00264 | 0.00508 | 0.03607 | 0.00000 |
| 103       | 0.04468  | 0.00153 | 0.00386 | 0.01968 | 0.00000 | 0.02940 | 0.00053 | 0.00335 | 0.02321 | 0.00000 |
| 104       | 0.00185  | 0.05602 | 0.00226 | 0.01748 | 0.00000 | 0.00002 | 0.02990 | 0.00059 | 0.01840 | 0.00000 |
| 105       | 0.00585  | 0.00181 | 0.05905 | 0.01375 | 0.00000 | 0.00032 | 0.00002 | 0.03219 | 0.01519 | 0.00000 |
| 106       | 0.00197  | 0.00003 | 0.00000 | 0.15119 | 0.00000 | 0.00025 | 0.00000 | 0.00000 | 0.07570 | 0.00000 |
| 107       | 0.01951  | 0.01389 | 0.01243 | 0.01837 | 0.04030 | 0.00188 | 0.00207 | 0.00088 | 0.00149 | 0.01792 |
| 108       | 0.02755  | 0.02027 | 0.01299 | 0.02775 | 0.18800 | 0.00606 | 0.00399 | 0.00355 | 0.00663 | 0.08613 |
| 109       | 0.02326  | 0.01398 | 0.01327 | 0.01814 | 0.03954 | 0.00129 | 0.00107 | 0.00126 | 0.00334 | 0.02794 |
| 110       | 0.02241  | 0.01510 | 0.01320 | 0.02348 | 0.10520 | 0.01638 | 0.00838 | 0.00932 | 0.02055 | 0.09380 |
| 111       | 0.00991  | 0.00302 | 0.00712 | 0.01723 | 0.20213 | 0.00197 | 0.00123 | 0.00080 | 0.00457 | 0.07423 |
| 112       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 113       | 0.04684  | 0.03424 | 0.02531 | 0.02884 | 0.18494 | 0.01072 | 0.00735 | 0.00807 | 0.01075 | 0.07964 |
| 114       | 0.00000  | 0.00001 | 0.00010 | 0.00009 | 0.00148 | 0.00020 | 0.00039 | 0.00030 | 0.00079 | 0.01995 |
| 115       | 0.01528  | 0.00675 | 0.00655 | 0.01520 | 0.04766 | 0.00025 | 0.00068 | 0.00017 | 0.00105 | 0.01178 |
| 116       | 0.00000  | 0.00001 | 0.00023 | 0.00062 | 0.01246 | 0.00035 | 0.00024 | 0.00063 | 0.00224 | 0.01822 |
| 117       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00004 | 0.00009 | 0.00027 | 0.01156 |
| 118       | 0.04762  | 0.03848 | 0.03421 | 0.04459 | 0.30687 | 0.00001 | 0.00003 | 0.00004 | 0.00054 | 0.01616 |
| 119       | 0.00001  | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.05834 | 0.03561 | 0.03733 | 0.03776 | 0.19908 |
| 120       | 0.01006  | 0.01636 | 0.01016 | 0.01748 | 0.07620 | 0.00003 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 121       | 0.02596  | 0.01943 | 0.01224 | 0.02550 | 0.16728 | 0.00130 | 0.00257 | 0.00151 | 0.00608 | 0.02214 |
| 122       | 0.03927  | 0.00096 | 0.00285 | 0.00404 | 0.01586 | 0.02958 | 0.00045 | 0.00322 | 0.00447 | 0.02504 |
| 123       | 0.00179  | 0.04968 | 0.00146 | 0.00257 | 0.01605 | 0.00001 | 0.03056 | 0.00053 | 0.00290 | 0.02176 |
| 124       | 0.00491  | 0.00212 | 0.05310 | 0.00188 | 0.01448 | 0.00001 | 0.00002 | 0.03088 | 0.00003 | 0.01892 |
| 125       | 0.00578  | 0.00581 | 0.00184 | 0.05813 | 0.01277 | 0.00071 | 0.00034 | 0.00003 | 0.03014 | 0.01342 |
| 126       | 0.00647  | 0.00158 | 0.00009 | 0.00000 | 0.14631 | 0.00070 | 0.00009 | 0.00005 | 0.00000 | 0.07106 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 51       | 52      | 53      | 54      | 55      | 56      | 57      | 58      | 59      | 60      |
| 1         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 2         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 3         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 4         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 5         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 6         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 9         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 10        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 16        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17        | 0.00127  | 0.00172 | 0.00279 | 0.00268 | 0.01213 | 0.00107 | 0.00292 | 0.00482 | 0.00308 | 0.01043 |
| 18        | 0.00111  | 0.00287 | 0.00500 | 0.00335 | 0.01115 | 0.00014 | 0.00061 | 0.00148 | 0.00079 | 0.00357 |
| 19        | 0.00082  | 0.00152 | 0.00271 | 0.00174 | 0.00594 | 0.00018 | 0.00089 | 0.00127 | 0.00091 | 0.00396 |
| 20        | 0.00616  | 0.00662 | 0.00994 | 0.01066 | 0.02631 | 0.00227 | 0.00446 | 0.00851 | 0.00575 | 0.02031 |
| 21        | 0.00047  | 0.00284 | 0.00492 | 0.00246 | 0.00876 | 0.00010 | 0.00052 | 0.00121 | 0.00074 | 0.00323 |
| 22        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00008 | 0.00020 | 0.00021 | 0.00057 |
| 23        | 0.00077  | 0.00129 | 0.00136 | 0.00099 | 0.00372 | 0.00024 | 0.00116 | 0.00189 | 0.00118 | 0.00567 |
| 24        | 0.00046  | 0.00183 | 0.00272 | 0.00122 | 0.00348 | 0.00129 | 0.00083 | 0.00166 | 0.00090 | 0.00364 |
| 25        | 0.00046  | 0.00182 | 0.00278 | 0.00182 | 0.00703 | 0.00012 | 0.00061 | 0.00136 | 0.00079 | 0.00356 |
| 26        | 0.00007  | 0.00038 | 0.00056 | 0.00007 | 0.00020 | 0.00001 | 0.00015 | 0.00046 | 0.00035 | 0.00112 |
| 27        | 0.00075  | 0.00228 | 0.00355 | 0.00219 | 0.00582 | 0.00090 | 0.00197 | 0.00359 | 0.00287 | 0.00707 |
| 28        | 0.00007  | 0.00045 | 0.00057 | 0.00018 | 0.00056 | 0.00013 | 0.00065 | 0.00127 | 0.00057 | 0.00259 |
| 29        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00021 | 0.00058 | 0.00031 | 0.00124 |
| 30        | 0.03935  | 0.02873 | 0.03629 | 0.02523 | 0.05464 | 0.00001 | 0.00012 | 0.00035 | 0.00031 | 0.00105 |
| 31        | 0.00004  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01752 | 0.01414 | 0.01656 | 0.01051 | 0.02340 |
| 32        | 0.04214  | 0.00594 | 0.00982 | 0.00571 | 0.01191 | 0.02265 | 0.00480 | 0.01034 | 0.00589 | 0.01107 |
| 33        | 0.00058  | 0.03984 | 0.00679 | 0.00716 | 0.01149 | 0.00103 | 0.02076 | 0.00594 | 0.00708 | 0.01191 |
| 34        | 0.00145  | 0.00247 | 0.03249 | 0.00423 | 0.01193 | 0.00066 | 0.00133 | 0.01609 | 0.00383 | 0.01272 |
| 35        | 0.00088  | 0.00290 | 0.00452 | 0.02186 | 0.01003 | 0.00038 | 0.00107 | 0.00211 | 0.01182 | 0.01121 |
| 36        | 0.00075  | 0.00276 | 0.00461 | 0.00256 | 0.01139 | 0.00014 | 0.00060 | 0.00143 | 0.00070 | 0.00348 |
| 37        | 0.00113  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00114 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.00106  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.00054  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.00282  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00155 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.00104  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.00045  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.00057  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00077 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.00051  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.00047  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.00103  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00137 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.00042  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00077 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.00558  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.00038  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00255 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.00147  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.00049  | 0.00403 | 0.00000 | 0.00000 | 0.00000 | 0.00138 | 0.00870 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.00131  | 0.01022 | 0.00000 | 0.00000 | 0.00000 | 0.00020 | 0.00249 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.00062  | 0.00476 | 0.00000 | 0.00000 | 0.00000 | 0.00037 | 0.00328 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.00469  | 0.01695 | 0.00000 | 0.00000 | 0.00000 | 0.00163 | 0.01239 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.00074  | 0.01111 | 0.00000 | 0.00000 | 0.00000 | 0.00010 | 0.00225 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00034 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.00035  | 0.00432 | 0.00000 | 0.00000 | 0.00000 | 0.00017 | 0.00305 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.00082  | 0.00666 | 0.00000 | 0.00000 | 0.00000 | 0.00042 | 0.00366 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.00075  | 0.00664 | 0.00000 | 0.00000 | 0.00000 | 0.00020 | 0.00249 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.00009  | 0.00141 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00068 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00119  | 0.00703 | 0.00000 | 0.00000 | 0.00000 | 0.00088 | 0.00699 | 0.00000 | 0.00000 | 0.00000 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 51       | 52      | 53      | 54      | 55      | 56      | 57      | 58      | 59      | 60      |
| 64        | 0.00010  | 0.00178 | 0.00000 | 0.00000 | 0.00000 | 0.00020 | 0.00286 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00092 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.01346  | 0.03617 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00055 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00489 | 0.01552 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.01246  | 0.00683 | 0.00000 | 0.00000 | 0.00000 | 0.00694 | 0.00758 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00000  | 0.00756 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00345 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.00053  | 0.00064 | 0.00788 | 0.00000 | 0.00000 | 0.00126 | 0.00130 | 0.01492 | 0.00000 | 0.00000 |
| 71        | 0.00073  | 0.00156 | 0.01565 | 0.00000 | 0.00000 | 0.00012 | 0.00009 | 0.00498 | 0.00000 | 0.00000 |
| 72        | 0.00081  | 0.00077 | 0.00872 | 0.00000 | 0.00000 | 0.00006 | 0.00020 | 0.00410 | 0.00000 | 0.00000 |
| 73        | 0.00341  | 0.00502 | 0.02514 | 0.00000 | 0.00000 | 0.00264 | 0.00238 | 0.02290 | 0.00000 | 0.00000 |
| 74        | 0.00017  | 0.00119 | 0.01669 | 0.00000 | 0.00000 | 0.00000 | 0.00004 | 0.00412 | 0.00000 | 0.00000 |
| 75        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00072 | 0.00000 | 0.00000 |
| 76        | 0.00035  | 0.00026 | 0.00448 | 0.00000 | 0.00000 | 0.00034 | 0.00032 | 0.00566 | 0.00000 | 0.00000 |
| 77        | 0.00033  | 0.00091 | 0.00872 | 0.00000 | 0.00000 | 0.00092 | 0.00007 | 0.00580 | 0.00000 | 0.00000 |
| 78        | 0.00025  | 0.00095 | 0.00907 | 0.00000 | 0.00000 | 0.00007 | 0.00009 | 0.00458 | 0.00000 | 0.00000 |
| 79        | 0.00000  | 0.00024 | 0.00191 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00162 | 0.00000 | 0.00000 |
| 80        | 0.00087  | 0.00170 | 0.01011 | 0.00000 | 0.00000 | 0.00071 | 0.00139 | 0.01172 | 0.00000 | 0.00000 |
| 81        | 0.00001  | 0.00017 | 0.00199 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00445 | 0.00000 | 0.00000 |
| 82        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00205 | 0.00000 | 0.00000 |
| 83        | 0.02457  | 0.01491 | 0.06140 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00122 | 0.00000 | 0.00000 |
| 84        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01010 | 0.00638 | 0.02689 | 0.00000 | 0.00000 |
| 85        | 0.02596  | 0.00274 | 0.01515 | 0.00000 | 0.00000 | 0.01242 | 0.00158 | 0.01505 | 0.00000 | 0.00000 |
| 86        | 0.00015  | 0.02383 | 0.01354 | 0.00000 | 0.00000 | 0.00072 | 0.01221 | 0.01309 | 0.00000 | 0.00000 |
| 87        | 0.00000  | 0.00000 | 0.01611 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00529 | 0.00000 | 0.00000 |
| 88        | 0.00119  | 0.00029 | 0.00089 | 0.01442 | 0.00000 | 0.00072 | 0.00111 | 0.00155 | 0.01877 | 0.00000 |
| 89        | 0.00107  | 0.00029 | 0.00124 | 0.02001 | 0.00000 | 0.00000 | 0.00013 | 0.00009 | 0.00545 | 0.00000 |
| 90        | 0.00054  | 0.00029 | 0.00065 | 0.00904 | 0.00000 | 0.00001 | 0.00018 | 0.00018 | 0.00542 | 0.00000 |
| 91        | 0.00498  | 0.00186 | 0.00598 | 0.04463 | 0.00000 | 0.00143 | 0.00181 | 0.00310 | 0.03053 | 0.00000 |
| 92        | 0.00026  | 0.00015 | 0.00065 | 0.01558 | 0.00000 | 0.00000 | 0.00004 | 0.00007 | 0.00524 | 0.00000 |
| 93        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00165 | 0.00000 |
| 94        | 0.00081  | 0.00015 | 0.00007 | 0.00620 | 0.00000 | 0.00015 | 0.00046 | 0.00051 | 0.00715 | 0.00000 |
| 95        | 0.00025  | 0.00029 | 0.00057 | 0.00831 | 0.00000 | 0.00168 | 0.00000 | 0.00004 | 0.00664 | 0.00000 |
| 96        | 0.00016  | 0.00020 | 0.00050 | 0.01022 | 0.00000 | 0.00000 | 0.00013 | 0.00009 | 0.00545 | 0.00000 |
| 97        | 0.00000  | 0.00000 | 0.00009 | 0.00049 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00271 | 0.00000 |
| 98        | 0.00026  | 0.00103 | 0.00133 | 0.01029 | 0.00000 | 0.00062 | 0.00009 | 0.00139 | 0.01482 | 0.00000 |
| 99        | 0.00000  | 0.00003 | 0.00002 | 0.00123 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00447 | 0.00000 |
| 100       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00245 | 0.00000 |
| 101       | 0.03976  | 0.01895 | 0.02677 | 0.09050 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00245 | 0.00000 |
| 102       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01657 | 0.01035 | 0.01149 | 0.03685 | 0.00000 |
| 103       | 0.04305  | 0.00503 | 0.00784 | 0.01950 | 0.00000 | 0.02140 | 0.00297 | 0.00815 | 0.01790 | 0.00000 |
| 104       | 0.00058  | 0.04089 | 0.00391 | 0.02139 | 0.00000 | 0.00089 | 0.01976 | 0.00252 | 0.02255 | 0.00000 |
| 105       | 0.00050  | 0.00006 | 0.03485 | 0.01722 | 0.00000 | 0.00069 | 0.00118 | 0.01913 | 0.01769 | 0.00000 |
| 106       | 0.00000  | 0.00000 | 0.00000 | 0.01455 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00759 | 0.00000 |
| 107       | 0.00123  | 0.00101 | 0.00073 | 0.00140 | 0.02047 | 0.00031 | 0.00080 | 0.00064 | 0.00113 | 0.01761 |
| 108       | 0.00060  | 0.00019 | 0.00067 | 0.00132 | 0.01883 | 0.00000 | 0.00001 | 0.00009 | 0.00016 | 0.00603 |
| 109       | 0.00055  | 0.00035 | 0.00024 | 0.00098 | 0.01002 | 0.00000 | 0.00014 | 0.00013 | 0.00036 | 0.00669 |
| 110       | 0.00537  | 0.00198 | 0.00341 | 0.00834 | 0.04442 | 0.00135 | 0.00133 | 0.00270 | 0.00309 | 0.03427 |
| 111       | 0.00018  | 0.00001 | 0.00016 | 0.00079 | 0.01479 | 0.00000 | 0.00000 | 0.00005 | 0.00011 | 0.00546 |
| 112       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00097 |
| 113       | 0.00074  | 0.00040 | 0.00014 | 0.00034 | 0.00628 | 0.00004 | 0.00055 | 0.00035 | 0.00045 | 0.00957 |
| 114       | 0.00014  | 0.00008 | 0.00028 | 0.00026 | 0.00587 | 0.00106 | 0.00000 | 0.00000 | 0.00008 | 0.00615 |
| 115       | 0.00024  | 0.00007 | 0.00023 | 0.00086 | 0.01187 | 0.00000 | 0.00001 | 0.00009 | 0.00016 | 0.00600 |
| 116       | 0.00000  | 0.00000 | 0.00000 | 0.00001 | 0.00034 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00188 |
| 117       | 0.00014  | 0.00016 | 0.00085 | 0.00147 | 0.00982 | 0.00045 | 0.00001 | 0.00013 | 0.00164 | 0.01192 |
| 118       | 0.00000  | 0.00000 | 0.00001 | 0.00003 | 0.00094 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00437 |
| 119       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00209 |
| 120       | 0.03990  | 0.02363 | 0.02598 | 0.02301 | 0.09222 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00177 |
| 121       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01882 | 0.01272 | 0.01256 | 0.00978 | 0.03950 |
| 122       | 0.04438  | 0.00504 | 0.00761 | 0.00542 | 0.02010 | 0.02489 | 0.00384 | 0.00847 | 0.00608 | 0.01869 |
| 123       | 0.00058  | 0.04411 | 0.00412 | 0.00746 | 0.01939 | 0.00110 | 0.02360 | 0.00319 | 0.00708 | 0.02010 |
| 124       | 0.00165  | 0.00032 | 0.03957 | 0.00342 | 0.02013 | 0.00079 | 0.00106 | 0.02047 | 0.00264 | 0.02146 |
| 125       | 0.00044  | 0.00030 | 0.00012 | 0.03374 | 0.01692 | 0.00041 | 0.00083 | 0.00117 | 0.01831 | 0.01893 |
| 126       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.01922 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00587 |

| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 61       | 62      | 63      | 64      | 65      | 66      | 67      | 68      | 69      | 70      |
| 1         | 0.14187  | 0.15090 | 0.07743 | 0.00000 | 0.00000 | 0.01157 | 0.00039 | 0.00000 | 0.00000 | 0.00000 |
| 2         | 0.00723  | 0.00054 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 3         | 0.14672  | 0.15364 | 0.07692 | 0.00000 | 0.00000 | 0.01260 | 0.00039 | 0.00000 | 0.00000 | 0.00000 |
| 4         | 0.00384  | 0.00023 | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 5         | 0.00708  | 0.00315 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 6         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 7         | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 8         | 0.14352  | 0.15866 | 0.08551 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 9         | 0.00723  | 0.00035 | 0.00005 | 0.00000 | 0.00000 | 0.01260 | 0.00039 | 0.00000 | 0.00000 | 0.00000 |
| 10        | 0.00030  | 0.00005 | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11        | 0.00707  | 0.00092 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12        | 0.00174  | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00064 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13        | 0.00723  | 0.00054 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14        | 0.11702  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01290 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15        | 0.00707  | 0.05051 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00039 | 0.00000 | 0.00000 | 0.00000 |
| 16        | 0.00723  | 0.00048 | 0.00557 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17        | 0.00310  | 0.01024 | 0.01649 | 0.01263 | 0.02577 | 0.01212 | 0.04386 | 0.06427 | 0.04762 | 0.10645 |
| 18        | 0.00016  | 0.00072 | 0.00038 | 0.00047 | 0.00121 | 0.00000 | 0.00000 | 0.00029 | 0.00014 | 0.00167 |
| 19        | 0.00333  | 0.01076 | 0.01655 | 0.01127 | 0.02163 | 0.01761 | 0.04631 | 0.06483 | 0.04814 | 0.11031 |
| 20        | 0.00003  | 0.00007 | 0.00000 | 0.00013 | 0.00099 | 0.00001 | 0.00018 | 0.00040 | 0.00126 | 0.00026 |
| 21        | 0.00025  | 0.00086 | 0.00036 | 0.00047 | 0.00253 | 0.00000 | 0.00000 | 0.00047 | 0.00051 | 0.00319 |
| 22        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 23        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 24        | 0.02071  | 0.05030 | 0.06919 | 0.05157 | 0.09820 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 25        | 0.00016  | 0.00042 | 0.00038 | 0.00047 | 0.00119 | 0.02547 | 0.04637 | 0.06514 | 0.04852 | 0.11243 |
| 26        | 0.00010  | 0.00029 | 0.00030 | 0.00007 | 0.00001 | 0.00002 | 0.00010 | 0.00009 | 0.00001 | 0.00003 |
| 27        | 0.00379  | 0.00997 | 0.01616 | 0.01374 | 0.03818 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 28        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00035 | 0.00048 | 0.00077 | 0.00072 | 0.00239 |
| 29        | 0.00016  | 0.00071 | 0.00038 | 0.00047 | 0.00121 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 30        | 0.00016  | 0.00072 | 0.00038 | 0.00047 | 0.00121 | 0.00000 | 0.00000 | 0.00029 | 0.00007 | 0.00039 |
| 31        | 0.00016  | 0.00072 | 0.00038 | 0.00047 | 0.00121 | 0.00000 | 0.00000 | 0.00029 | 0.00007 | 0.00041 |
| 32        | 0.01974  | 0.00013 | 0.00034 | 0.00021 | 0.00098 | 0.06718 | 0.00006 | 0.00007 | 0.00013 | 0.00120 |
| 33        | 0.00013  | 0.02261 | 0.00023 | 0.00013 | 0.00055 | 0.00000 | 0.07752 | 0.00016 | 0.00022 | 0.00070 |
| 34        | 0.00016  | 0.00070 | 0.02608 | 0.00011 | 0.00053 | 0.00000 | 0.00000 | 0.06781 | 0.00015 | 0.00091 |
| 35        | 0.00016  | 0.00072 | 0.00037 | 0.02586 | 0.00044 | 0.00000 | 0.00000 | 0.00029 | 0.06173 | 0.00039 |
| 36        | 0.00016  | 0.00072 | 0.00038 | 0.00038 | 0.00515 | 0.00000 | 0.00000 | 0.00029 | 0.00014 | 0.00310 |
| 37        | 0.33422  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01102 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38        | 0.02242  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39        | 0.34821  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01135 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40        | 0.01090  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00010 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41        | 0.02351  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 42        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 43        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 44        | 0.35535  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 45        | 0.02241  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.01135 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 46        | 0.00122  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00015 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 47        | 0.02672  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 48        | 0.00529  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00053 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 49        | 0.02241  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 50        | 0.02242  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 51        | 0.02242  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 52        | 0.02264  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 53        | 0.05027  | 0.24197 | 0.00000 | 0.00000 | 0.00000 | 0.01461 | 0.07808 | 0.00000 | 0.00000 | 0.00000 |
| 54        | 0.00040  | 0.00387 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00002 | 0.00000 | 0.00000 | 0.00000 |
| 55        | 0.05077  | 0.24200 | 0.00000 | 0.00000 | 0.00000 | 0.01461 | 0.07932 | 0.00000 | 0.00000 | 0.00000 |
| 56        | 0.00050  | 0.00067 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00080 | 0.00000 | 0.00000 | 0.00000 |
| 57        | 0.00001  | 0.00866 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 58        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 59        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 60        | 0.05597  | 0.30519 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 61        | 0.00040  | 0.00228 | 0.00000 | 0.00000 | 0.00000 | 0.01461 | 0.07961 | 0.00000 | 0.00000 | 0.00000 |
| 62        | 0.00030  | 0.00136 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00043 | 0.00000 | 0.00000 | 0.00000 |
| 63        | 0.00360  | 0.03202 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |



| Campaigns | Products |         |         |         |         |         |         |         |         |         |
|-----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|           | 61       | 62      | 63      | 64      | 65      | 66      | 67      | 68      | 69      | 70      |
| 64        | 0.00000  | 0.00004 | 0.00000 | 0.00000 | 0.00000 | 0.00055 | 0.00156 | 0.00000 | 0.00000 | 0.00000 |
| 65        | 0.00040  | 0.00386 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 66        | 0.00040  | 0.00387 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 67        | 0.00040  | 0.00387 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 68        | 0.13167  | 0.00059 | 0.00000 | 0.00000 | 0.00000 | 0.03494 | 0.00025 | 0.00000 | 0.00000 | 0.00000 |
| 69        | 0.00000  | 0.01431 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00032 | 0.00000 | 0.00000 | 0.00000 |
| 70        | 0.00693  | 0.02253 | 0.13879 | 0.00000 | 0.00000 | 0.03424 | 0.02809 | 0.13956 | 0.00000 | 0.00000 |
| 71        | 0.00000  | 0.00013 | 0.00137 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00103 | 0.00000 | 0.00000 |
| 72        | 0.00693  | 0.02688 | 0.13654 | 0.00000 | 0.00000 | 0.03821 | 0.02809 | 0.14152 | 0.00000 | 0.00000 |
| 73        | 0.00000  | 0.00000 | 0.00003 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00141 | 0.00000 | 0.00000 |
| 74        | 0.00000  | 0.00000 | 0.00134 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00165 | 0.00000 | 0.00000 |
| 75        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 76        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 77        | 0.02654  | 0.04615 | 0.24925 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 |
| 78        | 0.00000  | 0.00013 | 0.00137 | 0.00000 | 0.00000 | 0.03821 | 0.02809 | 0.14261 | 0.00000 | 0.00000 |
| 79        | 0.00000  | 0.00000 | 0.00108 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00031 | 0.00000 | 0.00000 |
| 80        | 0.00329  | 0.00502 | 0.05136 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 81        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00096 | 0.00045 | 0.00271 | 0.00000 | 0.00000 |
| 82        | 0.00000  | 0.00013 | 0.00137 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 83        | 0.00000  | 0.00013 | 0.00137 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00103 | 0.00000 | 0.00000 |
| 84        | 0.00000  | 0.00013 | 0.00137 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00103 | 0.00000 | 0.00000 |
| 85        | 0.06147  | 0.00000 | 0.00112 | 0.00000 | 0.00000 | 0.07245 | 0.00000 | 0.00024 | 0.00000 | 0.00000 |
| 86        | 0.00000  | 0.07752 | 0.00082 | 0.00000 | 0.00000 | 0.00000 | 0.06961 | 0.00055 | 0.00000 | 0.00000 |
| 87        | 0.00000  | 0.00000 | 0.01702 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00218 | 0.00000 | 0.00000 |
| 88        | 0.00139  | 0.00365 | 0.00899 | 0.05771 | 0.00000 | 0.00790 | 0.03406 | 0.03706 | 0.19640 | 0.00000 |
| 89        | 0.00000  | 0.00000 | 0.00007 | 0.00300 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00108 | 0.00000 |
| 90        | 0.00177  | 0.00445 | 0.00925 | 0.04750 | 0.00000 | 0.02072 | 0.03406 | 0.03706 | 0.20086 | 0.00000 |
| 91        | 0.00000  | 0.00000 | 0.00000 | 0.00071 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00972 | 0.00000 |
| 92        | 0.00000  | 0.00000 | 0.00000 | 0.00365 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00395 | 0.00000 |
| 93        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 94        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 95        | 0.02952  | 0.03541 | 0.04187 | 0.18603 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 96        | 0.00000  | 0.00000 | 0.00007 | 0.00300 | 0.00000 | 0.02574 | 0.03406 | 0.03706 | 0.20384 | 0.00000 |
| 97        | 0.00000  | 0.00000 | 0.00000 | 0.00052 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00008 | 0.00000 |
| 98        | 0.00362  | 0.00288 | 0.00717 | 0.07307 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 99        | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00029 | 0.00000 | 0.00000 | 0.00558 | 0.00000 |
| 100       | 0.00000  | 0.00000 | 0.00007 | 0.00300 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 101       | 0.00000  | 0.00000 | 0.00007 | 0.00300 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00054 | 0.00000 |
| 102       | 0.00000  | 0.00000 | 0.00007 | 0.00300 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00054 | 0.00000 |
| 103       | 0.01889  | 0.00000 | 0.00003 | 0.00141 | 0.00000 | 0.07022 | 0.00000 | 0.00000 | 0.00101 | 0.00000 |
| 104       | 0.00000  | 0.02756 | 0.00000 | 0.00090 | 0.00000 | 0.00000 | 0.08838 | 0.00000 | 0.00170 | 0.00000 |
| 105       | 0.00000  | 0.00000 | 0.04120 | 0.00084 | 0.00000 | 0.00000 | 0.00000 | 0.10061 | 0.00116 | 0.00000 |
| 106       | 0.00000  | 0.00000 | 0.00000 | 0.01117 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00968 | 0.00000 |
| 107       | 0.00051  | 0.00156 | 0.00384 | 0.00884 | 0.04350 | 0.00136 | 0.02403 | 0.03346 | 0.03789 | 0.17967 |
| 108       | 0.00000  | 0.00000 | 0.00000 | 0.00014 | 0.00204 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00282 |
| 109       | 0.00075  | 0.00174 | 0.00468 | 0.00875 | 0.03651 | 0.00649 | 0.02769 | 0.03346 | 0.03780 | 0.18619 |
| 110       | 0.00000  | 0.00000 | 0.00000 | 0.00006 | 0.00168 | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00044 |
| 111       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00427 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00538 |
| 112       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 113       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 114       | 0.00950  | 0.03166 | 0.03729 | 0.04681 | 0.16575 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 115       | 0.00000  | 0.00000 | 0.00000 | 0.00014 | 0.00201 | 0.01866 | 0.02769 | 0.03346 | 0.03780 | 0.18976 |
| 116       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00006 |
| 117       | 0.00164  | 0.00228 | 0.00107 | 0.00738 | 0.06444 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 118       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00013 | 0.00000 | 0.00000 | 0.00001 | 0.00403 |
| 119       | 0.00000  | 0.00000 | 0.00000 | 0.00014 | 0.00204 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 120       | 0.00000  | 0.00000 | 0.00000 | 0.00014 | 0.00204 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00065 |
| 121       | 0.00000  | 0.00000 | 0.00000 | 0.00014 | 0.00204 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00069 |
| 122       | 0.01325  | 0.00000 | 0.00003 | 0.00005 | 0.00165 | 0.05771 | 0.00000 | 0.00000 | 0.00000 | 0.00203 |
| 123       | 0.00000  | 0.01869 | 0.00000 | 0.00003 | 0.00092 | 0.00000 | 0.07841 | 0.00000 | 0.00000 | 0.00118 |
| 124       | 0.00000  | 0.00000 | 0.03014 | 0.00000 | 0.00090 | 0.00000 | 0.00000 | 0.09165 | 0.00000 | 0.00154 |
| 125       | 0.00000  | 0.00000 | 0.00000 | 0.04122 | 0.00075 | 0.00000 | 0.00000 | 0.00000 | 0.10210 | 0.00066 |
| 126       | 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00869 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00523 |

# Appendix J

## Example of Cycle Stock Inventory Estimate

After solving the PoT model, we calculate the inventory of product 31 from Example 1. We consider the model is run and the optimal solution is found. The actual average inventory ( $\tilde{I}_{31}$ ) of this product is calculated based on equation 4.15 and is compared to the estimated cycle stock inventory ( $I_{31}$ ). Calculations are presented in the following table:

| $c$   | $k$ | $t_{ck}$ (hour) | $N_k$ | $R_c$ (year) | $\alpha_{c31}$ | $d_{c31}$  | $R_c\alpha_{c31}$ | $R_c\alpha_{c31} - d_{c31}$ | $I_{31}$  | $\tilde{I}_{31}$ |
|-------|-----|-----------------|-------|--------------|----------------|------------|-------------------|-----------------------------|-----------|------------------|
| 6     | 3   | 87.21           | 6.5   | 13793080.76  | 0.00034        | 1,460.66   | 4,689.65          | 3,228.99                    | 112.36    | 77.36            |
| 10    | 3   | 53.081          | 6.5   | 13402767.38  | 0.15242        | 387,273.25 | 2,042,849.80      | 1,655,576.55                | 29,790.25 | 24,142.76        |
| 46    | 4   | 34.68           | 3.25  | 13998591.49  | 0.33728        | 292,392.34 | 4,721,444.94      | 4,429,052.60                | 44,983.44 | 42,197.68        |
| 79    | 3   | 32.04           | 6.5   | 14582336.18  | 0.04188        | 69,882.47  | 610,708.24        | 540,825.77                  | 5,375.57  | 4,760.46         |
| 97    | 4   | 43.14           | 3.25  | 15286744.19  | 0.02573        | 30,300.30  | 393,327.93        | 363,027.63                  | 4,661.58  | 4,302.48         |
| 116   | 3   | 64.44           | 6.5   | 15048129.08  | 0.01479        | 51,221.02  | 222,561.83        | 171,340.81                  | 3,940.08  | 3,033.30         |
| Total |     |                 |       |              |                |            |                   |                             | 88,863.28 | 78,514.03        |

The difference between actual and estimated cycle stock inventory is 10,349.25 which is 13% of the actual cycle stock inventory.

We used this estimation to avoid dealing with non-linear programming.

# Appendix K

## Cutting Pattern Code

First, the code creates all possible combination of the main cut through following functions; “nwMaxCalc”, calculates the maximum value for the number of each width according to the chosen thickness in a list called “nWmx”. Then “combos” generates all combinations of the widths values which are equal or less than the the maximum number calculated in “nWmx” and creates a list called “r”. Function “wcalc” calculates the total width given the list according to the given widths and kerf and returns the total width by “TotW”. Thus, for each thickness, as discussed in chapter 3, we are able to find all combinations of widths which create all possible main cuts.

Then for each main cut, “FinalCalcUD”, first calculates the remaining area of the assumed circle (the circle whose diameter equals the diagonal of the created main cut) and in the same procedure for the main cut, finds all combinations for above below cuts and save them in “seqUDVer” (for vertical cuts). If the combination could be fit into the remaining area of the circle, “List” (including area and combinations of widths) will be pushed into a heap called “CattingPattern”. A Same method used for above-bellow horizontal cuts. Right-left vertical and right-left horizontal cuts are generated by “FinalCalcRL”. All generated cuts are then saved in “CattingPattern”. The best “n” patterns with the highest area yield are chosen and kept in the list called “Best”. All created patterns by their specification including radius of the pattern, thickness of the main cut, widths of the

sub-cuts in the main cut, and width and thickness of above-bellow and rightleft cuts are stored and sorted based on the cutting pattern radius in a list called “xx”.

Logs are simulated according to their distribution functions through “Logs” function. The user can easily change the settings for new log classes. The created logs are stored in a list called “log” by three specifications: small radius, large radius and length. However, the logs are then saved in a list called “SLogs”.

For each log, the patterns which their radiuses are between small and large radius of the log are chosen. The length of each sub cut resulting from implementing the patterns on the log are calculated by functions “CutPattern1” (for main cut and vertical above-bellow and horizontal right-left cuts) and “CutPattern2” (for horizontal above-bellow and vertical right-left cuts). Then for each price list the value of each pattern on each log is calculated and the most valuable pattern is saved in the “BestPatternS”. The resulting pieces are then stored in a list called “LogSolutions”. The user should then dump information of the resulting pieces (list “LogSolutions”) and logs (list “SLogs”) into a pickle file through running “pickle.dump(obj, file[, protocol])” instruction. The saved pickle file is then used by two other pieces of codes; “Run” and “RunSort”.

The second code called “PieceMaps” is another file that creates the price of each piece, based on each price list and should be available prior to the main code. This file is where the user defines the price lists. The main code then imports and uses the price information of each piece based on the “PiceMaps” results.

The third code called “Run”, loads the pickled file and calculates actual, target and nominal percent yields and fractional outputs of each piece over all the logs and for all the price lists and writes them in an excel file.

As we assumed the first 2 log classes are generated once, and other 5 classes are just different sorts of these two classes, the forth code called “RunSort”, first loads the pickled files of the first 2 log classes, combines and sorts the logs to create more classes and same as “Run” code, then it calculates and writes the outputs of the new campaigns in an excel file.

```

import math
from math import sqrt
import heapq
import itertools
import random
import json # JSON (JavaScript Object Notation) used for writing a list in to a file
from PieceMaps import *

def combos(n): # Generates all combinations of a given sequence
    r=[]
    for x in n:
        t = []
        for y in range(x+1):
            for i in r:
                t.append(i+[y])
        r = t
    #print "r:", r
    return r

def wcalc(nums,W,aW,kerf): #calculating total width with given sequence and allowances and kerfs
    TotW=0
    for i in range(len(W)):
        TotW += nums[i]*(W[i]+aW[i] + kerf)
    TotW -=kerf
    return TotW

def nwMaxCalc(T,W,aW,kerf,kmax,maxW2,maxT2): #Generates maximum number of each width
value for a given main cut
        #if maximum W= kmax*T
    nWmx=[]
    for i in range(len(T)):
        if T[i]<=maxT2:
            nWmx.append([])

            for j in range(len(W)):
                maxw=(T[i]*kmax[i]+kerf)
                if maxw>maxW2:
                    maxw=maxW2
                nws= maxw/(W[j]+aW[j] +kerf)
                nWmx[i].append(int(nws))
            #print "i:", i
            if (i==0) or (i==4) or (i==5) or (i==3):
                nWmx[i][2]=0
                nWmx[i][3]=0

            elif i==1:
                nWmx[i][3]=0

            elif i==2:
                nWmx[i][2]=0

```

```

# print nWmx
return nWmx

def FinalCalcUD(T,R,TUDVerHor,W,aW,kerf,AreaLog,TotWidHor): # generating patterns
maxTUDVer=R-kerf-T/2. # maximum available area for thickness for Vertical cuts
maxTUDHor=TotWidHor # maximum available area for half of thickness for Horizontal cuts
CuttingPattern=[]
for l in range(len(TUDVerHor)):
    if (TUDVerHor[l])<=maxTUDVer: #vertical cuts
        Pcheck=pow(R,2)-pow((TUDVerHor[l]+kerf+T/2.),2)
        if Pcheck>=0:
            maxWUDVer=2*pow(Pcheck,0.5)
            kr=[999,999,999,999,999,999]
            numsWUDVer=nwMaxCalc(TUDVerHor,W,aW,kerf,kr,maxWUDVer,maxTUDVer)
            seqUDVer=combos(numsWUDVer[l]) #generating all sequences for Above-Bellow(Up and
Down) and Right-Left cuts
            for t in range(len(seqUDVer)):
                TotWidUDVer=wcalc(seqUDVer[t],W,aW,kerf) # Final width for A-B and R-L cuts
                SmUDVer=sum(seqUDVer[t])
                if 0<TotWidUDVer <= maxWUDVer :
                    AreaCutUDVer=2*(TotWidUDVer-(SmUDVer-1)*kerf)*(TUDVerHor[l]) #Area
calculations considering kerf
                    List=AreaCutUDVer, seqUDVer[t], TUDVerHor[l], "Vertical UD"
                    heapq.heappush(CuttingPattern,list(List)) #push the "List":(area, sequence and T) into a
heap called CattingPattern
            if TUDVerHor[l]<=maxTUDHor: #horizontal cuts
                Pcheck=pow(R,2)-pow((TUDVerHor[l]/2.),2)
                if Pcheck>=pow(T/2.,2):
                    maxWUDHor=pow(Pcheck,0.5)-(T/2.)-kerf
                    kr=[999,999,999,999,999,999]
                    numsWUDHor=nwMaxCalc(TUDVerHor,W,aW,kerf,kr,maxWUDHor,maxTUDHor)
                    seqUDHor=combos(numsWUDHor[l]) #generating all sequences for Above-Bellow(Up and
Down) and Right-Left cuts
                    for t in range(len(seqUDHor)):
                        TotWidUDHor=wcalc(seqUDHor[t],W,aW,kerf) # Final width for A-B and R-L cuts
                        SmUDHor=sum(seqUDHor[t])
                        if 0<TotWidUDHor <= maxWUDHor :
                            AreaCutUDHor=2*(TotWidUDHor-(SmUDHor-1)*kerf)*(TUDVerHor[l]) #Area
calculations considering kerf
                            List=AreaCutUDHor, seqUDHor[t], TUDVerHor[l], "Horizontal UD"
                            heapq.heappush(CuttingPattern,list(List)) #push the "List":(area, sequence and T) into a
heap called CattingPattern

        Best= heapq.nlargest(n,CuttingPattern) # Finding "n" best (largest) areas with its contents
(sequence and T)-> best patterns
        CuttingPattern=[] # make it empty for next iteration
        return Best

def FinalCalcRL(T,R,TRLVerHor,W,aW,kerf,AreaLog,TotWidVerHor): # generating patterns
maxTRLVer=T
maxTRLHor=R-TotWidVerHor/2.-kerf # maximum available area for thickness
CuttingPattern=[]
for l in range(len(TRLVerHor)):
    if TRLVerHor[l]<=maxTRLVer: # vertical cuts
        Pcheck=pow(R,2)-pow(((TRLVerHor[l])/2.),2)

```

```

if Pcheck>= pow(TotWidVerHor/2.,2):
    maxWRLVer=pow(Pcheck,0.5)-(TotWidVerHor/2.)-kerf
    kr=[999,999,999,999,999,999]
    numsWRLVer=nwMaxCalc(TRLVerHor,W,aW,kerf,kr,maxWRLVer,maxTRLVer)
    seqRLVer= combos(numsWRLVer[1]) #generating all sequences for Above-Bellow(Up and
Down) and Right-Left cuts
    for t in range(len(seqRLVer)):
        TotWidRLVer=wcalc(seqRLVer[t],W,aW,kerf) # Final width for A-B and R-L cuts
        SmRLVer=sum(seqRLVer[t])
        if 0<TotWidRLVer <= maxWRLVer :
            AreaCutRLVer=2*(TotWidRLVer-(SmRLVer-1)*kerf)*(TRLVerHor[1]) #Area calculations
considering kerf
            List=AreaCutRLVer, seqRLVer[t], TRLVerHor[1], "Vertical RL"
            heapq.heappush(CuttingPattern,list(List)) #push the "List":(area, sequence and T) into a
heap called CuttingPattern

if TRLVerHor[1]<=maxTRLHor: # horizontal cuts
    Pcheck=pow(R,2)-pow((TRLVerHor[1]+kerf+TotWidVerHor/2.),2)
    if Pcheck>=0:
        maxWRLHor=2*pow(Pcheck,0.5)
        kr=[999,999,999,999,999,999]
        numsWRLHor=nwMaxCalc(TRLVerHor,W,aW,kerf,kr,maxWRLHor,maxTRLHor)
        seqRLHor= combos(numsWRLHor[1]) #generating all sequences for Above-Bellow(Up and
Down) and Right-Left cuts
        for t in range(len(seqRLHor)):
            TotWidRLHor=wcalc(seqRLHor[t],W,aW,kerf) # Final width for A-B and R-L cuts
            SmRLHor=sum(seqRLHor[t])
            if 0<TotWidRLHor <= maxWRLHor :
                AreaCutRLHor=2*(TotWidRLHor-(SmRLHor-1)*kerf)*(TRLVerHor[1]) #Area
calculations considering kerf
                List=AreaCutRLHor, seqRLHor[t], TRLVerHor[1], "Horizontal RL"
                heapq.heappush(CuttingPattern,list(List)) #push the "List":(area, sequence and T) into a
heap called CuttingPattern
            Best= heapq.nlargest(n,CuttingPattern) # Finding "n" best (largest) areas with its contents
(sequence and T)-> best patterns
            CuttingPattern=[] # make it empty for next iteration
            return Best

def getIndx(l,List):
    found=False
    if l in List:
        indx=List.index(l)
        found=True
    if not(found):
        print "Index error", l, List
    return indx

def calc_r_rWanes(case,WR,WL,T1,T2,T3,wi,awi, WUDRL,WidthA, M,UD,RL):
    if case==1: #UD vertical, RL horizontal
        wrSq=WR*WR
        t1div2Sq=T1*T1/4.
        wr4Sq=(WR-(wi+awi)/4.)*(WR-(wi+awi)/4.)
        wAT3Sq=(WidthA/2.+kerf+T3)*(WidthA/2.+kerf+T3)
        T1kT2Sq=(T1/2.+kerf+T2)*(T1/2.+kerf+T2)
        r=(sqrt((wrSq+t1div2Sq))) *M+ (sqrt((wrSq+T1kT2Sq))) *UD+(sqrt((wrSq+wAT3Sq))) *RL
    #main radius

```

```

rWane1=(sqrt(wr4Sq+t1div2Sq))*M+(sqrt(wr4Sq+T1kT2Sq))*UD+(sqrt(wAT3Sq+pow(WR/2.-
(wi+awi)/4.,2)))*RL #upper wane

rWane2=(sqrt(wrSq+pow(3*T1/8.,2)))*M+(sqrt(wrSq+pow(T1/2.+kerf+3*T2/4.,2)))*UD+(sqrt(po
w(WidthA/2.+kerf+3*T3/4.,2)+pow(WR,2)))*RL #side wane
if case==2:
    wlsq=WL*WL
    t1div2Sq=T1*T1/4.
    wl22=(WL-(wi+awi)/4.)*(WL-(wi+awi)/4.)
    t1t2Sq=(T1/2.+kerf+T2)*(T1/2.+kerf+T2)
    wwA3=(WidthA/2.+kerf+T3)*(WidthA/2.+kerf+T3)
    r=(sqrt((wlsq+t1div2Sq))*M+ (sqrt((wlsq+t1t2Sq)))*UD+(sqrt((wlsq+wwA3)))*RL #main
radius
    rWane1=(sqrt(wl22+t1div2Sq))*M+sqrt(wl22+t1t2Sq)*UD+(sqrt(wwA3+pow(WL/2.-
(wi+awi)/4.,2)))*RL

rWane2=(sqrt(wlsq+pow(3*T1/8.,2)))*M+(sqrt(wlsq+pow(T1/2.+kerf+3*T2/4.,2)))*UD+(sqrt(pow(
WidthA/2.+kerf+3*T3/4.,2)+pow(WL,2)))*RL
if case==3:
    WW=WUDRL+kerf+T1/2.
    wwSq=WW*WW
    t2div2Sq=T2*T2/4.
    t238Sq=9.*T2*T2/64.
    t338Sq=9.*T3*T3/64.
    t3div2Sq=T3*T3/4.
    ww22=(WW-(wi+awi)/4.)*(WW-(wi+awi)/4.)
    ww33=(WidthA/2.+kerf+WUDRL)*(WidthA/2.+kerf+WUDRL)
    r=(sqrt(wwSq+t2div2Sq))*UD + (sqrt(ww33+t3div2Sq))*RL #main radius
    rWane1=(sqrt(ww22+t2div2Sq))*UD+(sqrt(pow(WidthA/2.+kerf+WUDRL-
(wi+awi)/4.,2)+t3div2Sq))*RL # Wane of type 1
    rWane2=(sqrt(wwSq+t238Sq))*UD+(sqrt(ww33+t338Sq))*RL # Wane of type 2
return r,rWane1,rWane2

```

```

def CutPattern1(mc,Thick,sq,sqq,rs,rl,l,T1,T2,T3,W,M,UD,RL):# Finding touching points and related
lengths for each subcut of a cut within a pattern (FOR MAIN CUT and VERTICAL UD and HORIZONTAL
RL)

```

```

    volSaw=0
    seq=list(sq)
    seqq=list(sqq)
    rc=[]
    WidthA=wcalc(seq,W,WidthAllow,kerf)
    if WidthA<0:
        WidthA=0
    WidthB=wcalc(seqq,W,WidthAllow,kerf) # for main cut
    if WidthB<0:
        WidthB=0
    WR=WL=WidthB/2.
    WUDRL=0
    iThick=getIdx(Thick,ThickSaw)
    i=0
    while (WR>0) or (WL>0):
        while (seqq[i]>0) and (sum(seqq)>0) :

```



```

if WR>=WL: #for right cuts
    case=1
    iWidth=i
    (r,rWane1,rWane2)= calc_r_rWanes(case,WR,WL,T1,T2,T3,W[i],WidthAllow[i],
WUDRL,WidthA, M,UD,RL)
    seqq[i]=-1
    WR=(W[i]+kerf+WidthAllow[i])
    if r<rs: # if r is less than smallest diameter, all the cut can be done for the whole length of the
log
        rc.append([r,l,int(l/2.)*2,int(l/2.)*2]) # appends[radius, length, integer Length] of a piece
        lenlum=int(l/2.)*2
    else:
        xcnn=l-(r-rs)*l/(rl-rs) #finds the length of the first cut
        rMax=max(rWane1,rWane2)
        if rMax<rs:
            rc.append([r,xcnn,int(l/2.)*2,int(l/2.)*2]) #the cuts are for the whole log (with wane)
            lenlum=int(l/2.)*2
        else:
            xWane=l-(rMax-rs)*l/(rl-rs)
            xcn=int(xcnn/2.)*2
            TLength=min(xWane,l) #finds the length of the touching point with wane
            rc.append([r,xcnn,xcn,int(TLength/2.)*2])
            lenlum=int(TLength/2.)*2

    else: #for left cuts
        case=2
        (r,rWane1,rWane2)= calc_r_rWanes(case,WR,WL,T1,T2,T3,W[i],WidthAllow[i],
WUDRL,WidthA, M,UD,RL)
        seqq[i]=-1
        WL=(W[i]+kerf+WidthAllow[i])
        iWidth=i

        if r<rs:
            rc.append([r,l,int(l/2.)*2,int(l/2.)*2])
        else:
            xcnn=l-(r-rs)*l/(rl-rs)
            rMax=max(rWane1,rWane2)
            if rMax<rs:
                rc.append([r,xcnn,int(l/2.)*2,int(l/2.)*2])
                lenlum=int(l/2.)*2
            else:
                xWane=l-(rMax-rs)*l/(rl-rs)
                xcn=int(xcnn/2.)*2
                TLength=min(xWane,l)
                rc.append([r,xcnn,xcn,int(TLength/2.)*2])
                lenlum=int(TLength/2.)*2
        if lenlum in LengthSell:
            ilength=LengthSell.index(lenlum)
            piece=(WidthSell[iWidth],ThickSell[iThick],LengthSell[ilength])
            ptyp=PieceSizeDict[piece]
            if M==1:
                npcs=1
            else:
                npcs=2
            volSaw+=npcs*PieceTargVolume[ptyp]
            for iobj in range(len(PriceLists)):

```

```

        #print "Cut1", piece,ptyp,npcs,iobj,objVal[iobj],PriceLists[iobj][ptyp]
        objVal[iobj]+=npcs*PriceLists[iobj][ptyp]
        #print piece,ptyp
        #print objVal[iobj]
        if ptyp in mc.keys(): # mc is a dictionary for counting number of each product with index
ptype. At the beginning it is empty if there is
            #not any product in mc, it goes through "else" and append that. otherwise it
calculates cumulative numbers for each product
            mc[ptyp]+=npcs
        else:
            mc[ptyp]=npcs

    if (i==3):
        break
    i+=1
return volSaw,rc,mc

def CutPattern2(mc,Thick,sq,sqq,rs,rl,l,T1,T2,T3,W,UD,RL):# Finding touching points and related
lengths for each subcut of a cut within a pattern (FOR HORIZONTAL UD and VERTICAL RL)
    seq=list(sq)
    seqq=list(sqq)
    rc=[]
    volSaw=0.
    WidthA=wcalc(seq,W,WidthAllow,kerf) # Total Width of the main cut
    if WidthA<0:
        WidthA=0
    WUDRL=wcalc(seqq,W,WidthAllow,kerf) # for main cut
    if WUDRL<0:
        WUDRL=0
    i=0
    WR=0
    WL=0
    M=0
    if Thick>0:
        ithick=getIndx(Thick,ThickSaw)

    while (WUDRL>0):
        while (seqq[i]>0) and (sum(seqq)>0) :
            case=3 #*****
            iwidth=i
            lenum=0
            (r,rWane1,rWane2)= calc_r_rWanes(case,WR,WL,T1,T2,T3,W[i],WidthAllow[i],
WUDRL,WidthA, M,UD,RL)
            seqq[i]-=1
            WUDRL-=(W[i]+kerf+WidthAllow[i])
            if r<rs: # if r is less than smallest diameter, all the cut can be done for the whole length of the
log
                rc.append([r,l,int(l/2.)*2,int(l/2.)*2]) # appends[radius, length, integer Length] of a piece
                lenlum=int(l/2.)*2
            else:
                xcnn=1-(r-rs)/(rl-rs) #finds the length of the first cut
                rMax=max(rWane1,rWane2)
                if rMax<rs:
                    intLen=int(l/2.)*2
                    rc.append([r,xcnn,int(l/2.)*2,int(l/2.)*2]) #the cuts are for the whole log (with wane)
                    lenlum=int(l/2.)*2
                else:

```

```

xWane=l-(rMax-rs)*l/(r1-rs)
xcn=int(xcnn/2.)*2
TLength=min(xWane,l) #finds the legth of the touching point with wane
rc.append([r,xcnn,xcn,int(TLength/2.)*2])
lenlum=int(TLength/2.)*2

if lenlum in LengthSell:
    ilength=LengthSell.index(lenlum)
    piece=(WidthSell[iwidth],ThickSell[ithickness],LengthSell[ilength])
    ptyp=PieceSizeDict[piece]
    if M==1:
        npcs=1
    else:
        npcs=2
    volSaw+=npcs*PieceTargVolume[ptyp]
    for iobj in range(len(PriceLists)):
#        print "Cut2", piece,ptyp,npcs,iobj,objVal[iobj],PriceLists[iobj][ptyp]
        objVal[iobj]+=npcs*PriceLists[iobj][ptyp]
#        print piece,ptyp
    if ptyp in mc.keys():
        mc[ptyp]+=npcs
    else:
        mc[ptyp]=npcs
    if (i==3):
        break
    i+=1
return volSaw,rc,mc

def Logs(S,L,PS1,PS2,PS3,PL1,PL2,PL3,PL4,PL5): #generating logs with two categories (small,large)
log=[]
#random.seed(123456789)

if (S==1):
    for i in range(nn):
        rnd=random.uniform(0, 2)
        RndNum=random.uniform(0,1)
        #print "RndNum:", RndNum
        if RndNum<=PS1:
            l=(8+rnd)
        elif PS1<RndNum<=PS1+PS2:
            l=(10+rnd)
        else:
            l=(12+rnd)
        rs=(random.uniform(2,3))
        rl=rs+random.uniform(5,20)/100.*1
        log.append([rs,rl,l]) # Log=[small radius, large radius, length]
elif (L==1):
    for i in range(nn):
        rnd=random.uniform(0, 2)
        RndNum=random.uniform(0,1)
        if RndNum<=PL1:
            l=(8+rnd)
        elif PL1<RndNum<=PL1+PL2:
            l=(10+rnd)
        elif PL1+PL2<RndNum<=PL1+PL2+PL3:
            l=(12+rnd)

```

```

elif PL1+PL2+PL3<RndNum<=PL1+PL2+PL3+PL4:
    l=(14+rnd)
else:
    l=(16+rnd)

rs=random.lognormvariate(1.19859,0.32343)
rl=rs+random.uniform(5,20)/100.*l
log.append([rs,rl,l]) # Log=[small radius, large radius, length]

return log

oo=input('What is the size of your class of logs? Small:1, Large:2-> ') # gets information about the
size of log classes from the user
nn=input("Please enter the number of logs in this class: -> ") # Asks the user about number of logs
within a class
PS1=0.4 # probability of having logs with length of 8-10 inside the class
PS2=0.4 # probability of having logs with length of 10-12 inside the class
PS3=0.2 # probability of having logs with length of 12-14 inside the class

PL1=0.0118 # probability of having logs with length of 8-10 inside the class
PL2=0.1242 # probability of having logs with length of 10-12 inside the class
PL3=0.2365 # probability of having logs with length of 12-14 inside the class
PL4=0.1307 # probability of having logs with length of 14-16 inside the class
PL5=0.4968 # probability of having logs with length of 16-18 inside the class

kerf=.15 #kerf
wRatioMx=[2,2,2,2,1.5,1.2] #maximum ratio of total width to the thickness of a lumber

mw=9999999
mt=9999999
n=20#number of best patterns regarding one main cut in terms of total area
numsW=nwMaxCalc(ThickSaw,WidthTarg,WidthAllow,kerf,wRatioMx,mw,mt)
print "starting to generate Patterns"
xx=[]
ii=0
for i in range(len(ThickSaw)): # generates the patterns and export them
    mxTotWid=wRatioMx[i]*ThickSaw[i]
    seq= combos(numsW[i]) # generating all the sequences

    for j in range(len(seq)):
        TotWid=wcalc(seq[j],WidthTarg,WidthAllow,kerf)
        Sm=sum(seq[j])
        if 0<TotWid <= mxTotWid :
            R=pow((pow(TotWid,2)+pow(ThickSaw[i],2)),0.5)/2. # Calculates the radius of the main cut
(pattern)
            AreaCut1=(TotWid-(Sm-1)*kerf)*ThickSaw[i] # calculates the area of the main cut
            AreaLog = math.pi*pow(R,2)
            FC=R,ThickSaw[i],seq[j]
            FirstCut=list(FC)
            UDCuts=FinalCalcUD(ThickSaw[i],R,ThickSaw,WidthTarg,WidthAllow,kerf,AreaLog,TotWid) #
U-D (above-bellow) cuts
            #print "UDCuts:", UDCuts
            RLCuts=FinalCalcRL(ThickSaw[i],R,ThickSaw,WidthTarg,WidthAllow,kerf,AreaLog,TotWid) #
R-L (Right-Left) cuts
            #print "RLCuts:", RLCuts

```

```

UDimag=[[0,[0,0,0,0],0,'None']]
RLimag=[[0,[0,0,0,0],0,'None']]

if (UDCuts!=[]) and (RLCuts!=[]):
    product=list(itertools.product(UDCuts,RLCuts)) # Products of two lists multiplication

elif (UDCuts==[]) and (RLCuts!=[]):
    product=list(itertools.product(UDimag,RLCuts))
elif (UDCuts!=[]) and (RLCuts==[]):

    product=list(itertools.product(UDCuts,RLimag))
else:
    product=list(itertools.product(UDimag,RLimag))
FinalList=list(FirstCut+[product]) # Producing a list of a certain main cut and all combinations
for UD and RL cuts

#print "FinalList:", FinalList
for k in range(len(product)):

xx.append([FinalList[0],FinalList[1],FinalList[2],FinalList[3][k][0][2],FinalList[3][k][0][1],FinalList[3][k][1][2],FinalList[3][k][1][1],FinalList[3][k][0][3],FinalList[3][k][1][3]]) # A list of the elements of
the produced patterns

report2="R:",FinalList[0],"T1:",FinalList[1],"Seq1:",FinalList[2],"T2:",FinalList[3][k][0][2],"Seq2:",FinalList[3][k][0][1],"T3:",FinalList[3][k][1][2],"Seq3:",FinalList[3][k][1][1]

    ii+=len(product)
print "Patterns all generated Number of Patterns =", len(xx)
#print "xx:", xx
xx.sort()
for i in range(len(xx)):
    xx[i].append(["P",i+1])

PatternFile=file('Patterns.txt','w') # generating a text file of patterns
json.dump(xx, PatternFile) # exporting patterns into that file

PatternFile.close() # Closing the file
print "-----"

print "Starting to generate Logs",oo

# GENERATING LOGS:
if oo<=1:
    S,L=1,0
else:
    S,L=0,1

VolumeLog=[]
SLogs=Logs(S,L,PS1,PS2,PS3,PL1,PL2,PL3,PL4,PL5)
print "Logs generated", " Number logs =",len(SLogs)

LogSolutions=[]
nlog=0 #print every 100 logs
nlogline=0

```

```

for a in range(len(SLogs)):

    objVal={}
    BestPatternS={}
    for iobj in range(len(PriceLists)):
        objVal[iobj]=0
        BestPatternS[iobj]=(-999,0,0,{})

    ## slope=(SLogs[a][1]-SLogs[a][0])/SLogs[a][2]
    ## avdiam=(SLogs[a][1]+SLogs[a][0])/2.
    ## minRad= avdiam-4.*slope
    minRad=SLogs[a][0]
    ## maxRad= avdiam+4.*slope
    maxRad=SLogs[a][1]
    True1=1
    True2=1
    b1=0
    tt=0
    pi=math.pi
    r1=SLogs[a][1]/12.
    r0=SLogs[a][0]/12.
    lenlog=SLogs[a][2]
    VolLog2=pi*lenlog*(pow(r0,2)+r0*r1+pow(r1,2))/3.
    while True1==1: # find the smallest eligible raduis of the sorted patterns as tt
        if xx[b1][0]<=minRad:
            b1+=1
            tt=b1
            if b1==len(xx):
                True1=0
            else:
                True1=0
        b2=len(xx)-1
        ttt=len(xx)-1
    while True2==1: # find the largest eligible raduis of the sorted patterns as ttt
        if xx[b2][0]>=maxRad:
            b2-=1
            ttt=b2
            if b2==-1:
                True2=0
            else:
                True2=0
    if tt>len(xx)-100:
        tt=len(xx)-100
    if tt>=0 and ttt>0:
        nlog+=1
        if nlog >=100:
            nlogline+=nlog
            print 'log',nlogline,SLogs[a][0], SLogs[a][1], tt, ttt, xx[tt][0], xx[ttt][0]
            # log #, rs, rl, p#(smallest), p#(largest), rp(smallest), rp(largest)0
            nlog=0
        #print "tt:", tt, "ttt:", ttt
        z=0
        for b in range(tt,ttt+1):
            SeqList=list(xx[b][2]) #keeping the original value of Seq1 when it becomes updated in
function RLCut

        #Main Cut

```

```

MC={}
ThickMain=xx[b][1]
ithick=getIdx(ThickMain,ThickSaw)

vMain,MainCut,MC=CutPattern1(MC,ThickMain,[0,0,0,0],xx[b][2],SLogs[a][0],SLogs[a][1],SLogs[a][2],xx[b][1],0,0,WidthTarg,1,0,0)
#print "xx:", xx[b]

#Up and Down Cuts
Seq2List=list(xx[b][4])
ThickUD=xx[b][3]
if xx[b][7]=="Vertical UD":

vUD,UDCut,MC=CutPattern1(MC,ThickUD,[0,0,0,0],Seq2List,SLogs[a][0],SLogs[a][1],SLogs[a][2],xx[b][1],xx[b][3],0,WidthTarg,0,1,0)
else:

vUD,UDCut,MC=CutPattern2(MC,ThickUD,[0,0,0,0],Seq2List,SLogs[a][0],SLogs[a][1],SLogs[a][2],xx[b][1],xx[b][3],0,WidthTarg,1,0)
# print UDCut

#Right and Left Cuts
Seq3List=list(xx[b][6])
ThickRL=xx[b][5]
if xx[b][8]=="Horizontal RL":

vRL,RLCut,MC=CutPattern1(MC,ThickRL,SeqList,Seq3List,SLogs[a][0],SLogs[a][1],SLogs[a][2],0,0,xx[b][5],WidthTarg,0,0,1)
else:

vRL,RLCut,MC=CutPattern2(MC,ThickRL,SeqList,Seq3List,SLogs[a][0],SLogs[a][1],SLogs[a][2],0,0,xx[b][5],WidthTarg,0,1)

volMCTot=vMain+vUD+vRL

PerPat2=volMCTot*100/VolLog2 #PERCENT YIELD

for iobj in range(len(PriceLists)):
    (curObj,curYield,curPat,curPieces)=BestPatterns[iobj]
    if objVal[iobj]>=curObj:
        BestPatterns[iobj]=(objVal[iobj],PerPat2,b, MC)
    objVal[iobj]=0.

LogSolutions.append(BestPatterns) #value, p#, piece dic
#print VolLog2

```

## PieceMaps

```
# generating dictionaries of lumbers and their corresponding values based on three formulas
```

```
ThickSell=[3., 4., 6., 8., 10., 12.]
```

```
ThickTarg=[2.5, 3.5, 5.5, 7.25, 9.25, 11.25]
```

```
ThickAllow=[.25,.25,.375,.625,.625,.625]
```

```
ThickSaw=[]
```

```
for i in range(len(ThickSell)):
```

```
    ThickSaw.append(ThickTarg[i]+ThickAllow[i])
```

```
WidthSell=[1., 2., 4., 6.]
```

```
WidthTarg=[0.75, 1.5, 3.5, 5.5]
```

```
WidthAllow=[.116, .160, .25, .375]
```

```
WidthSaw=[]
```

```
for i in range(len(WidthSell)):
```

```
    WidthSaw.append(WidthTarg[i]+WidthAllow[i])
```

```
LengthSell=[8., 10., 12., 14., 16.]
```

```
u=1
```

```
def getVol_byType(ptyp):
```

```
    vNom=PieceNomVolume[ptyp]
```

```
    vSaw=PieceSawVolume[ptyp]
```

```
    vTarg=PieceTargVolume[ptyp]
```

```
    return (vNom,vSaw,vTarg)
```

```
PieceSizes=[]
```

```
PieceSizeDict={}
```

```
PieceNomVolume=[]
```

```
PieceTargVolume=[]
```

```
PieceSawVolume=[]
```

```
PriceLists=[]
```

```
TempPriceList=[]
```

```
location=0
```

```
for i in range(len(WidthSell)):
```

```
    for j in range(len(ThickSell)):
```

```
        for k in range(len(LengthSell)):
```

```
            allowable=True
```

```
            if(WidthSell[i]>=4) and ThickSell[j]<>WidthSell[i]:
```

```
                allowable=False
```

```
            if allowable:
```

```
                piece=(int(WidthSell[i]), int(ThickSell[j]),int(LengthSell[k]))
```

```
                nomVol=WidthSell[i]*ThickSell[j]*LengthSell[k]/144.
```



```

        sawVol=WidthSaw[i]*ThickSaw[j]*LengthSell[k]/144.
        targVol=WidthTarg[i]*ThickTarg[j]*LengthSell[k]/144.
        PieceSizes.append(piece)
        PieceSizeDict[piece]=location
        PieceNomVolume.append(nomVol)
        PieceTargVolume.append(targVol)
        PieceSawVolume.append(sawVol)
        location+=1

TempPriceList=[]
for i in range(len(PieceSizes)):
    TempPriceList.append(PieceNomVolume[i])
PriceLists.append(TempPriceList)

for dim in range(3):
    TempPriceList=[]

    for i in range(len(PieceSizes)):
        (w,t,l)=PieceSizes[i]
        if dim ==0:
            vol=pow(w,1.5)*t*1/144.
        elif dim==1:
            vol=pow(t,1.5)*w*1/144.
        elif dim==2:
            vol=pow(l,1.5)*w*t/144.
        TempPriceList.append(vol)
    PriceLists.append(TempPriceList)

TempPriceList=[] #fit function
for i in range(len(PieceSizes)):
    (w,t,l)=PieceSizes[i]
    vol=(w*t*1/144.)*(1.0468*w+ 0.19851*t + 0.0487*1 +2.49827 - (0.0357*w*t)-
(0.00002781*w*t*1))
    TempPriceList.append(vol)
PriceLists.append(TempPriceList)

for j in range (len(WidthSell)): #emphasize on specific width
    TempPriceList=[]
    for i in range(len(PieceSizes)):
        (w,t,l)=PieceSizes[i]
        if w == WidthSell[j]:
            vol = 20*w*t*1/144.
        else:
            vol = w*t*1/144.
        TempPriceList.append(vol)
    PriceLists.append(TempPriceList)

```

```

for k in range (len(ThickSell)): #emphasize on specific Thickness
    TempPriceList=[]
    for i in range(len(PieceSizes)):
        (w,t,l)=PieceSizes[i]
        if t == ThickSell[k]:
            vol = 20*w*t*l/144.
        else:
            vol = w*t*l/144.
        TempPriceList.append(vol)

    PriceLists.append(TempPriceList)

for m in range (len(LengthSell)): #emphasize on specific length
    TempPriceList=[]
    for i in range(len(PieceSizes)):
        (w,t,l)=PieceSizes[i]
        if l == LengthSell[m]:
            vol = 20*w*t*l/144.
        else:
            vol = w*t*l/144.
        TempPriceList.append(vol)
    PriceLists.append(TempPriceList)

```

## Run

```
import pickle
from xlwt import Workbook, easyxf

import math
from PieceMaps import *

Z=open('tt.pkl','rb')
Cuts=pickle.load(Z)
Logs=pickle.load(Z)
#print Logs
PercentYield=[0 for i in range(20)] #List for average percent yield
Value=[0 for i in range(20)]
TotVolSaw=[0 for i in range(20)]
TotVolTarg=[0 for i in range(20)]
TotVolNom=[0 for i in range(20)]
TotProducts={}
TotVol={}
PerVol={}

Volume=0 #total log volume
pi=math.pi
for a in range(len(Logs)):
    r1=Logs[a][1]/12.
    r0=Logs[a][0]/12.
    lenlog=Logs[a][2]
    Volume+=pi*lenlog*(pow(r0,2)+r0*r1+pow(r1,2))/3.
#print "Log volume:", Volume

for i in range((20)): #each pricelist
    TotProducts[i]={}
    TotVol[i]={}
    PerVol[i]={}
    for j in range(0,len(Cuts)): #each log
        #PercentYield[i]+=(Cuts[j][i][1])
        Value[i]+=(Cuts[j][i][0])

        for k in Cuts[j][i][3].keys():
            #print k, Cuts[j][i][3][k]
            if k in TotProducts[i].keys():
                TotProducts[i][k]+=(Cuts[j][i][3][k])
            else:
                TotProducts[i][k]=Cuts[j][i][3][k]
```

```

#print "***"
for u in TotProducts[i].keys():
    TotVol[i][u]=(TotProducts[i][u]*PieceNomVolume[u])
    TotVolSaw[i]+=(TotProducts[i][u]*PieceSawVolume[u])
    TotVolTarg[i]+=(TotProducts[i][u]*PieceTargVolume[u])
    TotVolNom[i]+=(TotProducts[i][u]*PieceNomVolume[u])

    PerVol[i][u]=TotVol[i][u]/Volume
    TotVolSaw[i]=TotVolSaw[i]/Volume #Saw percent Yield
    TotVolTarg[i]=TotVolTarg[i]/Volume #Target Percent yield
    TotVolNom[i]=TotVolNom[i]/Volume

TotLog=len(Cuts)
##for i in range(len(PercentYield)):#average percent yield
##    PercentYield[i]=PercentYield[i]/TotLog
#print "PercentYield:", PercentYield

book=Workbook()
sheet1 = book.add_sheet('Sheet 1')
style1=easyxf('pattern: pattern solid, fore_colour light_green')
style2=easyxf('pattern: pattern solid, fore_colour light_orange')

sheet1.write(0,71,"PercentYieldSaw",style2)
sheet1.write(0,72,"PercentYieldTarg",style2)
sheet1.write(0,73,"PercentYieldNom",style2)
sheet1.write(0,74,"Value",style2)
sheet1.write(0,75,"LogVolume",style2)

for i in range(len(PieceSizes)):
    sheet1.write(0,i+1,i+1,style1)
    sheet1.write(1,i+1,str(PieceSizes[i]),style1)
for j in range(20):
    for k in range(2):
        sheet1.write(2*j+k+2,0,j+1,style1)
for i in range(len(TotProducts)):
    sheet1.write(2*i+2,71,TotVolSaw[i]*100)
    sheet1.write(2*i+2,72,TotVolTarg[i]*100)
    sheet1.write(2*i+2,73,TotVolNom[i]*100)
    sheet1.write(2*i+2,74,Value[i])
    sheet1.write(2*i+2,75,Volume)
    for j in (TotProducts[i]):
        sheet1.write(2*i+2,j+1,TotProducts[i][j])
        sheet1.write(2*i+3,j+1,PerVol[i][j]*100)

book.save('simple.xls')
print "Go to the excel file"

```

## RunSort

```
import math
from PieceMaps import *
from xlwt import Workbook, easyxf
pi=math.pi

Z=open('Small.pkl','rb')
Y=open('Large.pkl','rb')
CutsS=pickle.load(Z)
LogsS=pickle.load(Z)
CutsL=pickle.load(Y)
LogsL=pickle.load(Y)

nn=100000 #number of logs within a class
##PerYieldA=[0 for i in range(20)] #List for average percent yield
##PerYieldB=[0 for i in range(20)] #List for average percent yield
##PerYieldC=[0 for i in range(20)] #List for average percent yield
##PerYieldD=[0 for i in range(20)] #List for average percent yield
TotProductsA={}
TotProductsB={}
TotProductsC={}
TotProductsD={}
TotProductsE={}
TotVolA={}
TotVolB={}
TotVolC={}
TotVolD={}
TotVolE={}
PerVolA={}
PerVolB={}
PerVolC={}
PerVolD={}
PerVolE={}
TotVolSawA=[0 for i in range(20)]
TotVolTargA=[0 for i in range(20)]
TotVolNomA=[0 for i in range(20)]
TotVolSawB=[0 for i in range(20)]
TotVolTargB=[0 for i in range(20)]
TotVolNomB=[0 for i in range(20)]
TotVolSawC=[0 for i in range(20)]
TotVolTargC=[0 for i in range(20)]
TotVolNomC=[0 for i in range(20)]
TotVolSawD=[0 for i in range(20)]
```

```

TotVolTargD=[0 for i in range(20)]
TotVolNomD=[0 for i in range(20)]
TotVolSawE=[0 for i in range(20)]
TotVolTargE=[0 for i in range(20)]
TotVolNomE=[0 for i in range(20)]
CampA={}
CampB={}
CampC={}
CampD={}
CampE={}
Camp={}
ValA=[0 for i in range(20)]
ValB=[0 for i in range(20)]
ValC=[0 for i in range(20)]
ValD=[0 for i in range(20)]
ValE=[0 for i in range(20)]
VolA=0
VolB=0
VolC=0
VolD=0
VolE=0

Def
func(Camp,totProducts,totVol,perVol,TotVolSaw,TotVolTarg,TotVolNom,Value,Volum
e):
    #print Camp
    Value=[0 for i in range(20)]
    #PerYield=[0 for i in range(20)]
    Volume=0
    for m in (Camp): #each log Volume
        if m<nn:
            r1=LogsS[m][1]/12.
            r0=LogsS[m][0]/12.
            lenlog=LogsS[m][2]
            Volume+=pi*lenlog*(pow(r0,2)+r0*r1+pow(r1,2))/3.
            #print "Volume1:", Volume
        else:
            r1=LogsL[m-nn][1]/12.
            r0=LogsL[m-nn][0]/12.
            lenlog=LogsL[m-nn][2]
            Volume+=pi*lenlog*(pow(r0,2)+r0*r1+pow(r1,2))/3.
            #print "Volume2:", Volume
    print "Volume:", Volume
    for i in range((20)): #each pricelist
        totProducts[i]={}
        totVol[i]={}

```

```

perVol[i]={}
for j in (Camp): #each log

    #PerYield[i]+=(Camp[j][i][1])
    Value[i]+=Camp[j][i][0]

    for k in Camp[j][i][3].keys():
        #print k, CampA[j][i][3][k]
        if k in totProducts[i].keys():
            totProducts[i][k]+=Camp[j][i][3][k]

    else:
        totProducts[i][k]=Camp[j][i][3][k]
    for u in totProducts[i].keys():
        totVol[i][u]=(totProducts[i][u]*PieceNomVolume[u]) #volume of each product in
each campaign (pricelist
        TotVolSaw[i]+=(totProducts[i][u]*PieceSawVolume[u])
        TotVolTarg[i]+=(totProducts[i][u]*PieceTargVolume[u])
        TotVolNom[i]+=(totProducts[i][u]*PieceNomVolume[u])
        perVol[i][u]=totVol[i][u]/Volume
        #PerYield[i]=PerYield[i]/len(Camp)
        TotVolSaw[i]=(TotVolSaw[i]/Volume)*100 #Saw percent Yield
        TotVolTarg[i]=(TotVolTarg[i]/Volume)*100 #Target Percent yield
        TotVolNom[i]=(TotVolNom[i]/Volume)*100
        Len=len(Camp)
    return
totProducts,totVol,TotVolSaw,TotVolTarg,TotVolNom,perVol,Value,Volume,Len

```

```

for i in range(len(CutsS)):
    #Camp[i]={}
    if LogsS[i][2]<10:
        CampA[i]=CutsS[i]
    elif (LogsS[i][2]<12 and LogsS[i][2]>=10):
        CampB[i]=CutsS[i]
    elif (LogsS[i][2]<14 and LogsS[i][2]>=12):
        CampC[i]=CutsS[i]
    elif (LogsS[i][2]<16 and LogsS[i][2]>=14):
        CampD[i]=CutsS[i]
    else:
        CampE[i]=CutsS[i]

```

```

for i in range(len(CutsL)):
    if LogsL[i][2]<10:
        CampA[i+nn]=CutsL[i]
    elif (LogsL[i][2]<12 and LogsL[i][2]>=10):

```

```

    CampB[i+nn]=CutsL[i]
elif (LogsL[i][2]<14 and LogsL[i][2]>=12):
    CampC[i+nn]=CutsL[i]
elif (LogsL[i][2]<16 and LogsL[i][2]>=14):
    CampD[i+nn]=CutsL[i]
else:
    CampE[i+nn]=CutsL[i]

TotalProductA,TotalVolumeA,TotVolSawA,TotVolTargA,TotVolNomA,PerVolA,Value
A,VolumeA,LenA=func(CampA,TotProductsA,TotVolA,PerVolA,TotVolSawA,TotVol
TargA,TotVolNomA,ValA,VolA)
TotalProductB,TotalVolumeB,TotVolSawB,TotVolTargB,TotVolNomB,PerVolB,Value
B,VolumeB,LenB=func(CampB,TotProductsB,TotVolB,PerVolB,TotVolSawB,TotVolT
argB,TotVolNomB,ValB,VolB)
TotalProductC,TotalVolumeC,TotVolSawC,TotVolTargC,TotVolNomC,PerVolC,Value
C,VolumeC,LenC=func(CampC,TotProductsC,TotVolC,PerVolC,TotVolSawC,TotVolT
argC,TotVolNomC,ValC,VolC)
TotalProductD,TotalVolumeD,TotVolSawD,TotVolTargD,TotVolNomD,PerVolD,Value
D,VolumeD,LenD=func(CampD,TotProductsD,TotVolD,PerVolD,TotVolSawD,TotVol
TargD,TotVolNomD,ValD,VolD)
TotalProductE,TotalVolumeE,TotVolSawE,TotVolTargE,TotVolNomE,PerVolE,Value
E,VolumeE,LenE=func(CampE,TotProductsE,TotVolE,PerVolE,TotVolSawE,TotVolTarg
E,TotVolNomE,ValE,VolE)
book=Workbook()

style1=easyxf('pattern: pattern solid, fore_colour light_green')
style2=easyxf('pattern: pattern solid, fore_colour light_orange')
style3=easyxf('pattern: pattern solid, fore_colour yellow')

def
report(z,TotProducts,PerVol,TotVolSaw,TotVolTarg,TotVolNom,Value,Volume,Len):
    sheet1 = book.add_sheet("%r"%z,cell_overwrite_ok=True)
    sheet1.write(0,71,"PercentYieldSaw",style2)
    sheet1.write(0,72,"PercentYieldTarg",style2)
    sheet1.write(0,73,"PercentYieldNom",style2)
    sheet1.write(0,74,"Value",style2)
    sheet1.write(0,75,"LogVolume",style2)
    sheet1.write(0,76,"number of logs",style3)
    sheet1.write(1,76,Len,style3)
    for i in range(len(PieceSizes)):
        sheet1.write(0,i+1,i+1,style1)
        sheet1.write(1,i+1,str(PieceSizes[i]),style1)
    for j in range(21):
        for k in range(2):
            sheet1.write(2*j+k+2,0,j+1,style1)
    for i in range(len(TotProducts)):

```



```

sheet1.write(2*i+2,71,TotVolSaw[i])
sheet1.write(2*i+2,72,TotVolTarg[i])
sheet1.write(2*i+2,73,TotVolNom[i])
sheet1.write(2*i+2,74,Value[i])
sheet1.write(2*i+2,75,Volume)
for j in (TotProducts[i]):
    sheet1.write(2*i+2,j+1,TotProducts[i][j])
    sheet1.write(2*i+3,j+1,PerVol[i][j]*100)

book.save('simple.xls')
print "Go to the excel file"
A=report(1,
TotalProductA,PerVolA,TotVolSawA,TotVolTargA,TotVolNomA,ValueA,VolumeA,Len
nA)
B=report(2,
TotalProductB,PerVolB,TotVolSawB,TotVolTargB,TotVolNomB,ValueB,VolumeB,Len
B)
C=report(3,
TotalProductC,PerVolC,TotVolSawC,TotVolTargC,TotVolNomC,ValueC,VolumeC,Len
C)
D=report(4,
TotalProductD,PerVolD,TotVolSawD,TotVolTargD,TotVolNomD,ValueD,VolumeD,Le
nD)
E=report(5,
TotalProductE,PerVolE,TotVolSawE,TotVolTargE,TotVolNomE,ValueE,VolumeE,Len
E)

```

# Appendix L

## GLPK Code

```
/* initial Parameters */
param nProducts, integer; /* Maximum number of products*/
param nCampaigns, integer; /* Maximum number of campaigns*/
param T0; /*base production period*/
param Kmax, integer; /*maximum of K*/

/* sets */
set Products, default {1..nProducts}; /* set of products */
set Campaigns, default {1..nCampaigns}; /* set of campaigns */
set K, default {0..Kmax}; /* set of campaign coverages */
set L dimen 2;

/* parameters */
param Tk{k in K}, default T0*2^k;
param Nk{k in K}, default 1./Tk[k];
param Dp1 {p in Products}; /* annual demand for product p */
param Alpha {c in Campaigns, p in Products}; /* fraction output of product p per unit input of campaign c */
param Rc1 {c in Campaigns}; /* production input rate for campaign type c */
param Sc {c in Campaigns}; /* setup time for campaign c */
param Tmax {c in Campaigns}; /* maximum cycle length */
param Vp {p in Products}; /* value per unit for inventory of product p */
param Dp {p in Products}, default Dp1[p]/10000; /* annual demand for product p */
param TotDem, default sum{p in Products} Dp[p];
param Rc {c in Campaigns}, default Rc1[c]/10000; /* production input rate for campaign type c */

table data IN "CSV" "Demand1.csv":
    Products <- [Product], Dp1 ~ Demand, Vp ~ Price;
table data IN "CSV" "Campaign1.csv":
    Campaigns <- [Campaign], Rc1 ~ Rate, Sc ~ Setup;
table data IN "CSV" "Alpha1.csv":
    L <- [c,p], Alpha ~ alpha;

/* variables */
var tck {c in Campaigns, k in K} >=0; /* production time per cycle for campaign c with coverage k */
var yck {c in Campaigns, k in K} >=0, binary; /* campaign decision */
var DevPos{ p in Products} >=0; /* amount by which annual production of product p exceeds the demand for p */
var DevNeg{ p in Products} >=0; /* amount by which annual production of product p fails to meet the demand for p */
var IpAverage{p in Products}; /*Average inventory */
```

```

var Delta{p in Products}>=0;
var MaxDelta, >=0;

/*constraints */
s.t. OneCoverageForEachC {c in Campaigns} : sum{k in K} yck[c,k] <=1;
s.t. ProTimeIfCovK {c in Campaigns, k in K} :Nk[k]*tck[c,k]<=yck[c,k];
s.t. TimeAvailability : sum{c in Campaigns, k in K} (Nk[k]*tck[c,k]+Nk[k]*Sc[c]*yck[c,k]) <= 1;
s.t. DemandMeet {p in Products} : sum{c in Campaigns, k in K} Nk[k]*tck[c,k]*Rc[c]*Alpha[c,p] =
Dp[p]+DevPos[p]-DevNeg[p];
s.t. DemandCuts {p in Products, c in Campaigns, k in K} :
Nk[k]*tck[c,k]*Rc[c]*Alpha[c,p]<=Dp[p]*yck[c,k]+DevPos[p];
s.t. DeviationPos{p in Products} : DevPos[p]+DevNeg[p] <= Delta[p];
s.t. AvgInventory{p in Products} : IpAverage[p] = 0.5* sum{c in Campaigns, k in K}
(tck[c,k]*(Rc[c]*Alpha[c,p]));
s.t. Deviation: sum{ p in Products} Delta[p]*Dp[p] <=0.8* sum{ p in Products} Dp[p]**2;

/* Objective Function */
minimize z: sum{p in Products} 1* (Vp[p]*IpAverage[p]) + 50 * sum{p in Products}
(Delta[p]*(Dp[p]/TotDem)) ;

solve;
display Dp;
display Nk;
display Tk;
printf "Total Cost: %4.2f\n",
(sum{p in Products} (Vp[p]*IpAverage[p]) + 50 * sum{p in Products} (Delta[p]* (Dp[p]/TotDem)));
printf " \n";
printf "CAMPAIGNS\n";
printf " c N t t(weeks)\n";
for {c in Campaigns}
printf "%2i%6.2f%8.4f\n", c,(sum{k in K} (yck[c,k]*Nk[k])), " ",(sum{k in K} (tck[c,k])),
(sum{k in K} (tck[c,k]*52));
printf " \n";
printf "PRODUCTS\n";
printf " p Demand IpAverage DevPos DevNeg Delta\n";
for {p in Products}
printf "%2i%8.2f%8.2f%12.2f%9.2f%10.2f\n", p, Dp[p], IpAverage[p],
DevPos[p],DevNeg[p],MaxDelta;

data;
param nProducts :=70;
param nCampaigns :=126;
param T0 := 0.019230769230769;
param Kmax := 4;

end;

```

# Appendix M

## Simulation Codes

### Approach 1

```
import random
from xlrd import open_workbook
bookD=open_workbook('SimFile.xls')
sheetD=bookD.sheet_by_index(0)

nP=70 #number of productsp
nC=12 # number of campaigns

P=[i for i in range(nP)]
C=[j for j in range(nC)] # Here We do not bring the campaigns with N=0 and T=0

Demand=[]
V=[] # Inventory cost per time unit.
BackCost=[]
for i in range(nP):
    d=sheetD.cell(i+1,0).value
    v=sheetD.cell(i+1,1).value
    bb=sheetD.cell(i+1,1).value/.2
    Demand.append(d)
    V.append(v)
    BackCost.append(bb)

print "Enter each Product's demand arrival rate for C1...C6"
DemandArrRate=[23, 13, 41, 27, 18, 13, 20, 22, 46, 35, 15, 29, 13, 23, 23, 11, 21, 26, 28, 30, 39, 41,
14, 11, 22, 21, 46, 40, 9, 27, 37, 46, 17, 43, 9, 10, 19, 7, 47, 9, 36, 18, 16, 33, 21, 10, 30, 28, 17, 21,
13, 21, 36, 13, 6, 26, 27, 17, 30, 41, 35, 12, 7, 23, 19, 20, 31, 14, 28, 20]
print DemandArrRate
Dev=[(Demand[i]/float(DemandArrRate[i])).25 for i in range(nP)]
MaxDem=[]
MinDem=[]
for i in range(len(Demand)):
    MaxDem.append(Demand[i]/DemandArrRate[i]+Dev[i])
    MinDem.append(Demand[i]/DemandArrRate[i]-Dev[i])

tDay=7 #number of hours per day
nDay=5 #number of days per week
nWeek=52 #number of weeks per year
TotTime= nDay*tDay*nWeek #Total available time per year
SimTime=6*TotTime
```

```

LamArrTime=0.4
LamDemSize=0.4
#-----
# FORECASTED and Actual Demand based on hour time base
#-----
Forecast=[]for k in range(len(P))
Time=[0 for i in range(len(P))]
random.seed(11111111)
Due=[]for k in range(len(P))
TimeActual=[0 for i in range(len(P))]
for i in range(len(P)):
    AvgDTime=TotTime/float(DemandArrRate[i])
    AvgDSize=(MaxDem[i]+MinDem[i])/2.
    Time[i]+=AvgDTime
    #print "AvgDTime:", AvgDTime
    #print "AvgDSize:", AvgDSize
    while Time[i]<SimTime and TimeActual[i]<SimTime:
        #print "TIME BET:.", TimeBetArr
        ArrTimeD=float(random.expovariate(float(DemandArrRate[i]/float(TotTime))))
        DSize=random.uniform(MinDem[i], MaxDem[i])
        TimeActual[i]+=ArrTimeD
        TimeForecast=LamArrTime*ArrTimeD+(1-LamArrTime)*AvgDTime
        AvgDTime=TimeForecast
        DSizeForecast=LamDemSize*DSize+(1-LamDemSize)*AvgDSize
        AvgDSize=DSizeForecast
        Time[i]+=AvgDTime
    if Time[i]>SimTime and TimeActual[i]>SimTime:
        break
    Due[i].append([(int(TimeActual[i])),DSize,i,"A"])
    Forecast[i].append([(int(Time[i])),AvgDSize,i,"F"])

##### INVENTORY #####
from xlwt import Workbook
sheetC=bookD.sheet_by_index(1)
sheetA=bookD.sheet_by_index(2)

tunit=1 #hour
SimTimeP=4*TotTime #(run for 4 years)

N=[]
T=[]
R=[]
ST=[]
for i in range(nC):
    nn=sheetC.cell(i+1,1).value

```

```

N.append(nn)

tp=sheetC.cell(i+1,2).value
T.append(tp)

rp=sheetC.cell(i+1,3).value
R.append(rp)

st=sheetC.cell(i+1,4).value
ST.append(st)

K=[0,1,2,3,4]

Alpha=[] for i in range(nC)
for i in range(len(Alpha)):
    for j in range(nP):
        ss=sheetA.cell(i+1,j+1).value
        Alpha[i].append(ss)

Inv=[0 for i in range(len(P))]

m=input("How many weeks ahead of actual demands? ")
print " "

def ActualDem(t): # Demands for m periods
    Dema=[] for i in range(len(P))
    IndxDem=[0 for i in range(len(P))]
    for i in range(len(Demand)):
        for tt in range(len(Due[i])):
            if ((Due[i][tt][0])>t) and (Due[i][tt][0] <= (t+m*nDay*tDay)):
                Dema[i].append(Due[i][tt])
    if len(Dema[i])!=0:
        IndxDem[i]=Dema[i].index(Dema[i][len(Dema[i])-1])
        Dema[i].append(Forecast[i][IndxDem[i]])

    return Dema

def Func1(Inv,t): # calculate runout time
    ProdSelect=0
    NegInv={}
    Runout=[1000 for i in range(len(Demand))]
    ActDem=ActualDem(t)
    Sum=[0 for i in range(len(Demand))]
    Rlist=[]
    for i in range(len(ActDem)):
        for j in range(len(ActDem[i])-1):

```

```

    Sum[i]+=ActDem[i][j][1]
if Inv[i]<=0:
    RO=0
    NegInv[i]=Inv[i]-Sum[i]

elif Inv[i]>0 and Inv[i]<=Sum[i]:
    j=0
    ff=ActDem[i][j][1]
    while ff<=Inv[i]:
        ff+=ActDem[i][j+1][1]
        j+=1
    RO=ActDem[i][j][0]-t

elif Inv[i]>0 and Inv[i]>Sum[i]:
    hh=len(ActDem[i])-1
    if len(ActDem[i])==0:
        RO=1000
    elif ActDem[i][hh][0]==ActDem[i][hh-1][0]:
        RO=m*nDay*tDay+((Inv[i]-Sum[i])/(ActDem[i][hh][1]/0.1))
    else:
        RO=m*nDay*tDay+((Inv[i]-Sum[i])/(ActDem[i][hh][1]/(ActDem[i][hh][0]-
ActDem[i][hh-1][0])))

    Rlist.append(RO)
if NegInv=={}:
    MinP=Rlist[0]
    for j in range(len(Rlist)):
        if Rlist[j]<=MinP:
            MinP=Rlist[j]
            BestP=j
    ProSelect=BestP
else:
    #print NegInv
    ProSelect=min(NegInv.items(), key=lambda x: x[1])[0]
return ProSelect

def NextC(t): # finds the next campaign to be run based on the max RAlpha for p
    Prod=Func1(Inv,t)
    MaxC=R[0]*Alpha[0][Prod]
    for c in range(len(C)):
        intC=R[c]*Alpha[c][Prod]
        if intC>=MaxC:
            MaxC=intC
            BestC=c
    CampSelect=BestC
return CampSelect

```

```

Wbook = Workbook() # writing output
sheet1 = Wbook.add_sheet('Inventory')
sheet2 = Wbook.add_sheet('Cost')
sheet3= Wbook.add_sheet('LostSale')

r=0
t=0
Nums=[0 for i in range(len(C))]
List=[]
CampList=[]

#u=0
##v=1

for j in range(nP):
    sheet1.write(0,j+1,j+1)

while t<=SimTimeP:
    Camp=NextC(t)
    ProT=round((t+ST[Camp]+T[Camp]),0) # time at the end of producing campaign
    for j in range(len(Inv)):
        Inv[j]-=R[Camp]*Alpha[Camp][j]*ST[Camp]
    List.append(Camp)
    while t<ProT: # calculates Inventory over production time
        Dem=[0 for k in range(len(Demand))]
        for i in range(len(Due)):
            for j in range(len(Due[i])):
                if int(Due[i][j][0]) == (t+1):
                    Dem[i]+= Due[i][j][1]
                if Dem[i]<0:
                    print Dem

        for j in range(len(Inv)):
            Inv[j]+=R[Camp]*Alpha[Camp][j]-Dem[j]
            #print "InvStep:", [j], Inv[j], t
            if Inv[j]>=0:
                InvCosts=HoldingCost*Inv[j]*V[j]/(TotTime)
                sheet1.write(r+1,j+1,Inv[j])
                sheet2.write(r+1,j+1,InvCosts)
            else:
                InvCosts=10*HoldingCost*Inv[j]*V[j]/(TotTime)
                sheet3.write(r+1,j+1,InvCosts)
                sheet1.write(r+1,j+1,Inv[j])
        if t==TotTime:
            print "Number of campaigns in the first year:", Nums

```



```

if t==2*TofTime:
    print "Number of campaigns in the second year:", Nums
if t==3*TofTime:
    print "Number of campaigns in the third year:", Nums
sheet1.write(r+1,0,t+1)
t+=tunit
r+=1
CampList.append(ProT)

Nums[Camp]+=1
print "Number of campaigns running over the simulation time:", Nums
#print "List:", List
Wbook.save("SchedulingRunOut.xls")
Wbook = Workbook()
sheet1 = Wbook.add_sheet('SimDemand')
kkk=0
mmm=1
for j in range(len(Due)):

    for i in range(len(Due[j])):
        sheet1.write(i+1, kkk, Due[j][i][0])
        sheet1.write(i+1, mmm, Due[j][i][1])

    mmm+=2
    kkk+=2
Wbook.save("Demand-SchedulingRunOut.xls")
print "Go to the excel file"

```

## Approach 2

```
import random
from xlrld import open_workbook
bookD=open_workbook('SimFile.xls')
sheetD=bookD.sheet_by_index(0)

nP=70 #number of products
nC=12 # number of campaigns

P=[i+1 for i in range(nP)]
C=[j+1 for j in range(nC)] # Here We do not bring the campaigns with N=0 and T=0

Demand=[]
V=[] # Inventory cost per time unit.
for i in range(nP):
    d=sheetD.cell(i+1,0).value
    v=sheetD.cell(i+1,1).value
    Demand.append(d)
    V.append(v)

a=0
for i in range(len(Demand)):
    a+=Demand[i]

print "Enter each Product's demand arrival rate for C1...C6"
DemandArrRate=[23, 13, 41, 27, 18, 13, 20, 22, 46, 35, 15, 29, 13, 23, 23, 11, 21, 26, 28, 30, 39, 41,
14, 11, 22, 21, 46, 40, 9, 27, 37, 46, 17, 43, 9, 10, 19, 7, 47, 9, 36, 18, 16, 33, 21, 10, 30, 28, 17, 21,
13, 21, 36, 13, 6, 26, 27, 17, 30, 41, 35, 12, 7, 23, 19, 20, 31, 14, 28, 20]
print DemandArrRate
Dev=[(Demand[i]/float(DemandArrRate[i])*0.25) for i in range(nP)]
MaxDem=[]
MinDem=[]
for i in range(len(Demand)):
    MaxDem.append(Demand[i]/DemandArrRate[i]+Dev[i])
    MinDem.append(Demand[i]/DemandArrRate[i]-Dev[i])
tDay=7 #number of hours per day
nDay=5 #number of days per week
nWeek=52 #number of weeks per year
TotTime= nDay*tDay*nWeek #Total available time per year
SimTime=6*TotTime

LamArrTime=0.4
LamDemSize=0.4
```

```

#-----
# FORECASTED and Actual Demand based on hour time base
#-----
Forecast=[]for k in range(len(P))
Time=[0 for i in range(len(P))]
random.seed(123456789)
Due=[]for k in range(len(P))
TimeActual=[0 for i in range(len(P))]
for i in range(len(P)):
    AvgDTime=TotTime/float(DemandArrRate[i])
    AvgDSize=(MaxDem[i]+MinDem[i])/2.
    Time[i]+=AvgDTime
    while Time[i]<SimTime and TimeActual[i]<SimTime:
        #print "TIME BET: ", TimeBetArr
        ArrTimeD=float(random.expovariate(float(DemandArrRate[i]/float(TotTime))))
        DSize=random.uniform(MinDem[i], MaxDem[i])
        TimeActual[i]+=ArrTimeD
        TimeForecast=LamArrTime*ArrTimeD+(1-LamArrTime)*AvgDTime
        AvgDTime=TimeForecast
        DSizeForecast=LamDemSize*DSize+(1-LamDemSize)*AvgDSize
        AvgDSize=DSizeForecast
        Time[i]+=AvgDTime
    if Time[i]>SimTime and TimeActual[i]>SimTime:
        break
    Due[i].append([int(TimeActual[i]),DSize,i,"A"])
    Forecast[i].append([int(Time[i]),AvgDSize,i,"F"])

##### INVENTORY #####
from xlwt import Workbook
sheetC=bookD.sheet_by_index(1)
sheetA=bookD.sheet_by_index(2)

tunit=1 #1 hour
SimTimeP=4*TotTime #(run for 1 year)

N=[]
T=[]
R=[]
ST=[]
for i in range(nC):
    nn=sheetC.cell(i+1,1).value
    N.append(nn)

    tp=sheetC.cell(i+1,2).value
    T.append(tp)

```

```

rp=sheetC.cell(i+1,3).value
R.append(rp)

st=sheetC.cell(i+1,4).value
ST.append(st)

K=[0,1,2,3,4]

Alpha=[[[] for i in range(nC)]
for i in range(len(Alpha)):
    for j in range(nP):
        ss=sheetA.cell(i+1,j+1).value
        Alpha[i].append(ss)

Inv=[0 for i in range(len(P))]

def ActualDem(t,Sc,Tc): # summation of demands for Tc+Sc duration of each campaign run
    Dem=[[[] for i in range(len(P))]
Sum=[0 for i in range(len(Demand))]
    for i in range(len(Demand)):
        for tt in range(len(Due[i])):
            if ((Due[i][tt][0])>t) and (Due[i][tt][0] <= (t+Sc+Tc)):
                Dem[i].append(Due[i][tt])
    #print "Dem:", Dem
    for i in range(len(Dem)):
        for j in range(len(Dem[i])):
            Sum[i]+=Dem[i][j][1]
    SumD=Sum
    #print SumD
    return SumD

def NextC(Inv,t): # finds the next campaign to be run based on the final Inventory
    DP=[]
    TotalCost=[] # Cost of inventory at the end of the t+(m+q) horizon
    SUM=[]
    print Inv
    for c in range(len(C)):
        SUM=ActualDem(t,ST[c],T[c])
        #print " "
        HCost=0
        BCost=0
        CostInv=0
        for p in range(len(P)):
            #print "Inv[p]:", Inv[p]
            INV=Inv[p]+R[c]*Alpha[c][p]*T[c]-SUM[p]
            #print " "

```

```

#print "P:", p, "INV:", INV
if INV>0:
    HCost=V[p]*INV #inventory value
    CostInv+=HCost
    #print "HCost:", HCost
else:
    BCost=V[p]*(-INV) #lost sale value
    CostInv+=BCost
    #print "BCost:", BCost
TotalCost.append(CostInv)
#print "c:", c, "Cost:", CostInv

A=list(TotalCost)
A.sort()
CampSelect=TotalCost.index(A[0])
return CampSelect

Wbook = Workbook() # writing output
sheet1 = Wbook.add_sheet('Inventory')
sheet2 = Wbook.add_sheet('HoldingCost')
sheet3= Wbook.add_sheet('Backorder')

r=0
t=0
Nums=[0 for i in range(len(C))]
List=[]
CampList=[]
tt=0
u=0
v=1

for j in range(nP):
    sheet1.write(0,j+1,j+1)

#print "X:", X
while t<=SimTimeP:
    #print "ActDemand:", ActDemand
    Camp=NextC(Inv,t)
    ProT=round((t+ST[Camp]+T[Camp]),0) # time at the end of producing campaign
    for j in range(len(Inv)):
        Inv[j]-=R[Camp]*Alpha[Camp][j]*ST[Camp]
    List.append(Camp)
    while t<ProT: # calculates Inventory over production time

        Dem=[0 for k in range(len(Demand))]
        for i in range(len(Due)):

```

```

    for j in range(len(Due[i])):
        if int(Due[i][j][0]) == (t+1):
            Dem[i]+= Due[i][j][1]
            if Dem[i]<0:
                print Dem

for j in range(len(Inv)):
    Inv[j]+=R[Camp]*Alpha[Camp][j]-Dem[j]
    #print "InvStep:", [j], Inv[j], t
    if Inv[j]>=0:
        InvCosts=HoldingCost*Inv[j]*V[j]/(TotTime)
        sheet1.write(r+1,j+1,Inv[j])
        sheet2.write(r+1,j+1,InvCosts)
    else:
        InvCosts=10*HoldingCost*Inv[j]*V[j]/(TotTime)
        sheet1.write(r+1,j+1,Inv[j])
        #print j, InvCostNeg[j]
        sheet3.write(r+1,j+1,InvCosts)

if t==TotTime+1:
    print "Number of campaigns in the first year:", Nums
if t==2*TotTime+1:
    print "Number of campaigns in the second year:", Nums
if t==3*TotTime+1:
    print "Number of campaigns in the third year:", Nums
sheet1.write(r+1,0,t+1)
t+=tunit
r+=1

    Nums[Camp]+=1
print "Number of campaigns running over the simulation time:", Nums
Wbook.save("SchedulingCost.xls")
Wbook = Workbook()
sheet1 = Wbook.add_sheet('SimDemand')
kkk=0
mmm=1
for j in range(len(Due)):

    for i in range(len(Due[j])):
        sheet1.write(i+1,kkk,Due[j][i][0])
        sheet1.write(i+1,mmm,Due[j][i][1])

    mmm+=2
    kkk+=2
Wbook.save("Demand-SchedulingCost.xls")
print "Go to the excel file"

```

### Approach 3

```
import random
from operator import itemgetter
from xlrd import open_workbook
bookD=open_workbook('SimFile.xls')
sheetD=bookD.sheet_by_index(0)

Runs=[13, 26, 6.5, 6.5, 6.5, 6.5, 6.5, 13, 26, 13, 13, 13]

nP=70 #number of products
nC=12 # number of campaigns

P=[i+1 for i in range(nP)]
C=[j+1 for j in range(nC)] # Here We do not bring the campaigns with N=0 and T=0

Demand=[]
V=[] # Inventory cost per time unit.
BackCost=[]
for i in range(nP):
    d=sheetD.cell(i+1,0).value
    v=sheetD.cell(i+1,1).value
    bb=sheetD.cell(i+1,1).value/.2
    Demand.append(d)
    V.append(v)
    BackCost.append(bb)
a=0
for i in range(len(Demand)):
    a+=Demand[i]

print "Enter each Product's demand arrival rate for C1...C6"
DemandArrRate=[23, 13, 41, 27, 18, 13, 20, 22, 46, 35, 15, 29, 13, 23, 23, 11, 21, 26, 28, 30, 39, 41,
14, 11, 22, 21, 46, 40, 9, 27, 37, 46, 17, 43, 9, 10, 19, 7, 47, 9, 36, 18, 16, 33, 21, 10, 30, 28, 17, 21,
13, 21, 36, 13, 6, 26, 27, 17, 30, 41, 35, 12, 7, 23, 19, 20, 31, 14, 28, 20]
print DemandArrRate
Dev=[(Demand[i]/float(DemandArrRate[i])).25 for i in range(nP)]
MaxDem=[]
MinDem=[]
for i in range(len(Demand)):
    MaxDem.append(Demand[i]/DemandArrRate[i]+Dev[i])
    MinDem.append(Demand[i]/DemandArrRate[i]-Dev[i])

tDay=7 #number of hours per day
nDay=5 #number of days per week
nWeek=52 #number of weeks per year
TotTime= nDay*tDay*nWeek #Total available time per year
```

```

SimTime=6*TotTime

nRuns=[]
for i in range(len(Runs)):
    nRuns.append(Runs[i]/float(TotTime))

LamArrTime=0.4
LamDemSize=0.4
#print "SS:", SS

#-----
# FORECASTED and Actual Demand based on hour time base
#-----
Forecast=[[[]for k in range(len(P))]
Time=[0 for i in range(len(P))]
random.seed(11111111)
Due=[[[]for k in range(len(P))]
TimeActual=[0 for i in range(len(P))]
for i in range(len(P)):
    AvgDTime=TotTime/float(DemandArrRate[i])
    AvgDSize=(MaxDem[i]+MinDem[i])/2.
    Time[i]+=AvgDTime
    #print "AvgDTime:", AvgDTime
    #print "AvgDSize:", AvgDSize
    while Time[i]<SimTime and TimeActual[i]<SimTime:
        #print "TIME BET:", TimeBetArr
        ArrTimeD=float(random.expovariate(float(DemandArrRate[i]/float(TotTime))))
        DSize=random.uniform(MinDem[i], MaxDem[i])
        TimeActual[i]+=ArrTimeD
        TimeForecast=LamArrTime*ArrTimeD+(1-LamArrTime)*AvgDTime
        AvgDTime=TimeForecast
        DSizeForecast=LamDemSize*DSize+(1-LamDemSize)*AvgDSize
        AvgDSize=DSizeForecast
        Time[i]+=AvgDTime
    if Time[i]>SimTime and TimeActual[i]>SimTime:
        break
    Due[i].append([(int(TimeActual[i])),DSize,i,"A"])
    Forecast[i].append([(int(Time[i])),AvgDSize,i,"F"])

##### INVENTORY #####
from xlwt import Workbook
sheetC=bookD.sheet_by_index(1)
sheetA=bookD.sheet_by_index(2)

tunit=1 #hour
SimTimeP=4*TotTime #(run for 4 years)

```



```

N=[]
T=[]
R=[]
ST=[]
for i in range(nC):
    nn=sheetC.cell(i+1,1).value
    N.append(nn)

    tp=sheetC.cell(i+1,2).value
    T.append(tp)

    rp=sheetC.cell(i+1,3).value
    R.append(rp)

    st=sheetC.cell(i+1,4).value
    ST.append(st)

K=[0,1,2,3,4]

Alpha=[[[] for i in range(nC)]
for i in range(len(Alpha)):
    for j in range(nP):
        ss=sheetA.cell(i+1,j+1).value
        Alpha[i].append(ss)

Delta=[]
for i in range(nP):
    Delta.append(Demand[i]/a)

Inv=[0 for i in range(len(P))]

def ActualDem(t): # Demands for m+q periods including actual and forecasted demands
    Dem=[[[] for i in range(len(P))]
for i in range(len(Demand)):
    for tt in range(len(Due[i])):
        if ((Due[i][tt][0])>t) and (Due[i][tt][0] <= (t+m*nDay*tDay)):
            Dem[i].append(Due[i][tt])
        if ((Forecast[i][tt][0]) >(t+m*nDay*tDay)) and ((Forecast[i][tt][0])<=
(t+(m+q)*nDay*tDay)):
            Dem[i].append(Forecast[i][tt])
    #print "Dem:", Dem
    return Dem

```

```

def Func0(t): # sumation of demands for m+q periods
    Sum=[0 for i in range(len(Demand))]
    ActDem=ActualDem(t)
    for i in range(len(ActDem)):
        for j in range(len(ActDem[i])):
            Sum[i]+=ActDem[i][j][1]
    SumD=Sum
    return SumD

def NumC(number,t):
    #print "time:", t
    NoCalc=[0 for i in range(nC)]
    for i in range((nC)):
        NoCalc[i]=nRuns[i]*t
    DifCalc=dict([(i, NoCalc[i]-number[i])for i in range((nC))])
    q=sorted(DifCalc.items(), key=itemgetter(1))
    CampSort=q[nC-1][0]
    #print "sort2:", CampSort
    return CampSort

def NextC(t):
    Camp2=NumC(Nums, t)
    NextCamp=Camp2
    return NextCamp

Wbook = Workbook() # writing output
sheet1 = Wbook.add_sheet('Inventory')
sheet2 = Wbook.add_sheet('HoldingCost')
sheet3= Wbook.add_sheet('LostSale')

r=0
t=0
Nums=[0 for i in range(len(C))]
List=[]
CampList=[]
tt=0
u=0
v=1

for j in range(nP):
    sheet1.write(0,j+1,j+1)

#print "X:", X
while t<=SimTimeP:
    Camp=NextC(t)
    ProT=round((t+ST[Camp]+T[Camp]),0) # time at the end of producing campaign

```

```

for j in range(len(Inv)):
    Inv[j]-=R[Camp]*Alpha[Camp][j]*ST[Camp]
List.append(Camp)
while t<ProT: # calculates Inventory over production time
    Dem=[0 for k in range(len(Demand))]
    for i in range(len(Due)):
        for j in range(len(Due[i])):
            if int(Due[i][j][0]) == (t+1):
                #print t, Dem[i]
                Dem[i]+= Due[i][j][1]
                #print "after", Dem[i]
                #print " "
                if Dem[i]<0:
                    print Dem

    for j in range(len(Inv)):
        Inv[j]+=R[Camp]*Alpha[Camp][j]-Dem[j]
        if Inv[j]>=0:
            InvCosts=HoldingCost*Inv[j]*V[j]/(TotTime)
            sheet1.write(r+1,j+1,Inv[j])
            sheet2.write(r+1,j+1,InvCosts)
        else:
            InvCosts=10*V[j]*(Inv[j])/TotTime
            sheet3.write(r+1,j+1,InvCosts)
            #print j, InvCostNeg[j]
            sheet1.write(r+1,j+1,Inv[j])

    if t==TotTime+1:
        print "Number of campaigns in the first year:", Nums
    if t==2*TotTime+1:
        print "Number of campaigns in the second year:", Nums
    if t==3*TotTime+1:
        print "Number of campaigns in the third year:", Nums
    sheet1.write(r+1,0,t+1)
    t+=tunit
    r+=1

    Nums[Camp]+=1
print "Number of campaigns running over the simulation time:", Nums
Wbook.save("SchedulingNum.xls")
Wbook = Workbook()
sheet1 = Wbook.add_sheet('SimDemand')
kkk=0
mmm=1
for j in range(len(Due)):

```

```
for i in range(len(Due[j])):
    sheet1.write(i+1, kkk, Due[j][i][0])
    sheet1.write(i+1, mmm, Due[j][i][1])

    mmm+=2
    kkk+=2
Wbook.save("Demand-SchedulingNum.xls")
print "Go to the excel file"
```

#### Approach 4

```
import random
from operator import itemgetter
from xlrd import open_workbook
bookD=open_workbook('SimFile.xls')
sheetD=bookD.sheet_by_index(0)
Yes=0
Runs=[13, 26, 6.5, 6.5, 6.5, 6.5, 6.5, 13, 26, 13, 13, 13]

nP=70 #number of products
nC=12 # number of campaigns

P=[i+1 for i in range(nP)]
C=[j+1 for j in range(nC)] # Here We do not bring the campaigns with N=0 and T=0

Demand=[]
V=[] # Inventory cost per time unit.
BackCost=[]
for i in range(nP):
    d=sheetD.cell(i+1,0).value
    v=sheetD.cell(i+1,1).value
    Demand.append(d)
    V.append(v)
a=0
for i in range(len(Demand)):
    a+=Demand[i]

print "Enter each Product's demand arrival rate for C1...C6"
DemandArrRate=[23, 13, 41, 27, 18, 13, 20, 22, 46, 35, 15, 29, 13, 23, 23, 11, 21, 26, 28, 30, 39, 41,
14, 11, 22, 21, 46, 40, 9, 27, 37, 46, 17, 43, 9, 10, 19, 7, 47, 9, 36, 18, 16, 33, 21, 10, 30, 28, 17, 21,
13, 21, 36, 13, 6, 26, 27, 17, 30, 41, 35, 12, 7, 23, 19, 20, 31, 14, 28, 20]
print DemandArrRate
Dev=[(Demand[i]/float(DemandArrRate[i])*0.25) for i in range(nP)]
MaxDem=[]
MinDem=[]
for i in range(len(Demand)):
    MaxDem.append(Demand[i]/DemandArrRate[i]+Dev[i])
    MinDem.append(Demand[i]/DemandArrRate[i]-Dev[i])
tDay=7 #number of hours per day
nDay=5 #number of days per week
nWeek=52 #number of weeks per year
TotTime= nDay*tDay*nWeek #Total available time per year
SimTime=6*TotTime

nRuns=[]
```

```

for i in range(len(Runs)):
    nRuns.append(Runs[i]/float(TotTime))
LamArrTime=0.4
LamDemSize=0.4

#-----
# FORECASTED and Actual Demand based on hour time base
#-----
Forecast=[[]for k in range(len(P))]
Time=[0 for i in range(len(P))]
random.seed(111111111)
Due=[[]for k in range(len(P))]
TimeActual=[0 for i in range(len(P))]
for i in range(len(P)):
    AvgDTime=TotTime/float(DemandArrRate[i])
    AvgDSize=(MaxDem[i]+MinDem[i])/2.
    Time[i]+=AvgDTime
    while Time[i]<SimTime and TimeActual[i]<SimTime:
        ArrTimeD=float(random.expovariate(float(DemandArrRate[i]/float(TotTime))))
        DSize=random.uniform(MinDem[i], MaxDem[i])
        TimeActual[i]+=ArrTimeD
        TimeForecast=LamArrTime*ArrTimeD+(1-LamArrTime)*AvgDTime
        AvgDTime=TimeForecast
        DSizeForecast=LamDemSize*DSize+(1-LamDemSize)*AvgDSize
        AvgDSize=DSizeForecast
        Time[i]+=AvgDTime
    if Time[i]>SimTime and TimeActual[i]>SimTime:
        break
    Due[i].append([(int(TimeActual[i])),DSize,i,"A"])
    Forecast[i].append([(int(Time[i])),AvgDSize,i,"F"])

##### INVENTORY #####
from xlwt import Workbook
sheetC=bookD.sheet_by_index(1)
sheetA=bookD.sheet_by_index(2)

tunit=1 #hour
SimTimeP=4*TotTime #(run for 4 years)

N=[]
T=[]
R=[]
ST=[]
for i in range(nC):

```

```

nn=sheetC.cell(i+1,1).value
N.append(nn)

tp=sheetC.cell(i+1,2).value
T.append(tp)

rp=sheetC.cell(i+1,3).value
R.append(rp)

st=sheetC.cell(i+1,4).value
ST.append(st)

K=[0,1,2,3,4]

Alpha=[]
for i in range(nC):
    for j in range(len(Alpha)):
        for j in range(nP):
            ss=sheetA.cell(i+1,j+1).value
            Alpha[i].append(ss)

Inv=[0 for i in range(len(P))]

def ActualDem(t,Sc,Tc): # summation of demands for Tc+Sc duration of each campaign run
    Dem=[]
    Sum=[0 for i in range(len(Demand))]
    for i in range(len(Demand)):
        for tt in range(len(Due[i])):
            if ((Due[i][tt][0]>t) and (Due[i][tt][0] <= (t+Sc+Tc))):
                Dem[i].append(Due[i][tt])
    for i in range(len(Dem)):
        for j in range(len(Dem[i])):
            Sum[i]+=Dem[i][j][1]
    SumD=Sum
    return SumD

def Func1(Inv,t): # finds the next campaign to be run based on the final Inventory
    INV=0
    CampSelect=[]
    DP=[]
    TotalCost=[] # Cost of inventory at the end of the t+(m+q) horizon
    SUM=[]
    for c in range(len(C)):
        SUM=ActualDem(t,ST[c],T[c])
        HCost=0
        BCost=0
        CostInv=0

```

```

for p in range(len(P)):
    INV=Inv[p]+R[c]*Alpha[c][p]*T[c]-SUM[p]
    if INV>0:
        HCost=V[p]*INV#inventory value
        CostInv+=HCost
        #print "HCost:", HCost
    else:
        BCost=V[p]*(-INV) #lost sale value
        CostInv+=BCost
        #print "BCost:", BCost
    TotalCost.append(CostInv)
#print "c:", c, "Cost:", TotalCost

A=list(TotalCost)
A.sort()
CampSelect.append(TotalCost.index(A[0]))
CampSelect.append(TotalCost.index(A[1]))
CampSelect.append(TotalCost.index(A[2]))
return CampSelect

def NumC(number,t):
    CampSort=[]
    NoCalc=[0 for i in range(nC)]
    for i in range((nC)):
        NoCalc[i]=nRuns[i]*t
    DifCalc=dict([(i, NoCalc[i]-number[i])for i in range((nC))])
    q=sorted(DifCalc.items(), key=itemgetter(1))
    CampSort.append(q[nC-1][0])
    CampSort.append(q[nC-2][0])
    CampSort.append(q[nC-3][0])
    return CampSort

def NextC(t):
    Yes=0
    Camp1=Func1(Inv,t)
    Camp2=NumC(Nums, t)
    NextCamp=Camp2[0]
    for i in range(len(Camp2)):
        for j in range(len(Camp1)):
            if Camp2[i]==Camp1[j]:
                NextCamp=Camp1[j]
                Yes=1
    return NextCamp

Wbook = Workbook() # writing output

```



```

sheet1 = Wbook.add_sheet('Inventory')
sheet2 = Wbook.add_sheet('Holding')
sheet3= Wbook.add_sheet('BackOrder')

r=0
t=0
Nums=[0 for i in range(len(C))]
List=[]
CampList=[]
tt=0
u=0
v=1

for j in range(nP):
    sheet1.write(0,j+1,j+1)

#print "X:", X
while t<=SimTimeP:
    #print "ActDemand:", ActDemand
    Camp=NextC(t)
    ProT=round((t+ST[Camp]+T[Camp]),0) # time at the end of producing campaign
    #print "Camp:", Camp
    #print "ProT:", ProT
    for j in range(len(Inv)):
        Inv[j]-=R[Camp]*Alpha[Camp][j]*ST[Camp]
    List.append(Camp)
    while t<ProT: # calculates Inventory over production time
        #print "t:", t
        #print " "
        Dem=[0 for k in range(len(Demand))]
        for i in range(len(Due)):
            for j in range(len(Due[i])):
                if int(Due[i][j][0]) == (t+1):
                    Dem[i]+= Due[i][j][1]
                if Dem[i]<0:
                    print Dem

    #print "Dem", Dem

    for j in range(len(Inv)):
        Inv[j]+=R[Camp]*Alpha[Camp][j]-Dem[j]
        #print "InvStep:", [j], Inv[j], t
        if Inv[j]>=0:
            InvCosts=HoldingCost*Inv[j]*V[j]/(TotTime)
            sheet1.write(r+1,j+1,Inv[j])
            sheet2.write(r+1,j+1,InvCosts)

```

```

else:
    InvCosts=10*V[j]*(Inv[j])/(TotTime)
    sheet3.write(r+1,j+1,InvCosts)
    #print j, InvCostNeg[j]
    sheet1.write(r+1,j+1,Inv[j])
if t==TotTime+1:
    print "Number of campaigns in the first year:", Nums
if t==2*TotTime+1:
    print "Number of campaigns in the second year:", Nums
if t==3*TotTime+1:
    print "Number of campaigns in the third year:", Nums
sheet1.write(r+1,0,t+1)
t+=tunit
r+=1

Nums[Camp]+=1
#print Nums
print "Number of campaigns running over the simulation time:", Nums
Wbook.save("SchedulingMixCostNum.xls")
Wbook = Workbook()
sheet1 = Wbook.add_sheet('SimDemand')
sheet2= Wbook.add_sheet('ForecastDemand')
kkk=0
mmm=1
nnn=0
lll=1
for j in range(len(Due)):

    for i in range(len(Due[j])):
        sheet1.write(i+1,kkk,Due[j][i][0])
        sheet1.write(i+1,mmm,Due[j][i][1])
        mmm+=2
        kkk+=2

Wbook.save("Demand-MixCostNum.xls")
#print "Number of adaptation:", Yes
print "Go to the excel file"

```

## Approach 5

```
import random
from operator import itemgetter
from xlrd import open_workbook
bookD=open_workbook('SimFile.xls')
sheetD=bookD.sheet_by_index(0)
Yes=0
Runs=[13, 26, 6.5, 6.5, 6.5, 6.5, 6.5, 13, 26, 13, 13, 13]

nP=70 #number of products
nC=12 # number of campaigns

P=[i+1 for i in range(nP)]
C=[j+1 for j in range(nC)] # Here We do not bring the campaigns with N=0 and T=0

Demand=[]
V=[] # Inventory cost per time unit.
BackCost=[]
for i in range(nP):
    d=sheetD.cell(i+1,0).value
    v=sheetD.cell(i+1,1).value
    bb=sheetD.cell(i+1,1).value/.2
    Demand.append(d)
    V.append(v)
    BackCost.append(bb)
a=0
for i in range(len(Demand)):
    a+=Demand[i]
print "Enter each Product's demand arrival rate for C1...C6"
DemandArrRate=[23, 13, 41, 27, 18, 13, 20, 22, 46, 35, 15, 29, 13, 23, 23, 11, 21, 26, 28, 30, 39, 41,
14, 11, 22, 21, 46, 40, 9, 27, 37, 46, 17, 43, 9, 10, 19, 7, 47, 9, 36, 18, 16, 33, 21, 10, 30, 28, 17, 21,
13, 21, 36, 13, 6, 26, 27, 17, 30, 41, 35, 12, 7, 23, 19, 20, 31, 14, 28, 20]
print DemandArrRate
Dev=[(Demand[i]/float(DemandArrRate[i])).25 for i in range(nP)]
MaxDem=[]
MinDem=[]
for i in range(len(Demand)):
    MaxDem.append(Demand[i]/DemandArrRate[i]+Dev[i])
    MinDem.append(Demand[i]/DemandArrRate[i]-Dev[i])

tDay=7 #number of hours per day
nDay=5 #number of days per week
nWeek=52 #number of weeks per year
TotTime= nDay*tDay*nWeek #Total available time per year
SimTime=6*TotTime
```

```

nRuns=[]
for i in range(len(Runs)):
    nRuns.append(Runs[i]/float(TotTime))

LamArrTime=0.4
LamDemSize=0.4

#-----
# FORECASTED and Actual Demand based on hour time base
#-----
Forecast=[[[]for k in range(len(P))]
Time=[0 for i in range(len(P))]
random.seed(111111111)
Due=[[[]for k in range(len(P))]
TimeActual=[0 for i in range(len(P))]
for i in range(len(P)):
    AvgDTime=TotTime/float(DemandArrRate[i])
    AvgDSize=(MaxDem[i]+MinDem[i])/2.
    Time[i]+=AvgDTime
    while Time[i]<SimTime and TimeActual[i]<SimTime:
        ArrTimeD=float(random.expovariate(float(DemandArrRate[i]/float(TotTime))))
        DSize=random.uniform(MinDem[i], MaxDem[i])
        TimeActual[i]+=ArrTimeD
        TimeForecast=LamArrTime*ArrTimeD+(1-LamArrTime)*AvgDTime
        AvgDTime=TimeForecast
        DSizeForecast=LamDemSize*DSize+(1-LamDemSize)*AvgDSize
        AvgDSize=DSizeForecast
        Time[i]+=AvgDTime
    if Time[i]>SimTime and TimeActual[i]>SimTime:
        break
    Due[i].append([(int(TimeActual[i])),DSize,i,"A"])
    Forecast[i].append([(int(Time[i])),AvgDSize,i,"F"])

##### INVENTORY #####
from xlwt import Workbook
sheetC=bookD.sheet_by_index(1)
sheetA=bookD.sheet_by_index(2)

tunit=1 #hour
SimTimeP=4*TotTime #(run for 4 years)

N=[]
T=[]
R=[]
ST=[]

```

```

for i in range(nC):
    nn=sheetC.cell(i+1,1).value
    N.append(nn)

    tp=sheetC.cell(i+1,2).value
    T.append(tp)

    rp=sheetC.cell(i+1,3).value
    R.append(rp)

    st=sheetC.cell(i+1,4).value
    ST.append(st)

K=[0,1,2,3,4]

Alpha=[]
for i in range(nC):
    for j in range(len(Alpha)):
        for j in range(nP):
            ss=sheetA.cell(i+1,j+1).value
            Alpha[i].append(ss)

Inv=[0 for i in range(len(P))]

def ActualDem(t,Sc,Tc): # summation of demands for Tc+Sc duration of each campaign run
    Dem=[]
    for i in range(len(P)):
        Sum=[0 for i in range(len(Demand))]
        for i in range(len(Demand)):
            for tt in range(len(Due[i])):
                if ((Due[i][tt][0])>t) and (Due[i][tt][0] <= (t+Sc+Tc)):
                    Dem[i].append(Due[i][tt])
            for i in range(len(Dem)):
                for j in range(len(Dem[i])):
                    Sum[i]+=Dem[i][j][1]
        SumD=Sum
    return SumD

def Func1(Inv,t): # finds the next campaign to be run based on the final Inventory
    INV=0
    CampSelect=[]
    DP=[]
    TotalCost=[] # Cost of inventory at the end of the t+(m+q) horizon
    SUM=[]
    for c in range(len(C)):
        SUM=ActualDem(t,ST[c],T[c])
        HCost=0
        BCost=0

```

```

CostInv=0
for p in range(len(P)):
    INV=Inv[p]+R[c]*Alpha[c][p]*T[c]-SUM[p]
    if INV>0:
        HCost=V[p]*INV#inventory value
        CostInv+=HCost
    else:
        BCost=V[p]*(-INV) #lost sale value
        CostInv+=BCost
TotalCost.append(CostInv)

A=list(TotalCost)
A.sort()
CampSelect.append(TotalCost.index(A[0]))
CampSelect.append(TotalCost.index(A[1]))
CampSelect.append(TotalCost.index(A[2]))
return CampSelect

def NumC(number,t):
    CampSort=[]
    NoCalc=[0 for i in range(nC)]
    for i in range((nC)):
        NoCalc[i]=nRuns[i]*t
    DifCalc=dict([(i, NoCalc[i]-number[i])for i in range((nC))])
    q=sorted(DifCalc.items(), key=itemgetter(1))
    CampSort.append(q[nC-1][0])
    CampSort.append(q[nC-2][0])
    CampSort.append(q[nC-3][0])
    #print "sort2:", CampSort
    return CampSort

def NextC(t):
    Yes=0
    Camp1=Func1(Inv,t)
    Camp2=NumC(Nums, t)
    NextCamp=Camp1[0]
    for i in range(len(Camp1)):
        for j in range(len(Camp2)):
            if Camp1[i]==Camp2[j]:
                NextCamp=Camp1[i]
                Yes=1
    return NextCamp

Wbook = Workbook() # writing output
sheet1 = Wbook.add_sheet('Inventory')

```

```

sheet2 = Wbook.add_sheet('Holding')
sheet3= Wbook.add_sheet('BackOrder')

r=0
t=0
Nums=[0 for i in range(len(C))]
List=[]
CampList=[]
tt=0
u=0
v=1

for j in range(nP):
    sheet1.write(0,j+1,j+1)

while t<=SimTimeP:
    Camp=NextC(t)
    ProT=round((t+ST[Camp]+T[Camp]),0) # time at the end of producing campaign
    for j in range(len(Inv)):
        Inv[j]-=R[Camp]*Alpha[Camp][j]*ST[Camp]
    List.append(Camp)
    while t<ProT: # calculates Inventory over production time
        Dem=[0 for k in range(len(Demand))]
        for i in range(len(Due)):
            for j in range(len(Due[i])):
                if int(Due[i][j][0]) == (t+1):
                    Dem[i]+= Due[i][j][1]
                if Dem[i]<0:
                    print Dem

        for j in range(len(Inv)):
            Inv[j]+=R[Camp]*Alpha[Camp][j]-Dem[j]
            #print "InvStep:", [j], Inv[j], t
            if Inv[j]>=0:
                InvCosts=HoldingCost*Inv[j]*V[j]/(TotTime)
                sheet1.write(r+1,j+1,Inv[j])
                sheet2.write(r+1,j+1,InvCosts)
            else:
                InvCosts=10*V[j]*(Inv[j])/TotTime
                sheet3.write(r+1,j+1,InvCosts)
                sheet1.write(r+1,j+1,Inv[j])

        if t==TotTime+1:
            print "Number of campaigns in the first year:", Nums
        if t==2*TotTime+1:
            print "Number of campaigns in the second year:", Nums

```

```

if t==3*ToTime+1:
    print "Number of campaigns in the third year:", Nums
    sheet1.write(r+1,0,t+1)
    t+=tunit
    r+=1

    Nums[Camp]+=1
    print "Number of campaigns running over the simulation time:", Nums
    #print "List:", List
    Wbook.save("SchedulingMixCostNum.xls")
    Wbook = Workbook()
    sheet1 = Wbook.add_sheet('SimDemand')
    sheet2= Wbook.add_sheet('ForecastDemand')
    kkk=0
    mmm=1
    nnn=0
    lll=1
    for j in range(len(Due)):
        for i in range(len(Due[j])):
            sheet1.write(i+1,kkk,Due[j][i][0])
            sheet1.write(i+1,mmm,Due[j][i][1])
            mmm+=2
            kkk+=2

    Wbook.save("Demand-MixCostNum.xls")
    print "Go to the excel file"

```