MEDIUM TERM PRODUCTION PLANNING AND CAMPAIGN SCHEDULING FOR SAWMILL

by

Sina Saadatyar

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

at

Dalhousie University
Halifax, Nova Scotia
December 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 “Medium Term Production Planning and Campaign Scheduling for Sawmill” by Sina Saadatyar in partial fulfilment of the requirements for the degree of Master of Applied Science.

Dated December 7, 2012

Co-Supervisors: _____

_____

Readers: _____

_____

DALHOUSIE UNIVERSITY

DATE: December 7, 2012

AUTHOR: Sina Saadatyar

TITLE: Medium Term Production Planning and Campaign Scheduling for Sawmill

DEPARTMENT OR SCHOOL: Industrial Engineering

DEGREE: M.A.Sc. CONVOCATION: May YEAR: 2013

_____

Signature of Author

In memory of my beloved father, ***Masoud***

He is always in my thoughts and my heart

Dedicated to

my dear mother, ***Mahin***

and my lovely brother and sister, ***Siamak & Samira***

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this thesis, we study a multi-period, multi-product, production planning problem for the lumber industry, and present a hierarchical approach to control and schedule lumber production in a sawmill.

First, at the sawing unit, the lower level of the hierarchical structure, the combination of log classes, price lists and sawing patterns defines the expected output distribution in terms of lumber pieces. A price list, defining the value of outputs, is fed directly to the sawmill production control optimizer to select the best sawing patterns. This results in a broad variety of different lumber outputs.

Second, at the upper level, a mixed integer programming model has been proposed to maximize the total revenue at the sawmill. The lumber outputs determined at the lower level are used as data at the upper level. Market demand, lumber inventory cost, and supply cost are considered over the planning horizon. The proposed model has been developed and implemented on a real-scale prototype sawmill.

ACKNOWLEDGMENT

First and foremost, I would like to express my gratitude to Dr. Eldon Gunn for his supervision, advice, and guidance throughout my studies. Above all and the most needed, he provided me encouragement and support in various ways. Without him, this thesis would not have been possible.

I would like to thank sincerely from my co-supervisors Dr. Corinne MacDonald for her constructive supervision and enormous support during my master thesis.

I thank my supervisory committee, Dr. Yan Feng and Dr. William J. Phillips for their positive guidance.

I thank my dear colleague, Pegah Sohrabi for her kind support during my study.

I also thank our VCO research group members, Md. Shariful Islam and Andrew B. Matrin for their helpful comments.

Last but not least, it shall be emphasized that the continuous support of my family and my friends Ehsan, Mohammad, Pouria and Zahra.

Chapter 1

Introduction

The problem we are going to explore in this study is a multi-product, multi-period production planning problem in a sawmill. This problem arises because a diverse assortment of products is produced by various sawing processes from different "classes" of logs. Logs, the raw materials in this problem, are classified according to similar characteristics such as diameter or length; these characteristics define "log classes", though, within each, class they vary in shape. We need to find ways of planning the sawing process to meet a variable demand for products over the planning horizon in the forest supply chain.

Both Hax and Meal [17] and Anthony [1] in their studies divided supply chain planning into three major levels: strategic, tactical and operational. At the strategic level long term decisions are made to support the major objective of the organization. These decisions could be plant capacity, production technology, logistic networks and strategic marketing decisions. Strategic decisions usually occur at a timeframe of several years. The next level, the tactical level, makes midterm decisions including production levels or inventory levels and with a time frame of typically a few months. Finally, the operational level schedules operations to assure in-time delivery of final products to fulfill customer demand; the timeframe is anywhere from days to weeks. For instance daily schedules for process sequences are categorized as an operational level decision. Ronnqvist [28] categorized the planning levels in the forest supply chain in four levels. In addition to strategic, tactical and operational levels, he added online line planning which covers

decisions with a timeframe of less than a day. Table 1, taken from his study, illustrates these levels.

Table 1- Planning Levels in Forestry (after Ronnqvist [28])

	Forest management and harvesting	**Transportation and routing**	**Production**
Strategic Planning	Planting, Evaluation, Long time harvesting	Road building, road upgrading, fleet management	Investment planning
Tactical Planning	Annual harvest plans	Road upgrade, equipment utilization	Annual production planning
Operational Planning	Crew's scheduling, harvest sequencing	Catchment areas, back-haulage planning, scheduling	Lot sizing, scheduling
Online Planning	Bucking	Truck dispatching	Process control, Roll cutting, Cross-cutting

In this study we are focusing on the production part of the forest supply chain. Ronnqvist [28] in his study explains several stages of lumber production as follows. When trees are cut, they are bucked into different lengths and transported to sawmills. At the sawmill, before sawing logs are usually sorted based on different specifications including, species, length and top end diameter. Then, during the sawing process, the sorted logs are broken down into different lumber dimensions determined by the chosen sawing patterns. Typically, the sawn lumber is then required to be dried in a kiln. Drying duration depends on characteristics such as moisture content, species and dimensions of the lumber. The last stage is planing and grading; dried lumber is planed, trimmed and graded according to its quality. The finished products are then sorted and transported to

the market to fulfill costumer demand. Figure 1 shows the sequence of processes for producing lumber.

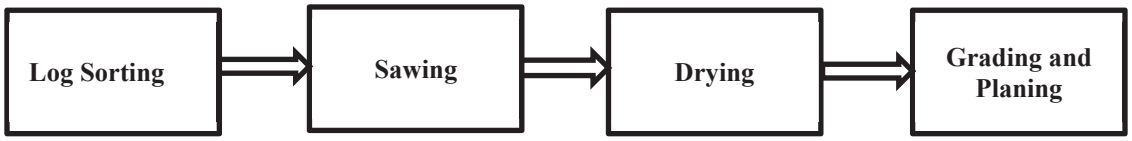


Figure 1- Lumber Production Sequence

Significant increases in the efficiency of lumber production can be realized by improving production scheduling at the sawing unit. Production scheduling is where appropriate cutting patterns are applied to suitable log pieces to produce the lumber products to fulfill customer demand. A consideration in production scheduling is the log supply, which consists of logs in different sizes and shapes. Sawing patterns dictate how to break the log down into lumber pieces. For any given log, multiple potential sawing patterns exist. The dimensions of the log and the chosen sawing pattern determine the lumber pieces generated. The sawing process simultaneously produces a mix of lumber pieces by applying the sawing pattern. This mix, referred to as a "basket of products" in some literature, coupled with nonconstant demand for lumber pieces can result in over or under production of certain products. The role of control in this procedure is to determine which logs to process and how to saw lumber pieces in order to produce outputs according to the market demand. Feedback for this control system is the current market situation. Figure 2 illustrates the general control system loop for a sawmill.

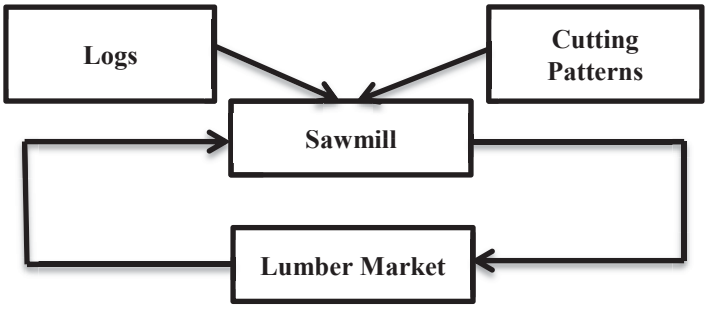


Figure 2- General Sawmill Control System

The fundamental task of any sawmill control system is to determine the output of a specific log. According to the nature of trees, each log has a unique shape and, thus is processed individually. This makes the modeling and optimizing of the sawing units complicated. In this study similar logs are grouped according to "classes", determined by characteristics such as species, diameter and/or length. Although the optimizer selects the best cutting pattern for each log individually, this does not mean logs are fed into it of random; in industry logs are usually sorted into classes in the mill yard prior to the sawing process and then all logs from one class are sawn before those of another class, so the model reports the results by class and by log. The price list is used as a control policy to lead the optimizer to choose the sawing patterns that achieve the most desirable lumber outputs. By combining log classes and price lists, "campaigns" are created through the sawing process. We use campaigns, a unique combination of log class and price list, as a way of modeling the planning process.

After calculating the lumber outputs from each campaign, the next step is to determine how to schedule these potential campaigns in order to maximize the revenue at the sawmill by considering inventory and supply costs. The proposed mixed integer programming model will schedule the potential campaigns.

In order to deal with uncertainty, our modeling includes a rolling planning horizon; the plant manager will recalculate the production plan at the end of each period based on updated data.

In the next chapter of this thesis, chapter 2, related literature is reviewed and compared with our work. Chapter 3 explains the details of the developed algorithm for creating

campaigns, such as the results of implementing this algorithm to generate more than 100 potential campaigns. Chapter 4 presents a mixed integer programming model for scheduling the potential campaigns from chapter three. Details and results of the mathematical model provided in this chapter. Chapter 5 shows the effectiveness of including a rolling planning horizon in our model. Finally, chapter 6 provides an overall conclusion and suggests possible further work to improve this study.

Chapter 2

Literature Review

In this chapter some studies related to our work will be reviewed in three main sections. The first will focus on sawing cutting patterns and the software developed to generate them. The second covers studies and papers related to production planning and sawmill scheduling. Finally, the third explains rolling planning horizon production and hierarchical structure.

2.1 Cutting Patterns

The fundamental question for each sawmill is how to breakdown logs into required lumber pieces. In this section, we review work that has been done in this area. For some time, optimization methods have been used to solve cutting stock problems. Gilmore and Gomory [11] provided a linear programming approach to the cutting stock problem by relaxing the integer variables; however, the results obtained by rounding the solution values were not economically acceptable. Suter and Calloway [31] in their study said that most lumber producers attempt to find the best cutting solution for specific boards based on traditional methods and their own experiences, and not mathematical model. Carnieria, Mendoza and Gavinho [5] presented a knapsack algorithm (double knapsack) to find the optimal cutting patterns for lumber or composite boards to obtain different combinations of furniture parts. They also provided heuristic methods to find the solution. Later, they [6] extended the previous model, and presented a fractional

algorithm that optimizes cutting boards or lumber into dimension parts by considering flexible customer order.

In addition to knapsack algorithm, dynamic programming has been a useful approach for solving cutting pattern problems. Tejavibulya [32] developed two dynamic programming models to determine optimum sawing solutions. The weak point in his study is that, due to the complexity of the problem, he considered logs as perfect cylinders without any defects or taper. Reinders and Hendriks [27] developed an algorithm for converting trees into lumber pieces. Their algorithm was based on "nested dynamic programming sub-algorithms". They assumed logs were truncated cones and ignored the taper, defects and saw kerfs.

In 1993, Funck and Zeng [8] produced a log breakdown model, SAW3D, which allowed more realistic three dimensional log shapes. They used dynamic programming algorithms to optimize lumber recovery from defectless logs of any shape in three-dimensional space. Two years later they further developed SAW3D and introduced SAW3DG, a model that includes internal log defects. The results showed that SAW3DG provided better solutions than the previous models because it was more realistic [9]. AUTOSAW is a sawing simulation software which considers saw kerfs and allows irregular log shapes [33]. AUTOSAW was implemented and tested in another study by Todoroki and Ronnqvist [34] to maximize the value or volume yield from logs. In another study, they [35] used AUTOSAW with dynamic pricing and three order books to be fulfilled. They concluded that fewer logs were needed and better system performance was obtained. Another study using AUTOSAW was done by Todoroki and Rönnqvist [36] in which they divided log breakdown optimization into two connected stages. At the first stage

logs are broken down into slabs and, at the second stage, slabs cut into boards. They used a dynamic programming formulation where the result of cutting slabs into boards will be given to the first stage to decide the optimal cutting for breaking down the logs into slabs. They showed that their proposed method would significantly increase the resulting value when quality is considered. The sawing method used in [36] is live sawing, where cuts are parallel and are all done by the main saw. They compared three scenarios. The first scenario is aimed to maximize the value of the lumber when quality is considered. In this case the sawmill should be capable of performing internal scanning before the first breakdown, and defect scanning before the second breakdown. The second scenario is focused on optimizing lumber value without considering any defects with a sawmill that is capable of scanning logs and flitches before breaking them down. The third scenario is hybrid scenario where volume is optimized at the first stage and lumber value at the second one. They used AUTOSAW for converting logs into boards and they assumed live sawing is slow and has limited capability for generating different combinations of cutting patterns.

Optitek is a piece of software provided by FPinnovations which optimizes the sawing process of logs into lumber pieces [12]. Although Optitek deals with irregular log shapes, the processing time is too slow to implement industrially, and it is usually used for research purposes and with a limited number of logs. Recently a few studies used this software to simulate the sawing process. Zhang and Tong [41] tried to provide a general second order polynomial equation based on the tree characteristic to estimate lumber output, and compare it with lumber output as determined by Optitek.

Reviewing the literature for sawing cutting patterns, we find several studies and complicated models. Recently, due to more powerful computers, more realistic and complex commercial software has been developed to optimize the sawing process, but the details of their optimization routines are not released.

In this study, since we did not have access to these pieces of software, a quick and novel algorithm to estimate the output of lumber from the sawing process was developed. This algorithm is based on the basic three dimensions of geometry. It can be run fast, less than 30 minutes for 100,000 logs, and it is flexible. This enables the easy generation of campaigns that facilitates the output of a sawmill being easily controlled by the pricing vector. This will be discussed in the next chapter.

Although our method for creating sawing patterns does not include complexities such as irregular shapes of trees or internal defects, it demonstrates a method by which reasonable lumber output of campaign can be estimated. This capability is required for the next part of this study.

2.2 Sawmill Scheduling

In recent years by improving the computers and technology, many complicated and large scale models have been developed and solved to schedule sawmill production. In 1991, Mendoza et al. [25] combined an optimization model with a real time simulation that considers production limitations such as resource capacity. The system works in the following way: at the first stage the mathematical model determines the best log input mix according to periodic lumber demand. Then log input information is used for simulating the sawing process to find the production schedule. They used a lumber

recovery rate percentage for transforming logs into lumber, but details are not explained as to how this rate is calculated. The process simulator gets the required data such as machine characteristic and constraints, the log mix input, found by optimization, and the current inventory. The output will be a sawmill schedule and lumber outputs. This model is not multi period and the dynamic feature of planning is ignored. Log classification and saw setup time are not considered in their system.

Two years later Maness and Adams [21] proposed a model for integrating the processes of bucking and sawing. They introduced an iterative solution approach using three interlinked models. The first model determines the optimal sawing pattern for each log, based on updated lumber values. The second model is a log bucking model which determines the optimal combination of cut for a given set of trees. This problem is solved using a dynamic programming approach with a knapsack problem. The main focus is a log allocation model which uses the cutting pattern optimizer and the bucking model together. It aims to distribute logs to different sawmills and choose optimal bucking and sawing strategies to maximize the benefits. They reported that the computational results show 26%-36% potential revenue gain due to the integration of bucking and sawing processes. A limitation of this study was that the dynamic aspects of planning are not considered; modeling was done over a single planning period.

Addressing this deficiency, Maness and Norton [22], in 2002, proposed a multi period production planning model for sawmill production. They integrated a linear programming model and a sawing simulator. In their model, they set target sales for products and assumed penalty costs and inventory costs for under production and over productions. The model maximizes revenue by considering four types of constraints including: supply,

production, market and inventory. The sawing simulator scans the logs by x-ray and uses this information to compute the recovery rate for each product according to input prices. These input prices are the shadow prices generated by the LP in each period and are dynamically updated. This model was tested on a large sawmill in British Columbia and it was shown that it responds to the changes in the market, by choosing different saw patterns and adjusting log consumption. Setup time associated with providing supply from different booms is not included in the modeling. Booms are "floating rafts of logs" for keeping the logs on the water used in British Columbia sawmills. Also, details of sawing, drying and finishing are ignored.

In a different way, Todoroki and Rönnqvist [35] suggested an optimization model that considers demand information to control production planning. In this study, the sawing of each log is optimized to maximize the volume or the value produced. This approach only takes into account the detailed sawing process and not issues such as drying and finishing. Although dynamic prices are considered, their proposed model is not multi-period.

Maturana, Pizani and [24], in 2010, compared two models for sawmill scheduling: a mathematical programming model and a heuristic model. They showed that their mathematical model performs better than the heuristic model in almost all aspects. It is notable that the cutting patterns are chosen to maximize volume, which does not necessarily mean they are maximizing revenue. They also ignored the setup time for change overs between running different raw materials. Further, the implemented model was run for a relatively small number of products and cutting patterns, 7 products and 6 cutting patterns, which is not realistic.

Randomness is investigated in sawmill production by Kazemi et al. In 2007, they [39] proposed a two-stage stochastic model for multi period, multi-product, production planning when yield uncertainty is assumed because of the non-homogenous characteristic of logs. Two years later Kazemi et al. [40] developed two robust optimization models for multi period, multi-product, production planning at the sawing unit of the sawmill. Again uncertainty in quality of the raw materials (logs) is considered. The purpose of this study is to find the production plans with robust demand satisfaction. Both of the above discussed studies were implemented on real scale prototype sawmills. Optitek, a log sawing simulator, is employed to generate lumber yields. One limitation of these studies is the small number of cutting patterns; they used 5 different cutting patterns, while in the work reported here more than 8000 cutting patterns are used. Also, a limited number of logs are processed to estimate the lumber yields.

In this thesis, a general multi-product, multi-period, sawmill scheduling problem is modeled and solved for a prototype real scale sawmill. In the reviewed literature typically few patterns and products are considered during the sawing process while we consider cases with 70 products and more than 6000 potential patterns. Setup time is another issue not covered in many sawmills scheduling models which we consider in our work. Adding setup time to the model increases the complexity of the problem, since integer variables are required, but this enables us to more realistically model the production process.

2.3 Rolling Horizon Method and Hierarchical Planning

The rolling horizon method and hierarchical planning structure are both examined in this thesis. In the rolling horizon methodology, before the first period, the model is solved for a certain number of future time periods, the planning horizon, using the current available information including inventory levels, and demand forecasts. However, only the current period's decisions are implemented in practice. At the start of the second period, the horizon is rolled one period forward and the production scheduling is updated since more accurate information about current yields and future demands has become available. Again, only current period decisions are implemented, and the process repeats. The frequency for updating the process and replanning does not necessarily happen every time period; it depends on the nature of the problem. Figure 3 demonstrates the rolling horizon method for three months of planning.

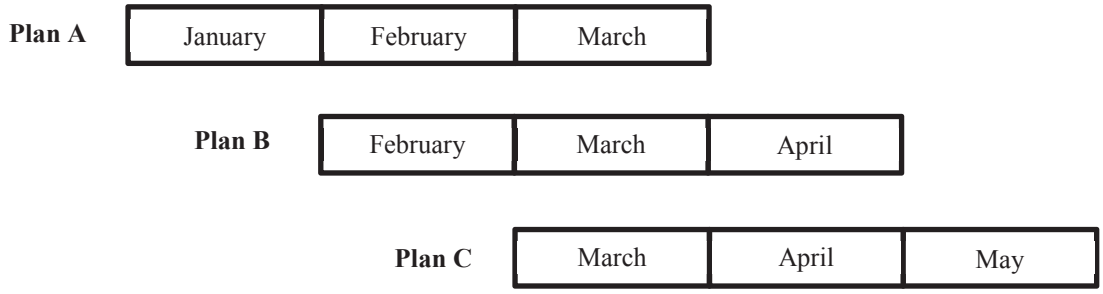


Figure 3- Rolling Planning Horizon

Rolling planning horizons are widely applied in dynamic production environments because of it is a practical and effective method [7].

Here, a brief literature review on this concept is provided. In 1977, Baker [2] examined the effectiveness of rolling schedules in production planning for an experimental study. He assessed how the length of forecast window, the future interval over which information is gathered, affects the rolling schedules for a single product in a

deterministic environment. His simulation results show that a main issue is judiciously choosing the forecast window. When seasonal effects are absent, the natural economic order quantity cycle is the most effective forecast window.

Five years later Carlson et al. [4] extended Baker's study. They used the same model to investigate the efficiency of extending the forecast window via forecasting. They found that extending the forecast window provides greater reduction in costs if the length of the forecast window is less than that of the natural economic order quantity cycle. They concluded that the forecast horizon should be at least as long as the natural cycle.

Most past studies like those mentioned have assumed a single product. Though, in the real world, production scheduling problems include multiple products. An example of a real world application with more than 200 products in the ice cream industry can be found in [14].

Kadipasaoglu and Sridharan [18] evaluated the efficiency of three strategies for decreasing nervousness in multi-level systems in uncertain demand environments. The three strategies are freezing the master production plan, using end-item safety stock, and lot-for-lot scheduling below level zero. Their results show that freezing the master production schedule is the best approach for reducing both instability and costs. This study compared alternative approaches in a multi-item, multi-level system with the assumption of an infinite production capacity.

In another study, by Xie et al. [38], a multi-item model with a single capacity constraint under demand uncertainty was examined. They provided a complete evaluation of the effects of freezing parameters, such as planning horizon, on the performance of rolling production. They concluded that a longer planning horizon decreases the costs but

produces unstable schedules. Based on their simulation results a trade-off exists between total costs and schedule instability according to the length of the frozen interval. They also found that it is not correct to generalize the result under deterministic demand to an uncertain demand environment.

In the forestry management area, Gunn [13] shows that under uncertain conditions, a deterministic model, using a rolling planning horizon for replanning, can result in a reasonable heuristic procedure for dealing with uncertainty.

Overall, in the real world, where everything is not deterministic, a rolling planning horizon responds better to the current situation as it uses more accurate data. Rolling planning horizon is implemented in the hierarchical planning structure in this study. In general there are two approaches for developing the production plan. The first approach is to consider the entire planning problem in a single level. This approach can result in huge models with large data requirements and difficult to interpret solutions. The alternative to single-level planning is multi-level planning in which the hierarchical structure of multiple levels is created. The decisions are made sequentially and affect each other. Figure 4 illustrates the hierarchical structure.

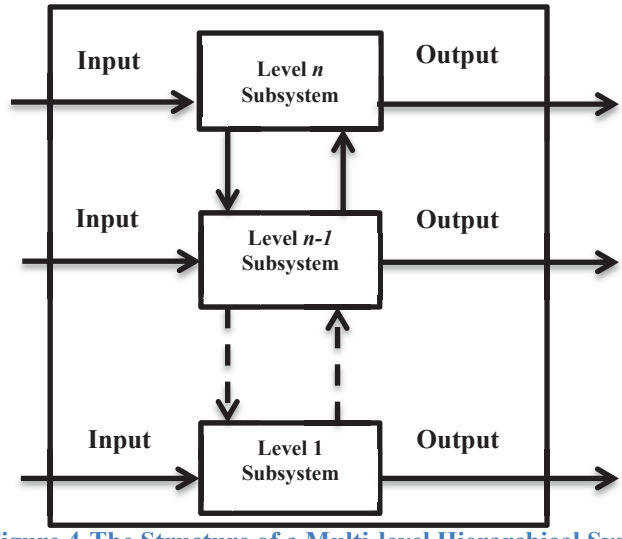


Figure 4-The Structure of a Multi-level Hierarchical System and the Data Used at each Level (after Mesarovic et al [26])

Gunn counts the main characteristics of the hierarchical planning as follows [23]:

1) The overall model is divided into different levels which makes the model easier and smaller; also increases the flexibility and practicality for use in a decision making environment.
2) A rolling planning horizon is can be implemented in hierarchical planning. Therefore, only the first period decisions are implemented, and after receiving feedback regarding the results of the first period decisions, new decisions are made.
3) Hierarchical planning is less risky in uncertain conditions. Since the model is broken down into multiple levels, more accurate data is available at the lower level allows more detailed decisions at lower levels.
4) Hierarchical planning is more compatible with organizational structure so it's easier to implement the model.

According to the above mentioned attributes of hierarchical planning, it is expected that the hierarchical structure is appropriate for sawmill planning in this study.

In the next chapter of this thesis the details of creating campaigns are explained.

Chapter 3

Creating Campaigns[1]

In the sawmilling process, one significant challenge is to estimate the production of lumber from 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 their sawing processes. This classification can be done according to their sizes, species, or any other desirable characteristics that can form "log classes". Log sawing is an automated process where logs are scanned and sawn in an attempt to maximize the 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 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 log class given a price list which yields a specific setup outputs through implementation of a set of optimized cutting patterns.

This chapter first details the methodology, and its assumptions, for creating 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

[1] The work behind this chapter was joint work with Pegah Sohrabi. See also her thesis [30].

discussion of important parameters of the algorithm. Finally, an overview of further research regarding algorithm application is presented.

3.1 Methodology

This framework consists of four stages. The first stage provides an algorithm to create the set of possible cutting patterns. These patterns are then sorted and 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 optimizer will perform to break down the logs and transform them to lumber, and thus generate product output proportions for the campaign. Figure 5 demonstrates the methodology used for generating campaigns. The programming language used for the algorithms is Python 2.6.6. Details of the Python code are provided in appendix B. Python language is open-source and thus our work is readily available to everyone.

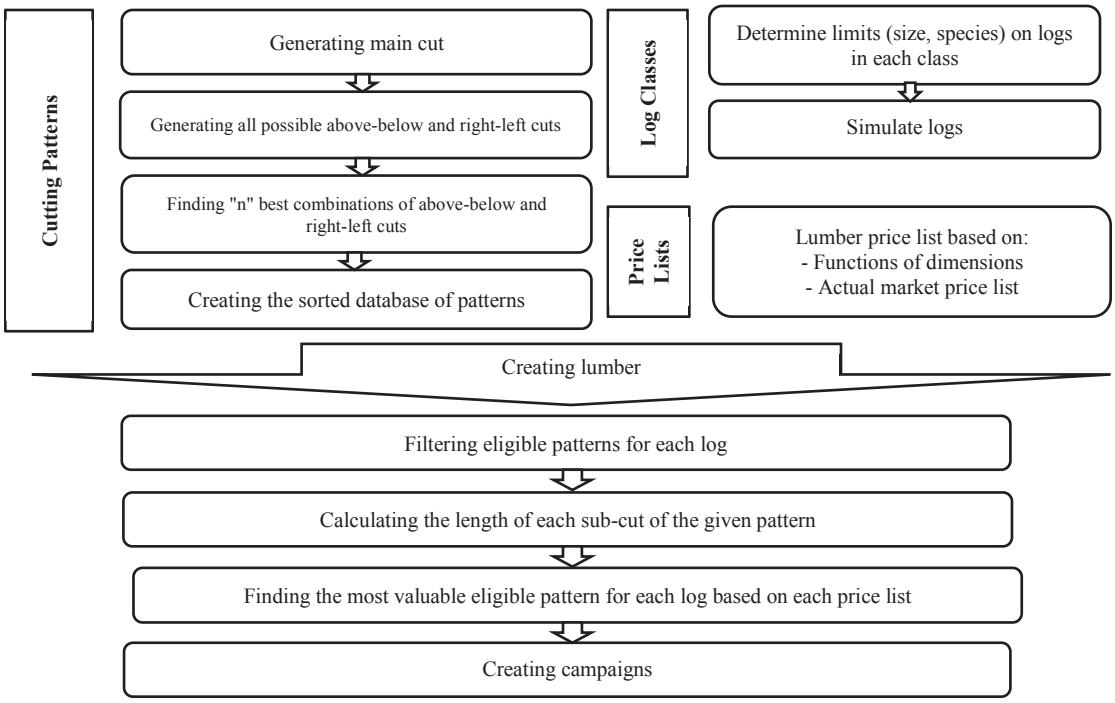


Figure 5- Methodology

The assumptions behind our methodology are as followings:

- 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 6. Although this assumption does not perfectly reflect the real world situation, it provides a reasonable estimation of actual logs.

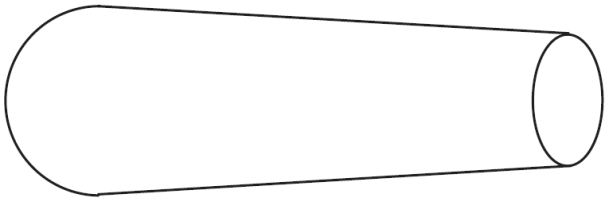


Figure 6- A Typical Log

- 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; then the cant is broken down into dimensional lumber through a gang saw. If the edges are capable of producing lumber, they can go to a re-sawing process, presumably in an edger. Figure 7 illustrates the assumed sawing process.

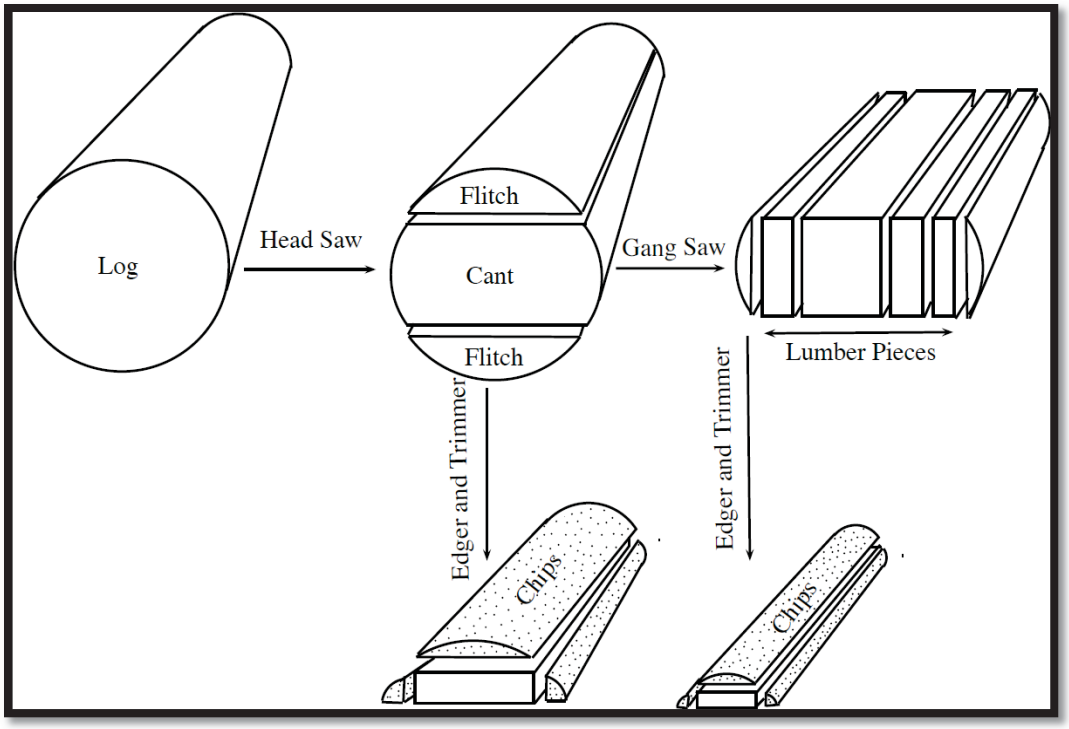


Figure 7- Assumed Sawing Process

- Wane on some pieces of limber 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 up to 25% of each side can have wane. Figure 8 illustrates a piece of lumber with wane.

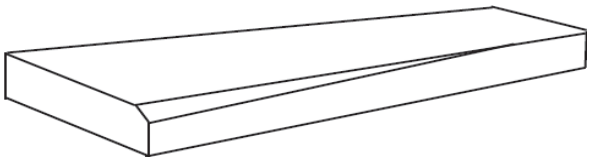


Figure 8- Lumber with Wane

- 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 to each other as well as right-left cuts. This assumption is shown in figure 9.

- The main cut has vertical sub-cuts. However, above-below and right-left cuts can have either vertical or horizontal.

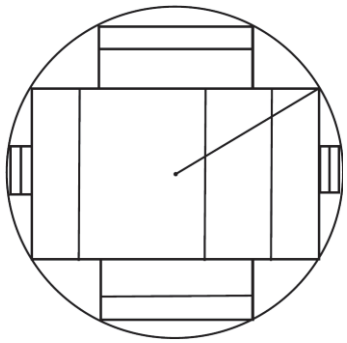


Figure 9- A Typical Cutting Pattern

- 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 "2 x 6 x 10" product are "1.5 x 5.5 x 10". To get this product, the target dimensions are set as "1.66 x 5.875 x 10". 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", within each dimension of the lumber.
- 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 saw and edger. 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 (8125 patterns) 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 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 cut 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)

- $wRatioMax_j$: set of maximum ratios 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})

$$W_j^{max} = T_j^m * wRatioMax_j \quad (1)$$

Step 2: For each $X_{i,j}$ find the maximum value according to the calculated W_j^{max}

$$X_{i,j} \leq \frac{W_j^{max}}{W_i} \quad (2)$$

Step 3: Find all possible combinations of sub-cuts ($X_{i,j}$) and calculate W_j^m:

$$W_j^m = (\sum_i X_{i,j} * (W_i + Kerf)) - kerf \quad (3)$$

$$\text{where } W_j^m \leq W_j^{max} \quad (4)$$

The function used for this step is called "combos" in the Python code (Appendix B).

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

For $j = 2$, $T_2^m = 4$ and $wRatioMax_2 = 2$ thus according to equation (1), $W_2^{max} = 8$

Using formula (2), we have $X_{1,2} \leq 4$ *and* $X_{2,2} \leq 2$.

Based on equation (3) W_2^m can have all the following values presented in Table 2:

Table 2- An Example of Widths Combinations

$X_{1,2}$	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

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 10. This procedure ensures that the larger (generally most valuable) cuts are at the center of the log, therefore trying to avoid any wane 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.

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 10. This radius represents the circle in which a pattern can be located.

$$R = \sqrt{(\frac{T_j^m}{2})^2 + (\frac{W_j^m}{2})^2} \quad (5)$$

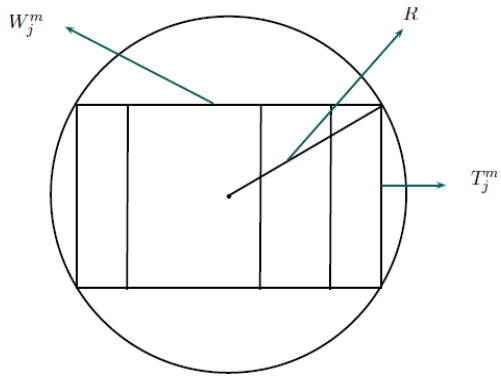


Figure 10- Main Cut

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 11, 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).

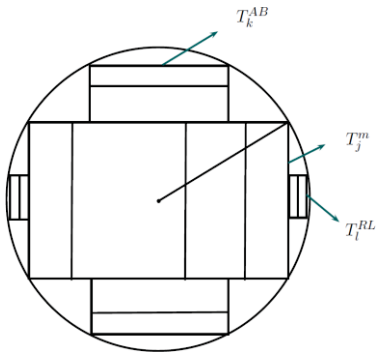


Figure 11- A Pattern with Above-below and Right-left Cuts

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 W_i and thickness 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 shows only the details of categories 1 and 2. The rest are presented in appendix A.

Table 3- Formulas for Above-below and Right-left Cuts

Vertical above-below sub-cuts	Horizontal above-below sub-cuts
For each T_k^{AB} where: $$T_k^{AB} \leq R - (\frac{T_j^m}{2} + \text{kerf}) \quad (6)$$ We calculate all the combinations of possible widths and findW_k^{AB}. $$W_k^{AB} = (\sum_i Y_{i,k} * (W_i + \text{Kerf})) - \text{kerf} \quad (7)$$ Where: $$W_k^{AB} \leq 2 * \sqrt{R^2 - (T_k^{AB} + \frac{T_j^m}{2} + \text{kerf})^2} \quad (8)$$	For each T_k^{AB} where: $$T_k^{AB} \leq W_j^m \quad (9)$$ We calculate all the combinations of possible widths and findW_k^{UD}. $$W_k^{AB} = (\sum_i Y_{i,k} * (W_i + \text{Kerf})) - \text{kerf} \quad (10)$$ Where: $$W_k^{AB} \leq \sqrt{R^2 - \left(\frac{T_k^{AB}}{2}\right)^2} - \frac{T_j^m}{2} - kerf \quad (11)$$

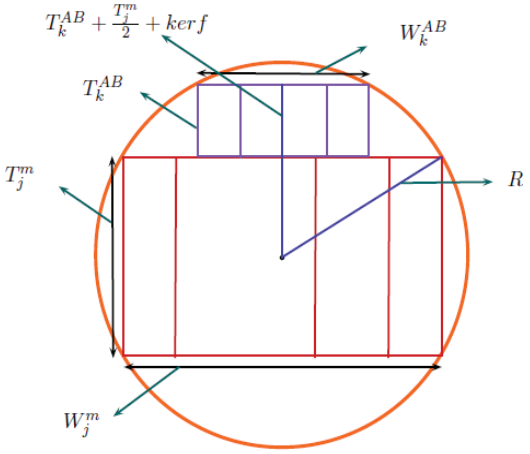
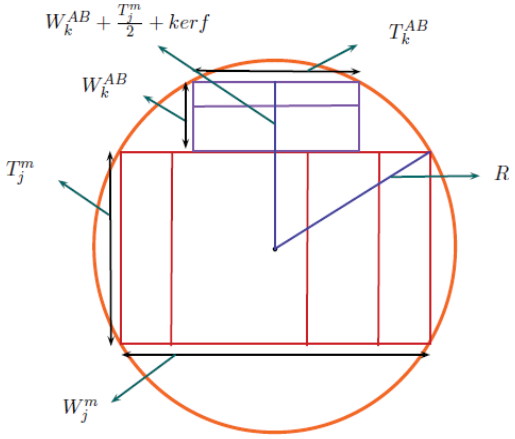

	

Figure 12- Vertical Above-below Sub-cuts	**Figure 13- Horizontal Above-below Sub-cuts**

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 6 and 9). In the second step all the combinations of possible widths are generated by considering sawing kerf (formulas 7 and 10). Meanwhile, the final width of each cut should be fitted in the allowable remaining area (formulas 8 and 11).

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 12. 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 cuts areas should be multiplied by two because of the symmetry assumption. The denominator stands for area of the log.

$$Area\ Percent\ Yield = \ Main\ cut\ area + \ 2 * above\ below\ area + 2 * right\ left\ area =$$

$$((W_j^m - ((\sum_i X_{ij}) - 1) * kerf) * T_j^m + 2 * (W_k^{AB} - ((\sum_i Y_{ik}) - 1) * kerf) * T_k^{AB} + 2 *$$

$$(W_l^{RL} - ((\sum_i Z_{il}) - 1) * kerf) * T_l^{RL})/(\pi * R^2))*100 \quad (12)$$

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. Notes that "n" is a user defined

variable and its value can be set in the Python code (Appendix B). A discussion how to choose "n" value is provided in section 3.3.1.

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:

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

Where:

R: Pattern radius

$\mathrm{T_j^m}$: Thickness of the main cut

$[\mathrm{X_{1j}}, \mathrm{X_{2j}}, \mathrm{X_{3j}}, \dots]$: A list for number of each sub-cut in the main cant with thickness $\mathrm{T_j^m}$ (in ascending order of widths $W_1, W_2, W_3, \dots$)

$\mathrm{T_k^{AB}}$: Thickness of the above-below cuts

$[\mathrm{Y_{1k}}, \mathrm{Y_{2k}}, \mathrm{Y_{3k}}, \dots]$: A list for number of each sub-cut in the above-below cuts with thickness $\mathrm{T_k^{AB}}$(in ascending order of widths $W_1, W_2, W_3, \dots$)

$\mathrm{T_l^{RL}}$: Thickness of the right-left cuts

$[\mathrm{Z_{1l}}, \mathrm{Z_{2l}}, Z_{3l}, \dots]$: A list for number of each sub-cut in the right-left cuts with thickness $\mathrm{T_l^{RL}}$(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], "Horizontal UD", "Vertical RL", ["P", 5087]] demonstrates that pattern number 5087 has the radius of $8.422(\text{in})$. The cant (main cut) thickness is $11.875(in)$ and this includes 3 sub-cuts with width $0.866\,(\text{in})$ and 5 sub-cuts with width$1.66\,(\text{in})$. Each of the above-below edges consists of two sub-cuts with thickness $3.75(\text{in})$ and width W_1 which are horizontaly cut. In each of right-left edges there are two vertical sub-cuts with thickness $2.75(\text{in})$ and width $0.866\,(\text{in})$. 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 the three basic factors i) small end radius, ii)taper and iii)length. Large radius (of large end) is calculated based on three previous factors (equation 13).

In our approach we generate log classes based on the following notation:

- $Class$: set including classes of loges {1,…,N}, each class involved 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 $Class$, $q \in Class$
- 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

$$R_q^l = R_q^s + Taper * L_q \quad (13)$$

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 derived 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 will discuss how it was used in section 3.3.1. 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 by considering the logs' taper.

Various log classes can be created according to different policies that a sawmill uses for a log sorting. 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. In other words, management can influence the output of certain log classes by deliberately setting the appropriate price list.

We have used two types of price lists: The first is based on actual market prices at a certain point in time that we got from the lumber price estimate guide [19] (table 4). This is the viewpoint that prices are real indicators of lumber value. If lumber can be sold into an infinite market with no requirement 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 view, a variety of price sets gives a variety of 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 14). 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.

$$Lumber\ value(\$) = Lumber\ volume(ft^3) * unit\ price(\frac{\$}{ft^3}) \quad (14)$$

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). This can be done through fitting a price function of available pieces to estimate

the value of missing lumber pieces. More details are available in section 3.2.1. 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 [22].

Table 4- Market Price for Doug Fir Lumber [19]

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

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. As shown in figure 3 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 Calculating the value of all eligible patterns implemented on each log based on each price list, 3.1.4.4 Finding the most valuable pattern for each log in each price list and 3.1.4.5 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: $R_S \leq R \leq R_L$ as shown in figure 14). 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 wanes and usually loss in the sub-cuts of the sides and even in the main cut.

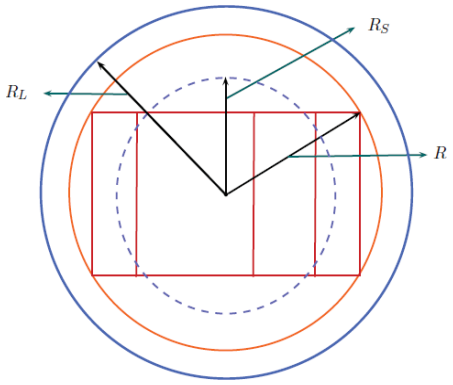


Figure 14- Eligible Pattern

3.1.4.2 Calculating the Length of each Sub-cut of the Given Pattern

Because the logs are tapered, it 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 is ascending order of widths continues till the largest sub-cuts is 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.

- $rWaneUD$: 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
- $\mathrm{Seq_j^m}$: Set of sub-cuts' widths of the main cut with $\mathrm{X_{i,j}}$ number of $\mathrm{W_i}$ for each $\mathrm{T_j^m}$
    - 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 thicknessT_j^m:

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

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

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

Step 3: Calculate rWaneUD, rWaneSide and rWane

$$\text{rWaneSide} = \sqrt{\left((\text{W}_\text{R} * \text{B}_\text{R}) + (\text{W}_\text{L} * \text{B}_\text{L})\right)^2 + \left(\left(\frac{1-wanePrSide}{2}\right) * \text{T}_\text{j}^\text{m}\right)^2} \quad (15)$$

$$\text{rWaneUD} = \sqrt{\left((\text{W}_\text{R} * \text{B}_\text{R}) + (\text{W}_\text{L} * \text{B}_\text{L}) - (wanePrUD * W_{min})\right)^2 + (\text{T}_\text{j}^\text{m}/2)^2} \quad (16)$$

As calculated above there can be two radii for wanes ($rWaneSide$, $rWaneUD$) 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 17, figure 15).

$$rWane = Maximum(rWaneUD, rWaneSide) \quad (17)$$

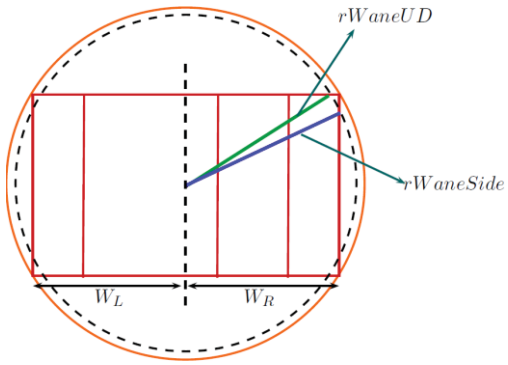


Figure 15- rWaneUD and rWaneSide

Step 4: CalculateLwane:

$$\text{Lwane} = \left\lfloor \text{Min}\left\{\text{L}, \text{L} * \left(1 - \frac{\text{rWane}-\text{R}_\text{S}}{\text{R}_\text{L}-\text{R}_\text{S}}\right)\right\}/2 \right\rfloor * 2 \quad (18)$$

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

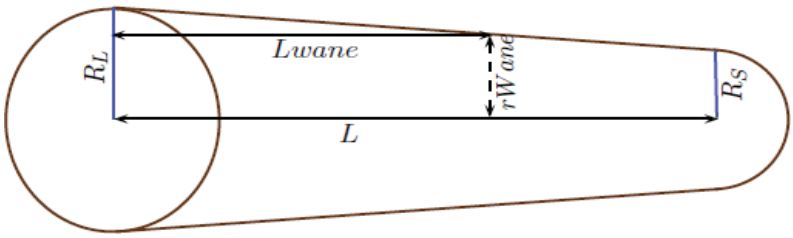


Figure 16- Touching Point and the Length of Lumber

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

Otherwise go to Step 6.

Step 6: If $Lwane = L_k$, 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}. 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

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 19 shows the value of the best pattern for all given price lists. As previously mentioned, X_{ijk}, Y_{ijk} and Z_{ijk} are number of lumber pieces with the width W_i, thickness T_j and length L_k, in the main cut, above-below cut and right-left cut, respectively. These data are all generated and stored in the database.

$$MaxValue^u = Maximum_{all\ patterns}\{\sum_i \sum_j \sum_k P_{ijk}^u \left(X_{ijk} + 2 * (Y_{ijk} + Z_{ijk})\right)\} for\ all\ u \quad (19)$$

Where:

$MaxValue^u$: The maximum over the values of all eligible patterns on each log based on price list u.

P_{ijk}^u: Price of lumber with the width W_i, thickness T_j and length L_k for price list u.

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 is equal to the number of logs in the class. The resulted 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). 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 17.

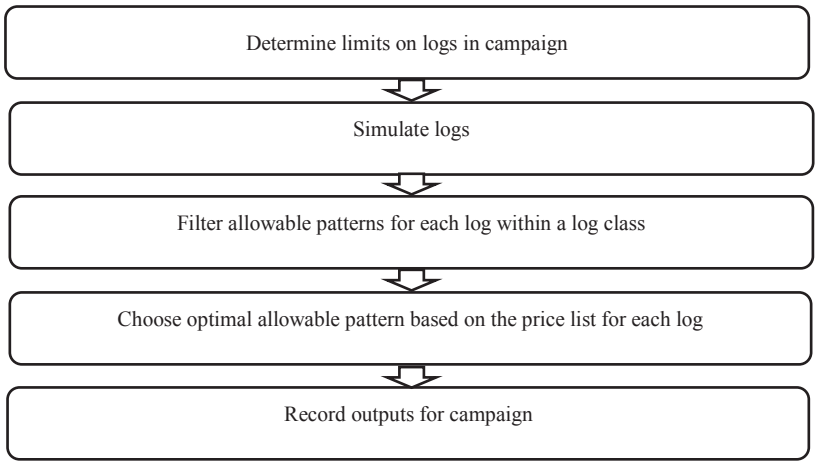


Figure 17- Algorithm for Generating Campaigns

$$Volume\ percent\ yield = \frac{\text{Total lumber volume}}{\text{Total volume of the logs}} * 100 \quad (20)$$

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 Procedure

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 5:

Table 5- Nominal Lumber Dimensions

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

Nominal, target and actual data for lumber dimensions according to Freeman and Son Ltd [16] reports are shown in table 6.

Table 6- Nominal, Target and Actual Data

Nominal (in)	Target (in)	Actual (in)
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

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 cut which decreases the chance of resawing 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 equals to two 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 scanner 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 were for the small end radius (with average 2.5(in)) and length as follows:

- $R_1^s \sim Uniform[2,3]$
- $L_1 = \begin{cases} P(8 \le L_1 < 10) = 0.4 \\ P(10 \le L_1 < 12) = 0.4 \\ P(12 \le L_1 < 14) = 0.2 \end{cases}$

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 [3]. 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 by Minitab® is Lognormal with the mean of 1.198 and standard deviation of 0.323.

- $R_2^s \sim lognormal(1.198, 0.323)$

The results of fitting the lognormal distribution function to log class 2 data by using Minitab® are shown in figure 18. As can be seen from the probability plot, the fit is reasonable over the mid range of the radii.

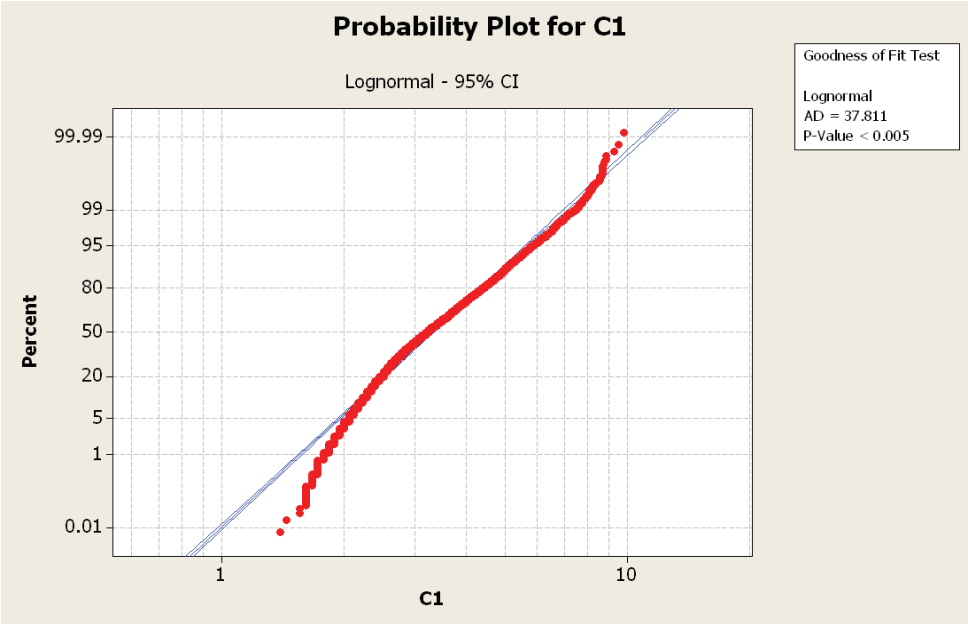


Figure 18- Minitab Probability Plot for Small End Radius

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 shown in figures 19 & 20.

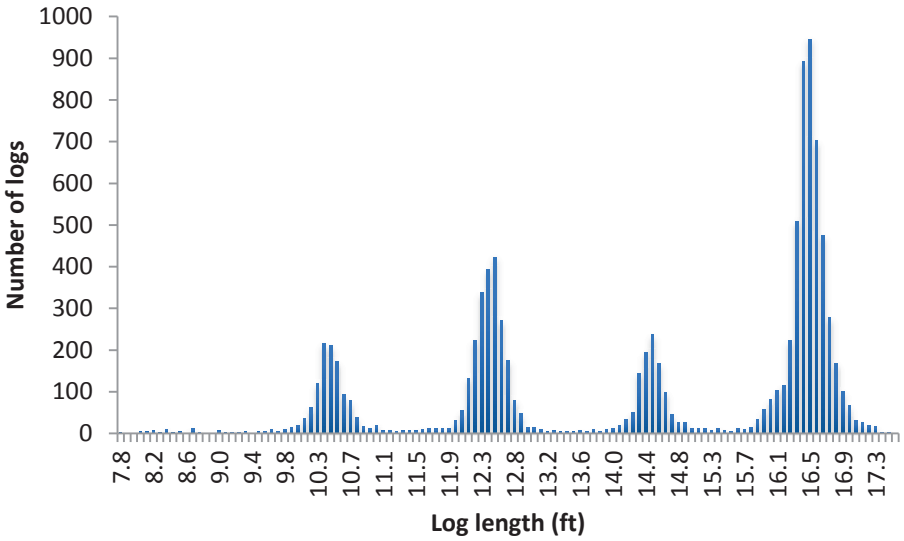


Figure 19- Real Data for Log Class 2

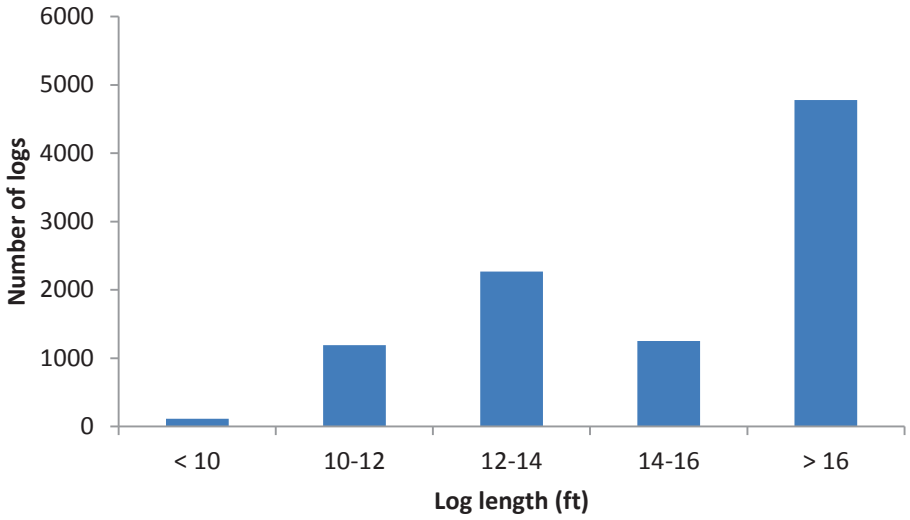


Figure 20- Fitted Uniform Distribution for Log Class 2

The following shows the mentioned distribution functions.

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

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 ends diameters usually increases. Thus, we assume a log radius can change 0.05 to 0.2 inches per each feet of length. To estimate the larger end radius of a log, by considering its taper (between 5% and 20% of the log's length), we assumed the relationship between smaller radius and larger radius for both classes as follows:

- $R_q^L = F(R_q^S, L_q)(in) = R_q^S + Uniform[0.05, 0.2](\frac{in}{ft}) * L_q, q = 1,2$ (21)

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

We combine small and large log classes (200,000 logs) together and sort their logs according to specific log lengths into five categories. This creates five 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 output of each log was calculated once, we do not need to recalculate them. Available logs are only sorted in specific lengths. Our classification includes 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 table 5. Due to lack of data for some products such as lumber with width 1(in), a function is fitted to the available ones to give the estimate of relative prices for products without data. The fit function is computed as described in equation 22.

$$Fit\ function\ (W^{nom}, T^{nom}, L^{nom}) = a + b * W^{nom} + c * T^{nom} + d * L^{nom} + e * (W^{nom} * T^{nom}) + f * (W^{nom} * T^{nom} * L^{nom}) \quad (22)$$

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 21 demonstrates the difference between actual and fitted prices for all the available products. 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 23:

$$Fit\ function(W^{nom}, T^{nom}, L^{nom}) = 2.498 + 1.047 * W^{nom} + 0.198 * T^{nom} + 0.049 * L^{nom} - 0.0357 * (W^{nom} * T^{nom}) - 0.0000278 * (W^{nom} * T^{nom} * L^{nom}) \quad (23)$$

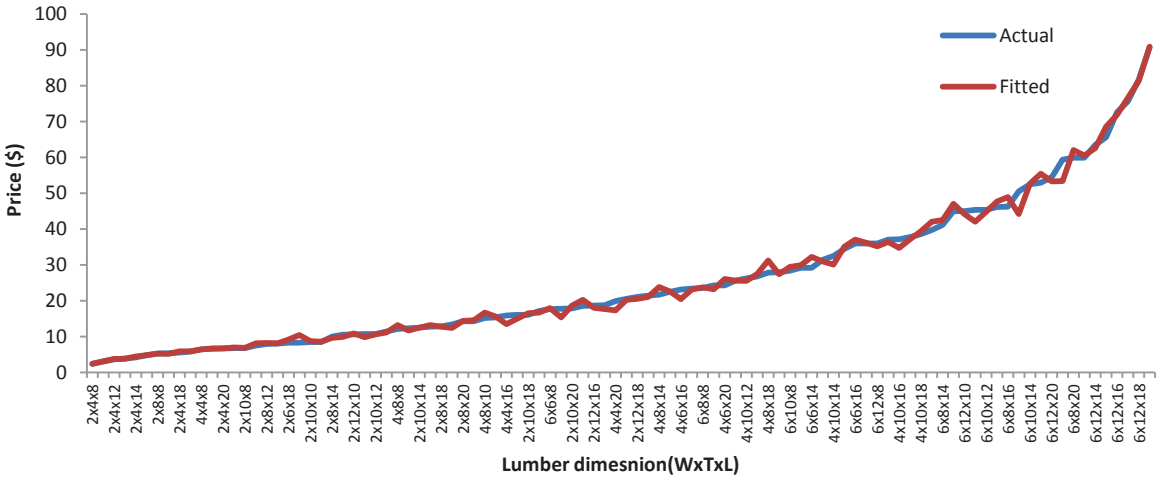


Figure 21- Price Fit Function

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

The other price lists are based on emphasizing certain lumber nominal dimensions. We created 19 price lists as follows:

- *Price list 2*: Since the unit price is the same for all products this price list emphasizes on obtaining the most lumber volume out of the log (equation 24).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}(ft) * L^{nom}(ft) * 1(\frac{\$}{ft^3}) \quad (24)$$

- *Price list 3:* This price list aims to focus on lumber with greater widths (equation 25).

$$Lumber\ value(\$) = W^{nom}\sqrt{W^{nom}}(ft)^{1.5} * T^{nom}(ft) * L^{nom}(ft) * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (25)$$

- *Price list 4:* This price list emphasizing lumber with greater thicknesses (equation 26).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}\sqrt{T^{nom}}(ft)^{1.5} * L^{nom}(ft) * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (26)$$

- *Price list 5*: This price list attempts to focus on lumber with greater lengths (equation 27).

$$Lumber\ value(\$) = W^{nom}(ft) * T^{nom}(ft) * L^{nom}\sqrt{L^{nom}}(ft)^{1.5} * 1\left(\frac{\$}{ft^{3.5}}\right) \quad (27)$$

The remaining price lists(6 to 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 pricelists 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 generating the same lumber outputs (we discuss this in section 3.3.3). 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 list maximizes the lumber output volume. The second example corresponds to log classes 3 to 7 when price list 1 is used. Price list one is the actual market price list.

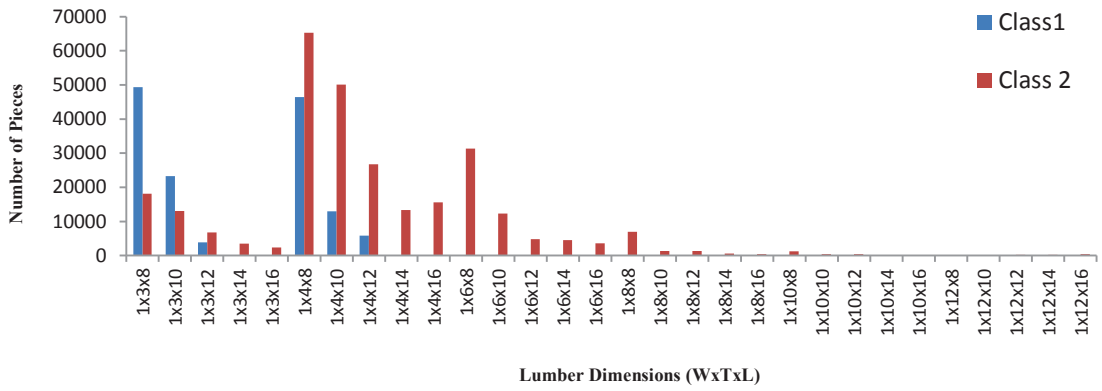


Figure 22- Price List 2, Log Classes 1 and 2 (Part 1)

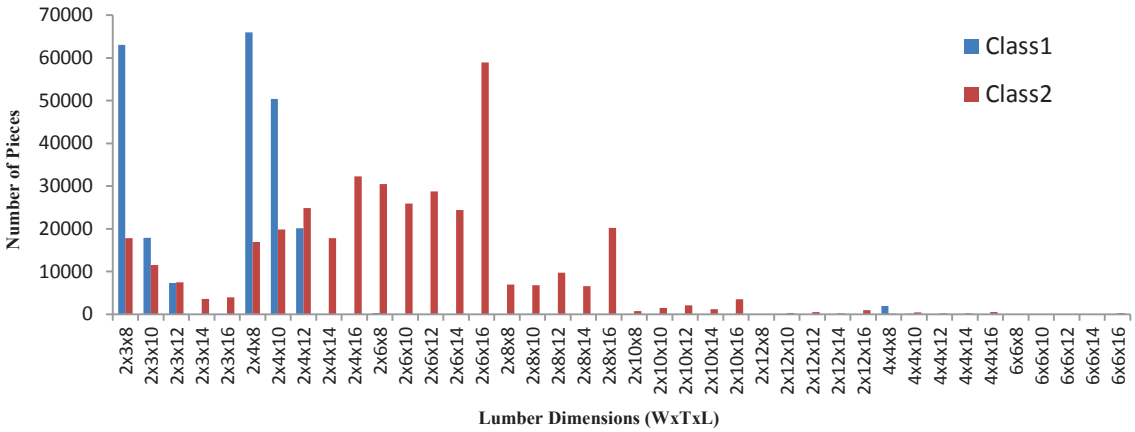


Figure 23- Price List 2, Log Classes 1-2(Part2)

As can be seen in figure 22 and 23, 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 of 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 7 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.

Table 7- Volume Percent Yields in Comparison Case 1

	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

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 its logs' 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 24 demonstrates the difference in outputs of the campaigns in the second comparison. As log classes are created by sorting logs in specifics lengths, it can be seen that log class 3 (blue) , in which logs are with the length less than 10, produces only the products with length 8(ft). This behavior 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 25. 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 26 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 27), when price lists focus on specific lengths, we see most products with length 8(ft) are usually created by applying price list 16(blue). This mechanism happens for the remaining price lists as well. Consequently, Price lists 17(red), 18(green), 1 9(purple) and 20(orange) mostly create lumber pieces with length 10(ft), 12(ft), 14(ft) and 16(ft) respectively.

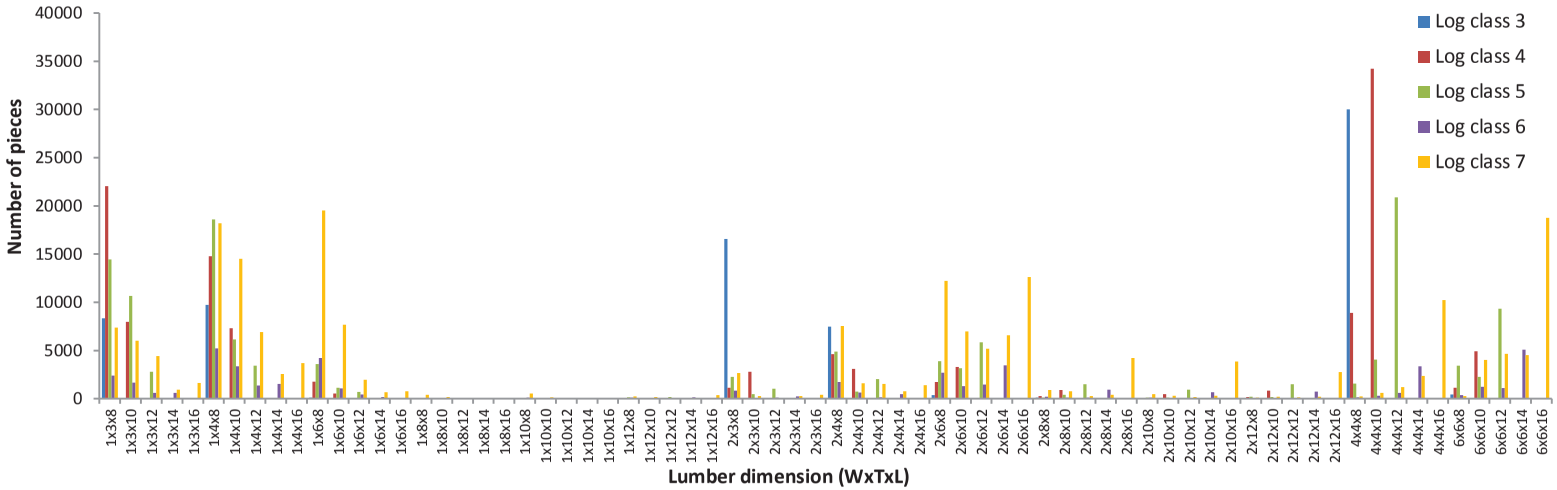


Figure 24- Price List 1, Log Classes 3-7

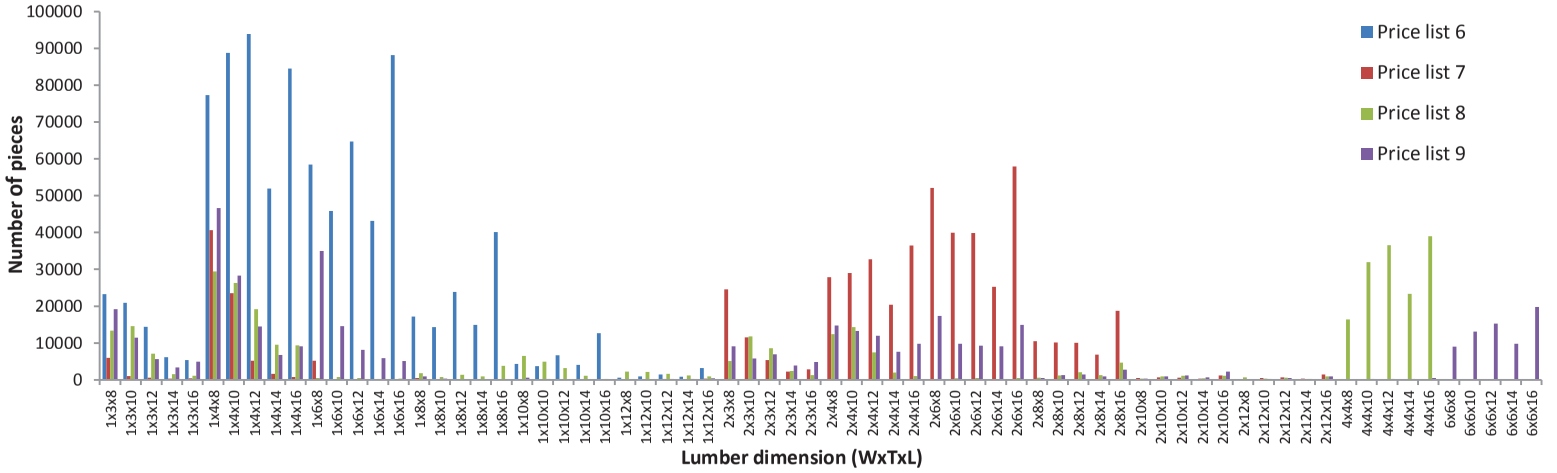


Figure 25- Log Class 2, Price Lists 6-9

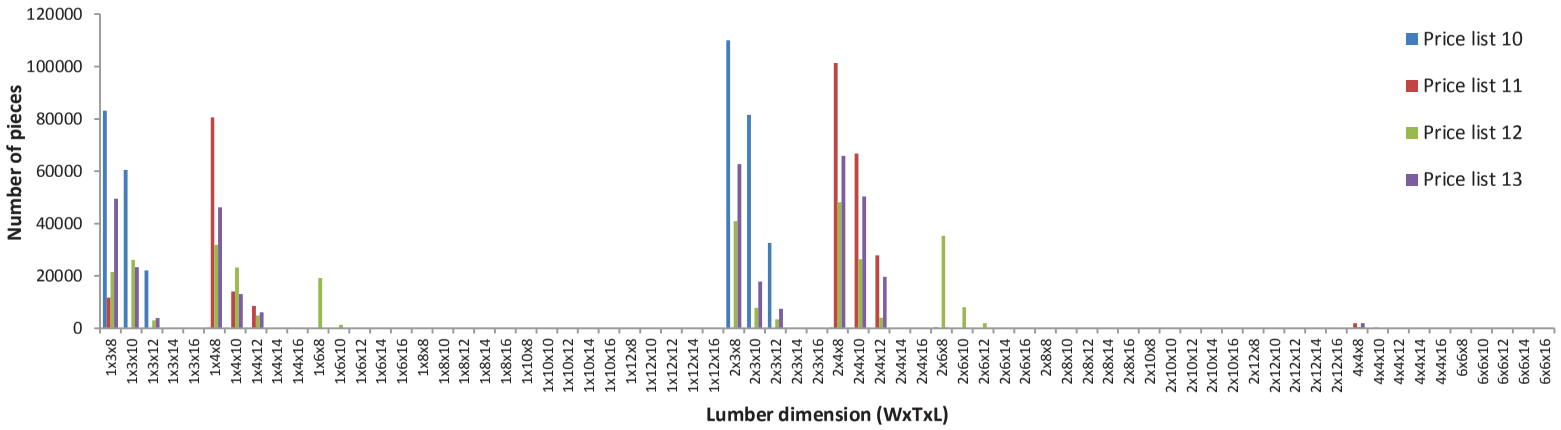


Figure 26- Log Class 1, Price Lists 10-13

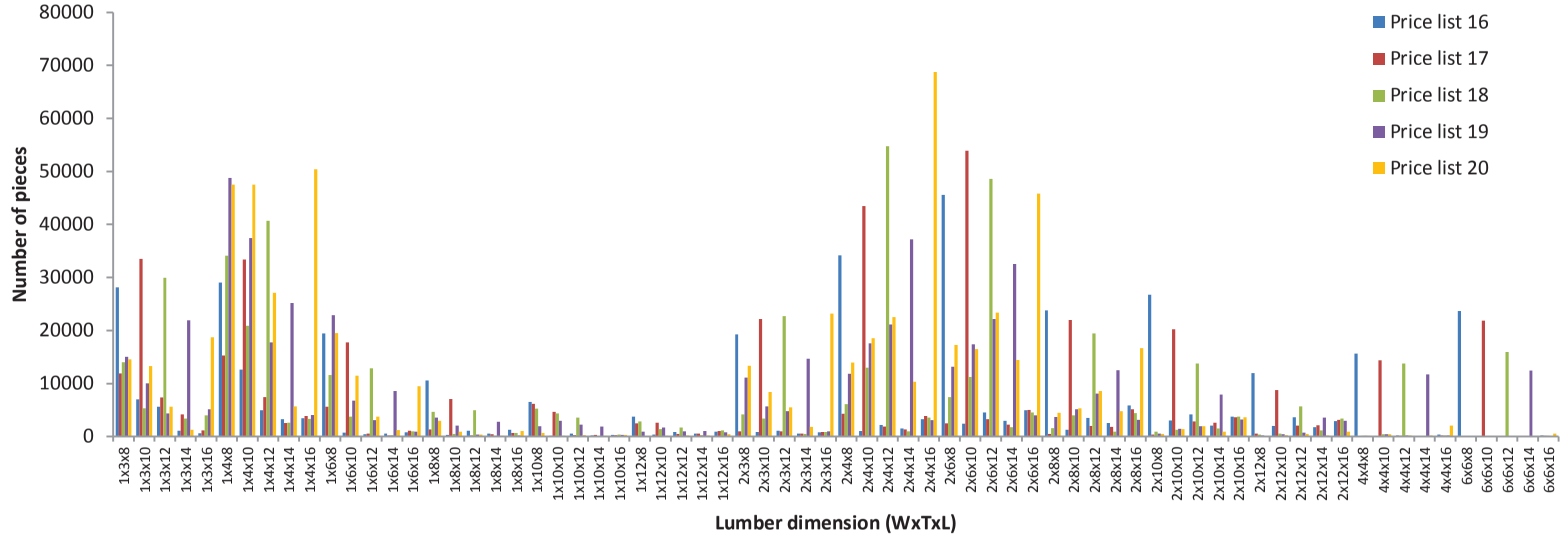


Figure 27- Log Class 2, Price Lists 16-20

3.3.3 Discussion

The previous example discussed in this chapter demonstrated the features of the 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 class cannot create large lumber, these price lists that emphasizing large pieces are not effective. For example price list 19 emphasizes pieces with length 14 (ft), these products cannot produced from the small log class because logs in this class are shorter than 14 (ft). The products that can be produced are evaluated on volume, leading to the same results of these campaigns. Additionally, those 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 makes it easy to rapidly generate a number of potential 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, these created campaigns are scheduled by using the mathematical model.

Chapter 4

Campaign Scheduling for Sawmill

A sawmill is a highly automated piece of machinery. As such, it is impossible to personally make individual decisions about each log. What actually happens is that the sawmill has a series of scanners and optimizers that decide how to break the log down into various products so as to maximize the value of the log in response to a set of product prices. This suggests viewing the sawmill as a hierarchical optimization problem. At the upper level, the mill management is trying to optimize sales within demand constraints while keeping inventory levels feasible, preferably low. They do this by providing various price lists to the lower level sawmill problem, which aims at optimizing the value of its production with respect to these prices.

In this chapter we start with the hierarchical perspective and then proceed to see how we might use this in a Mixed Integer Programming framework. Section 4.1 explains the methodology of this work. Section 4.2 discusses the mathematical model in details. Section 4.3 shows the implementation of the model, results and sensitivity analysis of the model. Finally section 4.4 provides a brief discussion for this chapter and demonstrates future research of this approach.

4.1 Methodology

Fundamental to the discussion here is the assumption that we know how the mill responds to different price lists. As previously discussed in chapter 3, if we feed a price list to the sawmill optimizer on assumed class of logs, we get a set of lumber outputs. We found these outputs and called them campaigns. That is the lower level model which is a price optimization response (mill operation level in figure 28).

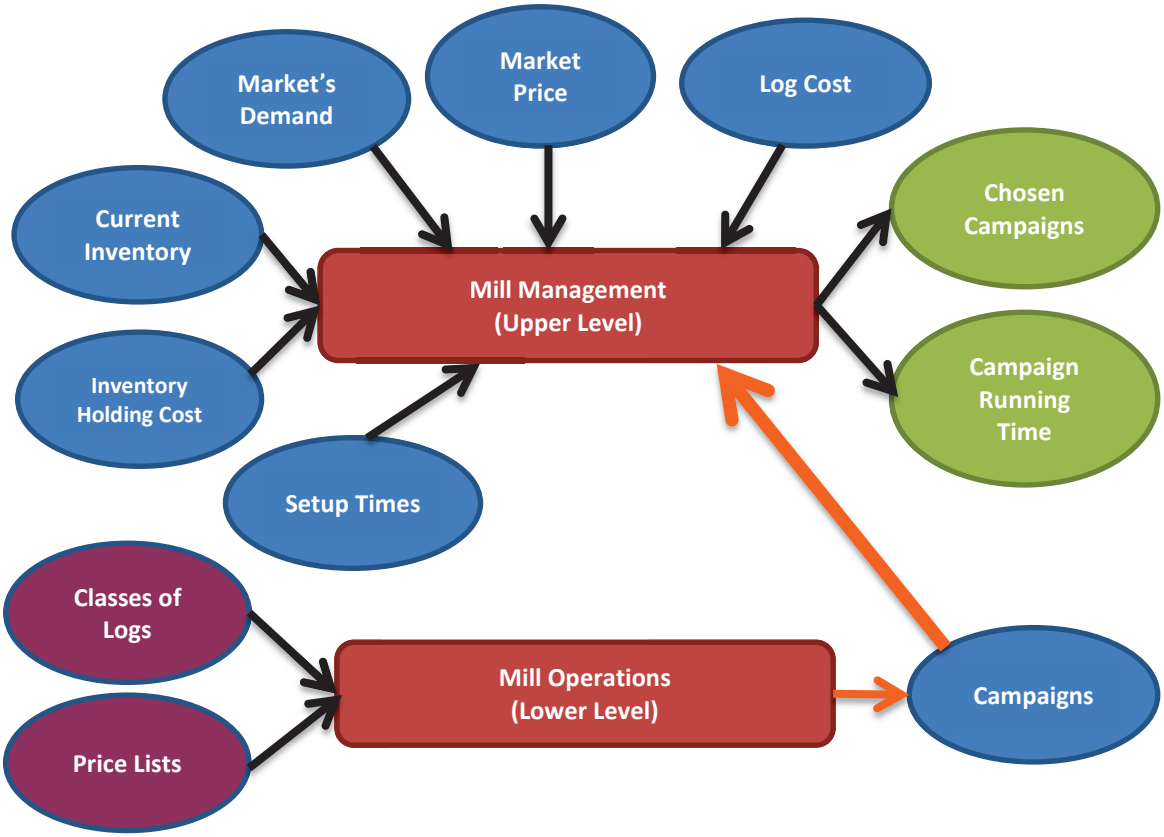


Figure 28- Hierarchical Structure

Figure 28 illustrates the hierarchical structure of the assumed system. As it can be seen, Campaigns, the result of the lower level, are the inputs of the upper level. Although these

two levels are linked to each other through campaigns, they can be solved separately. The lower level, which was discussed in some detail in the previous chapter, provides us different campaigns. At the upper level, mill management applies these campaigns to maximize the total benefit of the sawmill.

4.2 Mathematical Model

The proposed mathematical model tries to find the best possible schedule for the potential campaigns in order to maximize the total benefits of the sawmill. The revenue from selling the product minus totals costs including supply cost, inventory holding cost and penalty cost for out of range inventory provide the sawmill benefits. The sawmill has a limited production capacity, expressed in total time available. The question is how to schedule the potential campaigns in order to maximize the total net revenue of the sawmill within a defined time horizon under the capacity constraints. We have multiple product types in term of size, and also multiple product species from different tree species. We have multiple log classes and campaigns (discussed in chapter 3) to be scheduled over the planning horizon. Note that each campaign belongs to only one log class while within each log class we can have several campaigns. We assumed different market price levels for selling lumber pieces and there is a maximum sale for each price level of the market.

The changeover from one campaign to another will involve a setup. Setup can involve a major setup such as a switch between two different classes of logs or a minor setup such as changing the price lists. The setup for each log class ($\boldsymbol{STC_c}$) takes a longer time because some sawmill adjustment may be needed; also the source of logs must be

changed to the logs in the required log class. The basic campaign setup time ($\boldsymbol{ST_k}$) is shorter because the optimizer just changes the price vectors to switch to the desirable campaign. We need to use binary variables in this problem to account for setup times. More details are provided in section 4.3. The mathematical description of the model is as follows:

Index sets

$i \in I$: Set of product types (i_c is the index to the chip product)

$s \in S$: Set of product species

$k \in K$: Set of campaigns

$c \in C$: Set of classes

$t \in T$: Set of time periods

$(k, c) \in KC$: Set of campaigns and classes

$l \in L$: Set of market levels

Data

$\boldsymbol{v_{ist}^{l}}$: Selling price for product i from species s in period t at market level l (\$/ft^3)

$\boldsymbol{N_{ist}^{l}}$: Amount of product i from species s which can be sold for price v_{ist}^{l} in period t (ft^3**)**

$\boldsymbol{h_{ist}}$: Inventory holding cost for product i from species s in period t (\$/ft^3)

$\boldsymbol{O_{isk}}$: Output of product i in species s from campaign k per unit volume input run of campaign k (percentage)

$\boldsymbol{V_{tk}}$: The log volume input rate of campaign k in period t (ft^3 per period)

$\boldsymbol{I_{ist}^{Min}}$: The minimum allowable inventory of product i from species s at the end of period t (ft^3)

$\boldsymbol{S_{ist}^{Max}}$: The maximum allowable sales of product i from species s in period t (ft^3)

$\boldsymbol{S_{ist}^{Min}}$: The minimum allowable sales of product i from species s in period t (ft^3)

$\boldsymbol{LC_k}$: Cost of the logs per tonne for campaign k ($/tonne)

$\boldsymbol{ST_k}$: Setup time for running campaign k (period)

$\boldsymbol{STC_c}$: Setup time for running class c (period)

$\boldsymbol{\alpha}$ **:** A constant number for transforming log volume to log weight (tonne/ft^3)

γ: Penalty cost for violating minimum inventory constraint ($/$ft^3$)

ICAP: The maximum storage capacity of the mill for holding lumber (ft^3)

Variables

$\boldsymbol{X_{ist}^{l}}$: Amount of product i from species s sold in period t for price v_{ist}^{l}(ft^3)

$\boldsymbol{I_{ist}}$: Inventory of product i from species s at the end of period t (I_{is0} is data on initial inventory) (ft^3)

$\boldsymbol{\varepsilon_{ist}}$: Amount by which the inventory balance constraints are violated below (ft^3)

$\boldsymbol{\lambda_{kt}}$: Proportion of time that campaign k is run during period t (period)

$\boldsymbol{P_{ist}}$: Production of the product i from species s in period t (ft^3)

$\boldsymbol{toChips_{ist}}$**:** Amount of product i, species s which goes to the chips in period t (ft^3)

$$\boldsymbol{Y_{kt}}: \begin{cases} 1 & if\ campaign\ k\ is\ run\ at\ period\ t \\ 0 & if\ campaign\ k\ is\ not\ run\ at\ period\ t \end{cases}$$

$$\boldsymbol{Z_{ct}}: \begin{cases} 1 & if\ class\ c\ is\ run\ at\ period\ t \\ 0 & if\ class\ c\ is\ not\ run\ at\ period\ t \end{cases}$$

Objective function:

$$Max \sum_i \sum_s \sum_l \sum_t ((X_{ist}^l * v_{ist}^l) - (h_{ist} * I_{ist}) - (\gamma * \varepsilon_{ist})) - (\sum_k \sum_t ((\lambda_{kt} * V_{kt})) * \boldsymbol{\alpha} * LC_k)$$

Subject to:

1) $I_{ist} = I_{is(t-1)} + P_{ist} - (\sum_l X_{ist}^l) - toChips_{ist}$ *for each , s, t and* $i \neq i_c$

2) $I_{cst} = I_{cs(t-1)} + P_{cst} - (\sum_l X_{cst}^l) + \sum_i toChips_{ist}$ *for each i, s and t*

3) $P_{ist} = \sum_k \lambda_{kt}$*$(O_{isk} * V_{tk})$ *for each i , s and t*

4) $\sum_{k,(k,c)\epsilon KC}(\lambda_{kt} + Y_{kt} * ST_k) + \sum_c(Z_{ct} * STC_c) \leq 1$ *for each t*

5) $S_{ist}^{Min} \leq \sum_l X_{ist}^l \leq S_{ist}^{Max}$ *for each i , s and t*

6) $I_{ist}^{Min} - \boldsymbol{\varepsilon_{ist}} \leq I_{ist}$ *for each i , s and t*

7) $X_{ist}^l \leq N_{ist}^l$ *for each i , s, l and t*

8) $\lambda_{kt} \leq Y_{kt}$ *for each k and t*

9) $Y_{kt} \leq Z_{ct}$ *for each k , c and t, where (k,c)*ϵ*KC*

10) $\sum_i \sum_s I_{ist} \leq ICAP$ *for each i , s and t*

11) $I_{i,s,0} \leq I_{i,s,end}$ *for each i, s (end is the last period)*

12) $\sum_k Y_{kt} \geq Z_{ct}$ *for each k , c and t, where (k,c)*ϵ*KC*

13) $\sum_c Z_{ct} \geq 1$ *for each t*

14) $\lambda_{kt} \geq Y_{kt} * ST_{kt}$ *for each k and t*

15) $\sum_k \lambda_{kt} \geq STC_c * Z_{ct}$ *for each c and t, where sum is over k: (k,c)*ϵ*KC*

16) $0 \leq P_{ist}$, $0 \leq X_{ist}^{l}$

17) $0 \leq \lambda_{kt} \leq 1$

The objective function, which shows the total benefit, includes four terms. The first term demonstrates the resulting income for sold lumber at different market level prices for all periods. The second term is holding cost which is subtracted from the total benefit. The third term shows the penalty cost for the lost inventory. Finally the last term represents the raw material supply expenses. The form given here is due to the fact that, although sawing typically transforms volumes, logs are often purchased by weight. Thus the logs volume for each campaign in each period is calculated and then converted to weight and consequently the total cost.

We have several sets of constraints in this problem. Constraints 1) are inventory balance equations which states that the inventory of each lumber piece at the end of each period equals to the inventory of that lumber at the end of the previous period plus the produced lumber during this period minus total sale of this specific lumber in this period minus the amount of that product transformed to chips.

The second set of constraints 2) is inventory balance equation for the chips (Product number 71). These state that amount of the chips in each period equals to amount of chips in the previous period, plus produced chips during this period minus total sale of chips in this period plus sum of the other products transformed to chips in current period.

Constraint set 3) shows that production of each product in each species is related to three factors: i) running time proportion of each campaign in that period, ii) percentage output of each product and species for each campaign and iii) log input rate for each campaign. The multiplication of these factors results in the produced lumber from each species in each period.

Constraints 4) requires that total time of running campaigns plus setup times used for campaigns and classes in each period cannot exceed the period time.

Constraint set 5) limits the total sales in each period to the lower bound and upper bound.

Constraint set 6) represents the desired lower bound for inventory of each product. Although inventory can go below that limit there is a penalty cost for that.

Constraint set 7) defines the market limit on sales at the price level l for each p, s in period t.

Constraint set 8) states that if the campaign is not chosen then the proportion of time for that campaign is zero, in other words you have to setup a campaign to run it.

Constraint set 9) allows the campaign to be run when the class which includes that campaign is selected.

Constraints set 10) states that total inventory in each period should be less than or equal to the storage capacity of the sawmill.

Constraints set 11) forces that total inventory at the end of the running time to be greater than or equal to total inventory at the starting point. This removes the possibility of "free" production. In an actual situation we could impose ending inventory constraints based on corporate considerations.

Constraints sets 12, 13, 14 and 15 are added to the model to make the integer solving process faster by adding some cutting planes to the feasible region. The differences between the result with them and without them will be discussed later.

Constraints set 12) states that if the class is run, at least on campaign in that class should be run. In the other words you never setup a class when you are not running any campaign in that class

Constraints set 13) forces at least one class in each period to be run.

Constraints set 14) sets the lower bound for running each campaign which is equal to its setup time. If the campaign is selected, the running time should be at least equal to the time consumed for setting up that campaign.

Constraint set 15) repeats the idea of the previous set of constraints for the class. It requires that, when a sawmill chooses a class, the total running time for campaigns in that class should be at least equal to time consumed to setup up that class.

Constraint set 16) requires the productions and sold lumber to both be non-negative.

Finally last constraints set shows that the running time of each campaign in each period is non-negative and less than the period time so time proportion is less than one.

4.3 Implementation of the Model

To demonstrate the performance of the proposed mathematical model for scheduling different campaigns in a sawmill, a realistic example had been solved by this model. Section 4.3.1 explains the example and illustrates the details of data used for it. Section 4.3.2 compares the solution times with and without additional constraints. Section 4.3.3 provides the resulting solution for the mentioned problem and finally, Section 4.3.4 examines the model for different scenarios.

In implementing this model we have made the following definitions and assumptions. These can be changed by others who use this approach.

- The output products are assumed to be described by nominal dimensions. Thus output yields are in terms of nominal volume per ft^3 solid wood input.
- The chosen output units are all cubic feet (ft^3).
- No down time considered for sawmill.
- Demand is known for the first four weeks and forecasted for the remaining weeks.

4.3.1 Data

We did not have access to the real data such as capacity, demand or production rate for sawmills. We thus have tried to construct reasonable data sets to test our concepts. We describe these here.

- Production Capacity

A sawmill with an annual production rate 80 M fbm (80,000,000) board feet lumber nominal output is assumed.

- Product types

As developed in the previous chapter, 70 different types of lumber in terms of dimensions are produced by the campaigns. Since every sawmill produces chips, this is also added to the list of products as product with index of c. As a result the problem solved by considering in total 71 types of outputs.

- Lumber species

Different species result in different product types with different processing times, qualities and market prices. In this example, two different species of logs are assumed to be used by the defined campaigns. We refer to these as spruce (species 1) and pine (species 2) for purposes of illustration.

- Number of periods for running the model

In this example we are trying to solve a tactical production planning problem. So the assumed horizon for this example is a season of 13 weeks and the periods defined as weeks.

- Campaigns and Classes

In the previous section 126 campaigns are generated. As explained before these campaigns are created through 7 different log sorts (classes) and various price vectors. All 126 campaigns are considered for the first species which is considered spruce. For the second species (pine) the first two classes (small and large) are repeated to generate 36 campaigns for the second species. Although it would be possible to repeat all the generated campaigns for the second species, we only assumed 36 campaigns; because more campaigns increase the size of the model and in this study we just want to show that

it is possible to have more than one species. In total, for these two species, 9 classes and 162 campaigns are assumed. Table 8 shows the classes and campaigns in each class. Log classes 1-7 are the spruce log classes and log classes 8-9 are the pine.

Table 8- Campaigns from Log Classes and Price Lists

Class / Price List	1	2	3	4	5	6	7	8	9
Actual	1	17	37	53	70	88	107	127	143
Volume	2	18	38	54	71	89	108	128	144
W^1.5	3	19	39	55	72	90	109	129	145
T^1.5	4	20	40	56	73	91	110	130	146
L^1.5	5	21	41	57	74	92	111	131	147
W=1	6	22	42	58	75	93	112	132	148
W=2	7	23	43	59	76	94	113	133	149
W=4	8	24	44	60	77	95	114	134	150
W=6	9	25	45	61	78	96	115	135	151
T=3	10	26	46	62	79	97	116	136	152
T=4	11	27	47	63	80	98	117	137	153
T=6	12	28	48	64	81	99	118	138	154
T=8	13	29	49	65	82	100	119	139	155
T=10	-	30	50	66	83	101	120	-	156
T=12	-	31	51	67	84	102	121	-	157
L=8	14	32	52	68	85	103	122	140	158
L=10	15	33	-	69	86	104	123	141	159
L=12	16	34	-		87	105	124	142	160
L=14	-	35	-			106	125	-	161
L=16	-	36	-				126	-	162

The price lists are those discussed in chapter 3. The first price list is based on the actual market price list. The second price list aims at maximizing volume yield by considering same unit price for all products. The third, fourth and fifth price lists emphasize the larger width, thickness and length respectively. The remaining price lists reward specific width, thickness or length as explained in chapter 3. The output fractions of these campaigns are provided in appendix D.

- Lumber Market levels

To show the effects of the market on product prices, various lumber market levels are defined for each species. In both species, 3 different market levels (high, medium and low) are considered (figure 29). For the first species, spruce, high level are sold for full price, medium level for 80% of the full price and the low level 50% of the full price. The second species (pine) is more sensitive to the market levels in which the high level goes for full price, medium level for 50% of the full price and the low level just for 20% of the full price. The maximum sale for each species in each level will be explained later.

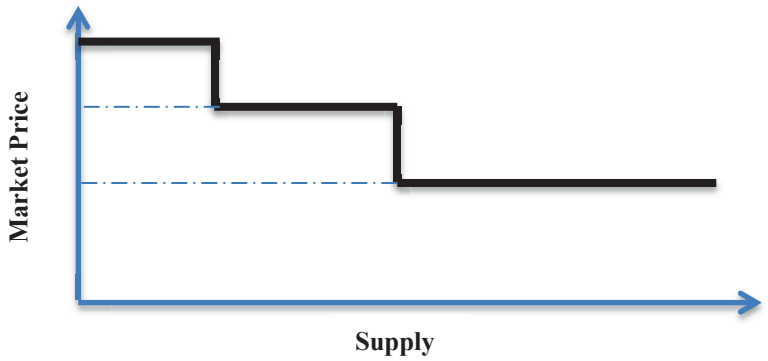


Figure 29- Market Levels

Table 9- Input Product Data for the Model

Product	Dimensions (WxTxL) (in*in*ft)	Average Demand $ft^3/week$	Unit Price ($/$ft^3$)	Holding Cost ($/$ft^3$)	Product	Dimensions (WxTxL) (in*in*ft)	Average Demand $ft^3/week$	Unit Price ($/$ft^3$)	Holding Cost ($/$ft^3$)
1	1x3x8	3449.388	4.422433	0.040492	36	2x4x8	10924.27	5.48813	0.045616
2	1x3x10	2073.339	4.519666	0.04096	37	2x4x10	7921.274	5.585085	0.046082
3	1x3x12	799.2673	4.616899	0.041427	38	2x4x12	4771.97	5.68204	0.046548
4	1x3x14	399.5905	4.714132	0.041895	39	2x4x14	2605.536	5.778995	0.047014
5	1x3x16	235.4637	4.811365	0.042362	40	2x4x16	2553.478	5.87595	0.047481
6	1x4x8	6046.227	4.58502	0.041274	41	2x6x8	3492.103	5.74146	0.046834
7	1x4x10	3616.337	4.682198	0.041741	42	2x6x10	3322.807	5.838193	0.047299
8	1x4x12	2400.034	4.779375	0.042209	43	2x6x12	3927.914	5.934925	0.047764
9	1x4x14	1232.175	4.876553	0.042676	44	2x6x14	4290.089	6.031658	0.048229
10	1x4x16	955.7446	4.97373	0.043143	45	2x6x16	4598.379	6.12839	0.048694
11	1x6x8	1843.663	4.910195	0.042837	46	2x8x8	1212.677	5.99479	0.048052
12	1x6x10	981.8016	5.007261	0.043304	47	2x8x10	1413.427	6.0913	0.048516
13	1x6x12	785.901	5.104328	0.043771	48	2x8x12	1887.214	6.18781	0.04898
14	1x6x14	714.9148	5.201394	0.044237	49	2x8x14	2096.009	6.284321	0.049444
15	1x6x16	637.3305	5.29846	0.044704	50	2x8x16	2415.516	6.380831	0.049908
16	1x8x8	601.5389	5.23537	0.044401	51	2x10x8	731.84	6.24812	0.04927
17	1x8x10	322.471	5.332325	0.044867	52	2x10x10	793.945	6.344408	0.049733
18	1x8x12	338.2702	5.42928	0.045333	53	2x10x12	947.0752	6.440696	0.050196
19	1x8x14	313.6838	5.526235	0.045799	54	2x10x14	890.4678	6.536983	0.050659
20	1x8x16	361.5875	5.62319	0.046265	55	2x10x16	966.5798	6.633271	0.051121
21	1x10x8	381.007	5.560545	0.045964	56	2x12x8	367.9265	6.50145	0.050488
22	1x10x10	213.0774	5.657389	0.04643	57	2x12x10	447.0633	6.597516	0.05095
23	1x10x12	184.7001	5.754233	0.046895	58	2x12x12	601.3857	6.693581	0.051411
24	1x10x14	125.8033	5.851077	0.047361	59	2x12x14	595.679	6.789646	0.051873
25	1x10x16	130.8041	5.94792	0.047827	60	2x12x16	697.56	6.885711	0.052335
26	1x12x8	197.2974	5.88572	0.047528	61	4x4x8	4353.936	7.29435	0.0543
27	1x12x10	149.2529	5.982453	0.047993	62	4x4x10	3301.418	7.39086	0.054764
28	1x12x12	142.5346	6.079185	0.048458	63	4x4x12	2167.333	7.48737	0.055228
29	1x12x14	152.6021	6.175918	0.048923	64	4x4x14	1155.354	7.583881	0.055692
30	1x12x16	137.6048	6.27265	0.049388	65	4x4x16	988.7899	7.680391	0.056156
31	2x3x8	6879.644	5.361465	0.045007	66	6x6x8	1244.962	9.066521	0.06282
32	2x3x10	2873.813	5.458531	0.045474	67	6x6x10	1756.269	9.161918	0.063278
33	2x3x12	1760.281	5.555598	0.04594	68	6x6x12	2022.72	9.257316	0.063737
34	2x3x14	859.7346	5.652664	0.046407	69	6x6x14	1942.296	9.352714	0.064196
35	2x3x16	811.5066	5.74973	0.046874	70	6x6x16	1691.476	9.448111	0.064654

- Selling price ($\boldsymbol{v_{ist}^{l}}$)

For the first species, the previous price fitted function per volume unit (cubic feet) is assigned to the products (details of the fit function are explained in chapter 3). Table 9

shows the unit price for all lumber dimensions of the first species. The second species price is considered 20% above the first one. These are the full prices for the high level market. The medium and low levels prices relations are those described above in defining the market levels. Although the prices in the model are dynamic and can be varied in each period, for this particular run of the model they are kept constant from period to period.

There were no data for chips price, so for spruce chips we assumed 3 dollars per cubic feet and for the second species, pine, 2 dollars per cubic feet. These amounts are less than the assumed unit prices for each of the products in table 9. Thus, the model should try to avoid producing chips as much as possible.

- Sales upper bound for each level($\boldsymbol{N_{ist}^{l}}$)

Given a lack of real demand data for each product, we tried to create a demand for products that had a chance of being satisfied from the campaigns. The average output fractions of each product from all campaigns were calculated and then normalized in the way that the total demand equals to the nominal production rate. The assumed average demand ($\boldsymbol{AvgD_i}$) created by this calculation, is shown in table 9. Then 80% of the demand assigned for species one and 20% for the second species. The assumption here is that we know the exact demand for each product in the next 4 periods and the average demand forecasted for the rest of the horizon (figure 30). To generate demands for the first 4 periods, a random number distributed uniformly between 0.75 and 1.25 is multiplied by the average demand to give the actual demand for each product.

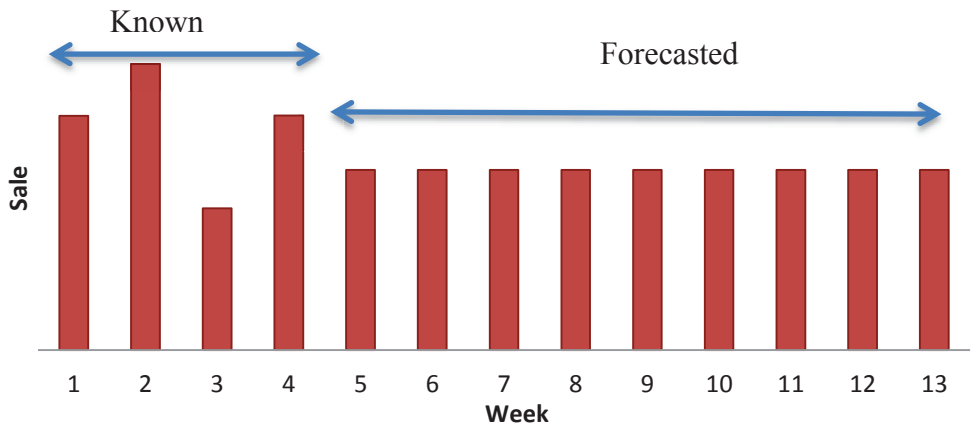


Figure 30- Planning Horizon

After calculating the actual demand for the first four periods and average demand for remaining periods based on the predefined market levels, $\boldsymbol{N_{ist}^{l}}$ is calculated. The spruce market levels were computed as 50% of the demand for the high level, 30% for the medium level and 120% for the low level.

$$N_{i1t}^{1} = AvgD_i * 0.8 * 0.5$$

$$N_{i1t}^{2} = AvgD_i * 0.8 * 0.3$$

$$N_{i1t}^{3} = AvgD_i * 0.8 * 1.2$$

The demand for the second species (pine) occurs in five periods, period 5 to 9, and for the rest of the weeks is zero. This is an attempt to represent a situation where pine cannot always be sold. The pine market levels were computed as 20% of the demand for the high level, 30% for the medium level and 150% for the low level. The following formula show how these demand levels for periods 5 to 9 (t=5, 6, 7, 8 and 9) are calculated.

$$N_{i2t}^{1} = AvgD_i * 0.2 * 0.2 *^{*} 13/5$$

$$N_{i2t}^{2} = AvgD_i * 0.2 * 0.3 * 13/5$$

$$N_{i2t}^{3} = AvgD_i * 0.2 * 1.5 * 13/5$$

- Inventory holding cost ($\boldsymbol{h_{ist}}$)

The holding cost consists of two parts, the variable cost which depends on the value of the lumber and it's considered 25% of that value (the high level value) annually. The fix cost for is then considered 1$ per year. So the holding cost calculated by the following formula:

$$\boldsymbol{h_{ist}} = (v_{ist}^{1} * 0.25 * 1/52) + 1/52 \quad for\ all\ i, s\ and\ t$$

Table 9 gives the holding cost for the first species in terms of dollar per cubic feet per week.

- Output fractions of each campaign ($\boldsymbol{O_{isk}}$)

In the previous section by providing different price vectors to various log classes many campaigns are generated. Then the nominal output fraction of each product calculated for all of the created campaigns. The nominal output fraction is the percentage of product *i* and species *s* produced per unit volume (ft^3) input of campaign *k*. For the chips calculation the rest of the output from each campaign as chips output. In this model we have not attempted to account for sawdust, shavings and/or bark as outputs although this is easy to do. Note that the first 7 classes here are only producing species one and just the last two classes producing species two. The details of these campaign outputs are provided in appendix D.

$$\boldsymbol{O_{c,s,k}} = 1 - TargetYield_k$$

- The maximum amount of input rate of campaign ($\boldsymbol{V_{tk}}$)

The maximum rate of input run has been treated as the same for all campaigns in all periods. To calculate this number, nominal weekly output is divided by the 55.54% (considered as an average nominal yield for all campaigns) and then multiplied by 1.20 to cover the time we lose for setup times.

$$weekly\ nominal\ demand\left(\frac{ft^3}{week}\right) = Annual\ nominal\ output\ \left(\frac{fbm}{year}\right) * 52\left(\frac{week}{year}\right) * \frac{1}{12}\ (\frac{ft^3}{fbm})$$

$$weekly\ nominal\ demand\left(\frac{ft^3}{week}\right) = 80{,}000{,}000\ \left(\frac{fbm}{year}\right) * \frac{1}{52}\left(\frac{week}{year}\right) * \frac{1}{12}\left(\frac{ft^3}{fbm}\right) = \mathbf{128205}$$

$$\boldsymbol{V_{tk}} = \frac{weekly\ nominal\ demand}{Average\ nominal\ yield} * 1.2 = 276987\ (ft\text{^}3) for\ all\ t\ and\ k$$

$$\boldsymbol{V_{tk}} = \frac{128205}{0.5554} * 1.2 = 276987\ (ft\text{^}3) for\ all\ t\ and\ k$$

- The minimum inventory ($\boldsymbol{I_{ist}^{Min}}$)

The minimum inventory is set to be zero for all products in all periods. Although the minimum inventory is zero, the violation in inventory is allowed in the constraints and inventory can be dropped below zero by $\boldsymbol{\varepsilon_{ist}}$ to ensure that the model will be feasible.

$$\boldsymbol{I_{ist}^{Min} = 0} \qquad for\ all\ i, s\ and\ t$$

- The maximum and minimum allowable sales ($\boldsymbol{S_{ist}^{Min}}$, $\boldsymbol{S_{ist}^{Max}}$)

There is no limitation for minimum sale in this problem, so the assumed minimum sales are zero for all products. The maximum sale is set as twice of the average demand for each product.

$$\boldsymbol{S_{ist}^{Min} = 0}$$

$$\boldsymbol{S_{ist}^{Max} = 2 * AvgD_i}\ \ for\ all\ i, s\ and\ t$$

- Log cost ($\boldsymbol{LC_k}$)

The supply cost considered in this specific example is just for buying logs landed at the mill. Logs in larger classes are treated as more expensive than smaller ones. These prices are in the right order for NS mills but we have not attempted to get actual values. Table 10 illustrates the buying cost per tonne for each class that we have used.

Table 10- Log Cost

Class	1	2	3	4	5	6	7	8	9
Log Cost ($/ton)	70	80	70	72	74	76	78	60	70

- Setup times ($\boldsymbol{ST_k}$, $\boldsymbol{STC_c}$)

Two types of setup times considered as below:

$$ST_k = 1/240$$

$$STC_c = 1/80$$

As each period has been considered as a week of 5 days, and each day includes 8 hours thus totally each week lasts 40 hours. The above setup times for each campaign and class correspond to 10 and 30 minutes respectively.

- A constant number for transforming logs volume to purchased weight ($\boldsymbol{\alpha}$)

The only part of supply cost considered in this example is log cost. At the mill, logs are purchased based on their weights. $\boldsymbol{\alpha}$ converts ft^3 consumption to tonnes. A factor widely used in NS is 1.167 m^3/tonne for softwood.

$$\boldsymbol{\alpha} = Log\ density\left(\frac{tonnes}{m^3}\right) * \ converting\ factor\left(\frac{m^3}{ft^3}\right)$$

$$= \frac{1}{1.167} * \frac{1}{35.31} = 0.0242646\left(\frac{tonnes}{ft^3}\right)$$

- Penalty cost for out of the range inventory **(γ)**

To discourage negative inventory levels, a high penalty cost is used for below zero inventory. Twenty dollar per cubic feet is the assumed penalty cost. This is greater than selling price of any product; therefore the model will never sell product which is not in stock.

$$\gamma = 20\ (\frac{\$}{ft^3})$$

- The maximum capacity of the mill for holding lumber (***ICAP)***

In this example assumed sawmill is capable to hold the inventory up to ten time of the weekly nominal lumber demand. Weekly nominal lumber demand is 128205 (ft^3)

$$capacity(ft^3) = 10 * 128205(ft^3)$$

- Initial inventory ($\boldsymbol{I_{is0}}$)

To generate the initial inventory for the sawmill, a random number from a uniform distribution between 0.5 and 1.5 for each product is multiplied by average demand to create the initial inventory for species one. For the second species as there is no sale in the first four periods, so the initial inventory is considered as zero.

4.3.2 Comparing Models with and without Additional Constraints

As explained previously, a few additional sets of constraints (12, 13, 14 and 15) are not logically necessary. The model will have the same integer optimal solution without these constraints. However, they have been added to the model to strengthen the linear programming relaxation and thus improve solvability (see Williams [37]). The following graphs show the converging time versus gap percentage for two models. The gap is the difference between the best feasible integer solution found by the model and the current upper bound during the branch and bound process.

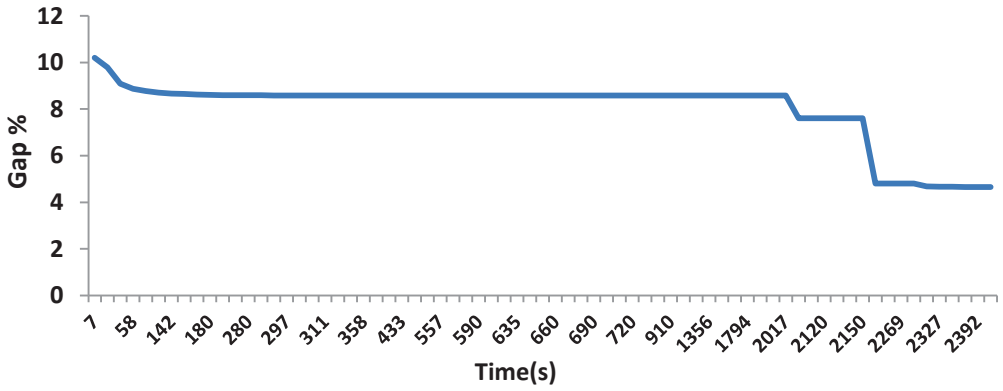


Figure 31- Convergence Rate (with additional constraints)

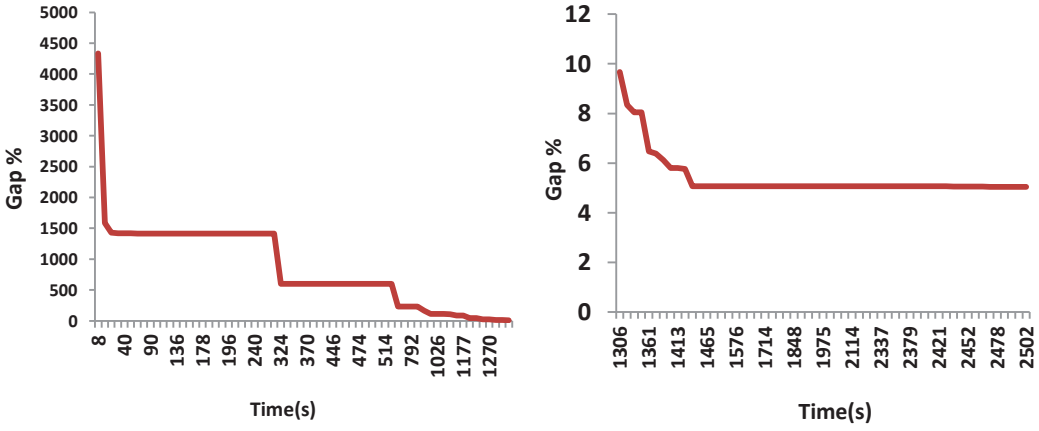


Figure 32- Convergence Rate (without additional constraints)

The enriched model with additional constraints starts from about 10% gap and ends up with 4.65% gap in about 20 minutes (figure 31). The model without those constraints starts from a large gap (4000%) and then after about 10 minutes reaches less than 10% gap (figure 32, left side) then the gap decreases to 4.8% in next 10 minutes (figure 32, right side). Both models eventually reach a similar solution. The strengthening constraints do have a beneficial effect and may be necessary for more difficult problems.

4.3.3 Results

The mathematical model was coded in *GNU Linear Programming Kit (*GLPK) (see Makorin [20]). The GLPK code is available in appendix C. The required data for running the model was provided in excel csv (comma separated values) files. The model is a relatively hard MIP (mixed integer programming) problem so that the GLPK solver cannot solve it in a reasonable amount of time. Thus we used the ***Gurobi 5.0*** solver [15] to solve the model. The LP solution for the model with strengthening constraints was found in just 4 seconds and the MIP solution within 4.5% gap in about 6000 seconds, with the objective as follows:

Best Objective: **8,385,012$**

Best Bound: 8,762,668

It cannot be guaranteed that the found solution is the best MIP solution, but it can be claimed that it is less than 5% worse than best potential solution for MIP.

Figure 33 illustrates the resulting campaign schedule for the whole horizon (13 weeks), for the assumed sawmill. Each line in this figure illustrates the schedule for one week. Red and yellow boxes show the setup times for class and campaign, respectively. Other colors represent production of a unique class. The class number for each is provided at the top right of the figure; 9 colors for 9 classes. Also each box in week carries a number, the campaign number. For example, at the second period campaign 8 and 12 from class 1 and campaign 17 and 35 from class 2 were run.

As can be seen a variety of different campaigns and classes are selected during these thirteen weeks. All nine classes were selected at least once and 60 campaigns out of 162 potential campaigns are chosen.

In 13 weeks, 27 setups for classes and 67 setups for campaigns occurred. The following formula shows the setup times and running time calculations:

$$Total\ Setup\ Time = \ Class\ Setup\ Times\ + \ Campaign\ Setup\ Times$$

$$Total\ Setup\ Time = 27 * 30 + 67 * 10 = 1480\ minutes$$

$$Total\ Running\ Time = Total\ Time - \ Total\ Setup\ Time$$

$$Total\ Running\ Time = 13 * 40 * 60 - 1480 = 31200 - 1480 = 29720\ minutes$$

$$\text{Utilization } = \frac{Total\ Running\ Time}{Total\ Time} * 100 = \frac{29720}{31200} * 100 = 95.2564\ \%$$

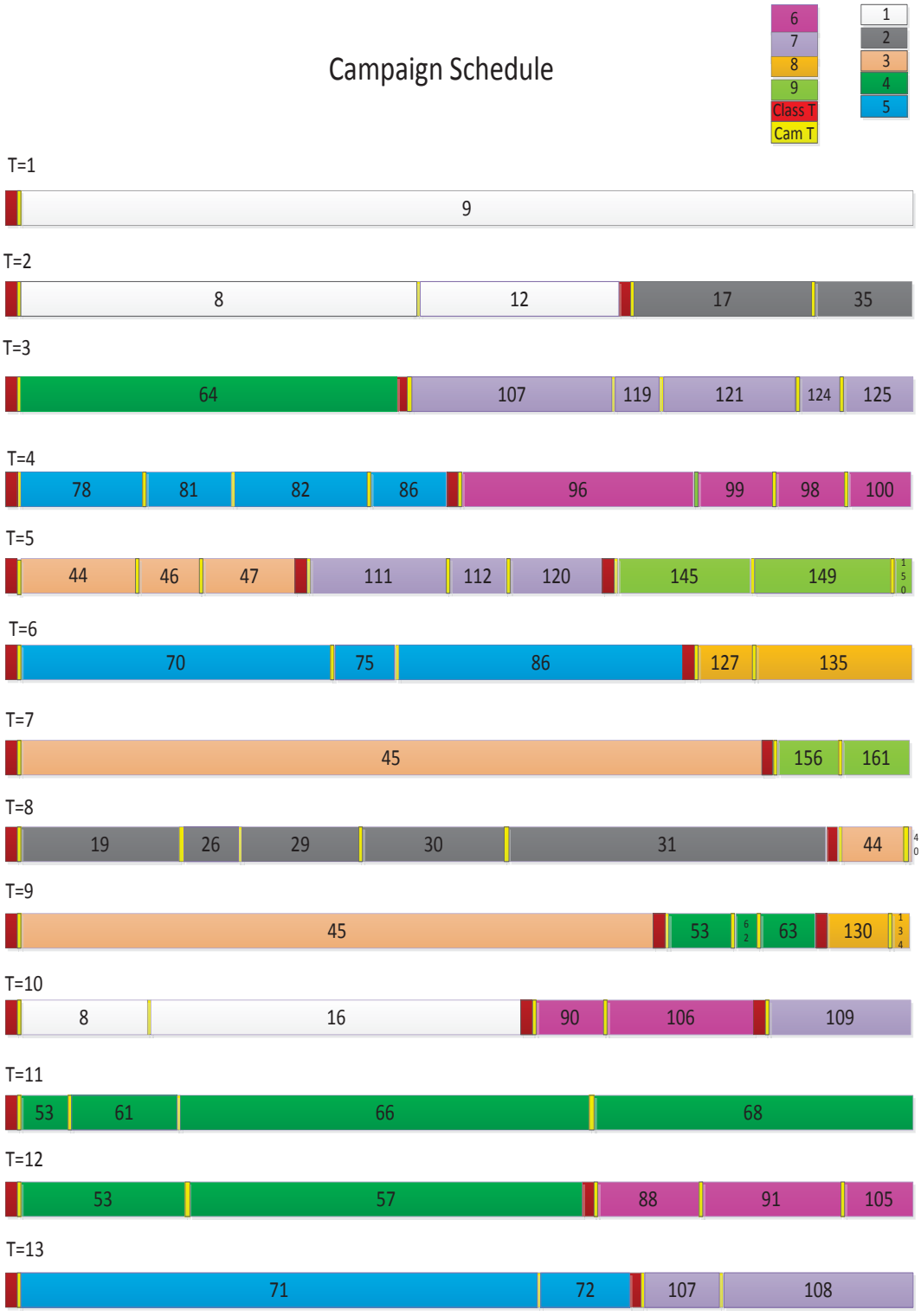


Figure 33- Campaigns Schedule

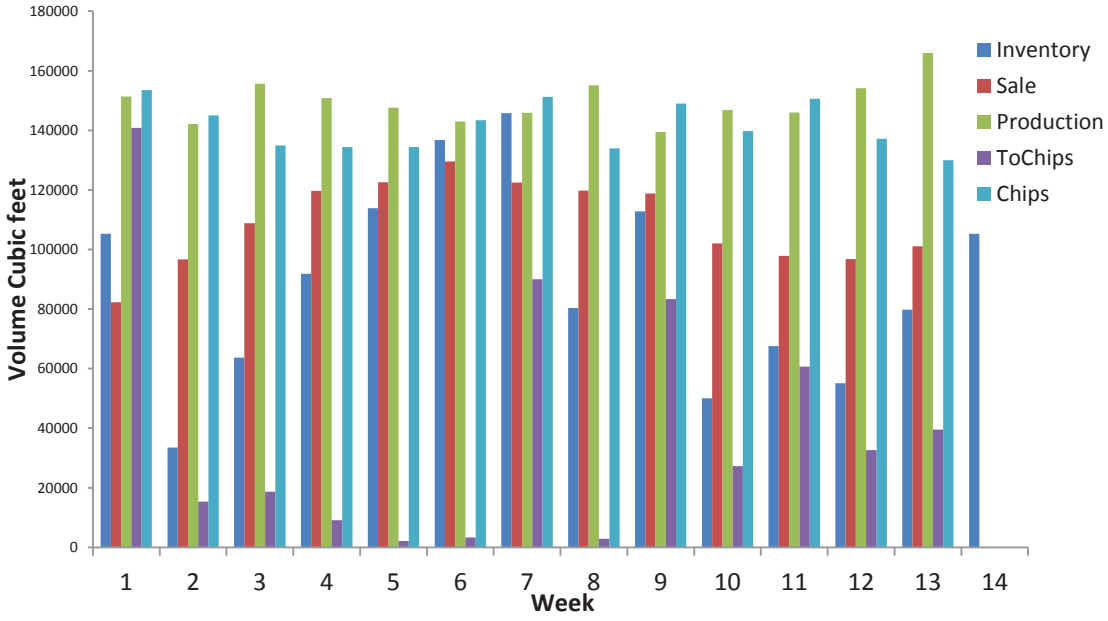


Figure 34- Weekly Results

Figure 34 illustrates the types of weekly aggregate results that we can get from the model. The blue color shows the inventory (ft^3) at the start of each week. In this example, the total starting inventory is 105227 (ft^3) and ends with the same inventory and the end of horizon (13 weeks), as required by constraint (11) in the model.

As can be seen in the figure 34, inventory decreases at the end of the first period and most of that goes to chips. It turns to chips because there is no demand at the high or medium level for that lumber and also there is an inventory holding cost for keeping inventory (tochips is high in the first week). Then, the inventory increases because there would be a demand for the second species from period 5 to 9. The peak of inventory happens at the seventh week so model decides to produce chips to lower the inventory. At the end of period 9, since there is no more demand for species two, extra inventory turns to the chips and inventory level decreases in the next four periods. Finally at the last period, inventory reaches the starting inventory. The model keeps more inventory for the

periods when species two demand is added. Another property of the solution is that whenever inventory is high (week1, 7 and 9), lumber converts to chips to decrease the inventory level for the coming period. The total inventory never gets stable because demand for species two happens in the middle of the horizon (weeks 5 to 9). The model increases the inventory till period 5, keeps it at a high level till period 9 and then decreases the inventory.

The red color shows the total lumber sale for each week in nominal cubic feet. As illustrated in figure 34, total sales increase smoothly until period five. The reason is the manager wants to sell lumber mostly in the market's high and then medium levels to get the most profit of the market. At the first week, since there is no control on starting inventory, the sale in not very high and a large part of the production goes to chips. After that, by wisely choosing the campaign and classes according to the market levels, the sale increases. Sale stays high for the periods we have demand for both species (weeks 5 to 9) and then decreases and reaches the stable condition. There is meaningful relation between sale and inventory in a way that, whenever inventory is high the sale is high too and model can response better to the market demands.

The green bars show the total nominal lumber output production (ft^3) in each week. It depends on two factors; i) nominal yield of chosen campaigns in that period, where campaigns with higher volume yield cause more lumber production, ii)less setup time for classes and campaigns result in more running time and production. Note that it is beneficial for sawmill to run all the time since there is no operational cost (such as labor, electricity...) considered.

The light blue color demonstrates the chips produced during the sawing process. Campaigns with higher volume yield result in less chips production. For example in period 13, higher production and lower chips can be found, on the other side in periods 9 and 11 high chips and low production is occurred.

The purple bars determine the amount of lumber (ft^3) turned into chips, "ToChips", in each week. Although both chips and ToChips are treated as chips products of sawmill, they are created differently. Chips are produced during the sawing process according to the target yield of each campaign. ToChips are actually the part of lumber production which cannot be sold in the market for a good price so they turned into chips. Therefore, total produced chips consist of two parts. First part is the amount of chips happens because of the process of sawing and it is unavoidable and the second part, "ToChips", is the lumber pieces without any good price in market which turn into chips. It usually happens when inventory is high and helps to decrease the inventory holding cost. There are three reasons for having high amount of ToChips:

1- There is no limitation for chips sale in each week, so all produced chips are going to the market.
2- The price of chips for many products is higher than lower level in the market; therefore, instead of selling the lumber in the cheapest market price, the model turns the lumber into chips and earns more benefits. In this example, the first 43 products are cheaper in the lower level than chips so we never have sold them in the lower market level. Also for the second species chips price is more than the lower market level price for all products, except six by six lumber pieces.
3- There is no operational time or cost for turning lumber into chips.

The following figure provides us the same data but trends easier to see. The relations between inventory and ToChips or production and chips are can be distinguished.

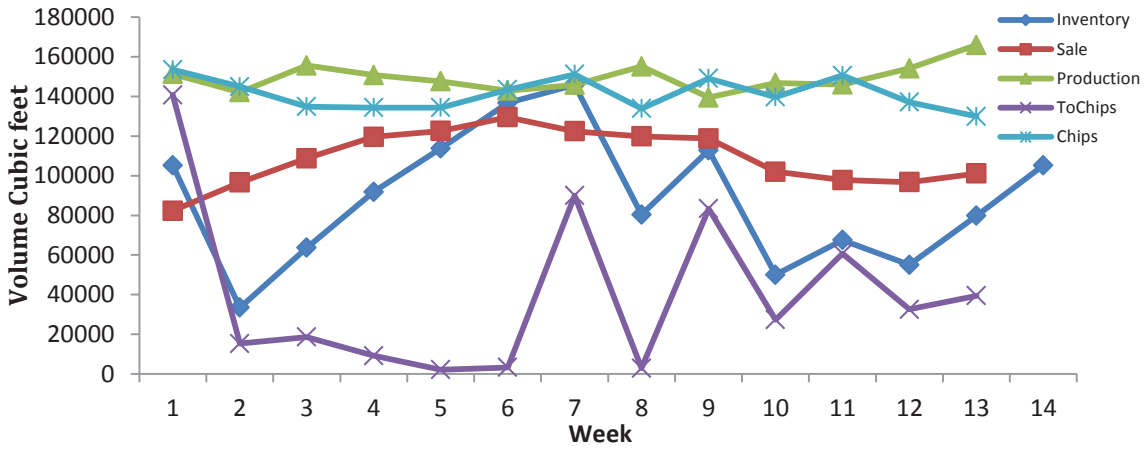


Figure 35- Weekly Results Graph

Table 11- Weekly Running Results

Period(week)	Inventory(ft^3)	Lumber Production(ft^3)	Lumber Sale(ft^3)	ToChips(ft^3)	Chips(ft^3)
1	105226.8	151363.4213	82323.5	140802.5789	153563.446
2	33464.17	142163.1559	96647.02	15342.3276	144974.375
3	63637.98	155636.2804	108803.8	18647.1862	134897.086
4	91823.29	150779.1739	119631.9	9145.416514	134429.811
5	113825.1	147624.1237	122566	2142.335145	134358.727
6	136740.9	142949.702	129518.1	3281.866792	143449.173
7	145743.3	145836.2286	122409.9	89964.85832	151224.918
8	80352.13	155132.8988	119819.6	2841.749479	133903.627
9	112823.7	139442.8914	118816.2	83394.63808	149046.863
10	50055.75	146838.2068	102032.3	27311.51599	139750.305
11	67550.13	146001.1783	97854.57	60653.1586	150595.546
12	55043.58	154174.555	96777.6	32594.19497	137149.486
13	79846.34	165931.2502	101090.4	39460.40717	129948.303
Sum	1136133	1943873.07	1418291	525582.23	1837291.67
Average	87394.86	149528.7	109099.3	40429.4	141330.13

Table 11 demonstrates the result for total products in all 13 weeks. As can be seen, the inventory balance equation always holds and the inventory shortage variable in constraints (6) are 0. Thus production is feasible. For example, starting inventory in period 9 depends on the previous period inventory, production, sale and tochips:

$$Inventory\ (8) + Production(8) - Sale(8) - ToChips(8) = Inventory(9)$$

$$80352.1 + 155132.9 - 119819.6 - 2841.7 = 112823.7$$

Total lumber production equals to the sum of the lumber sale and tochips (table 11)

$$Total\ Lumber\ Production = Total\ Lumber\ Sale + Total\ ToChips$$

$$1943873 = 1418291 + 525582$$

$$Lumber\ Sale\ Percentage = \frac{Total\ Lumber\ Sale}{Total\ Lumber\ Production} * 100 = \frac{1418291}{1943873} * 100 = 73\%$$

$$ToChips\ Percentage = \frac{Total\ Tochips}{Total\ Lumber\ Production} * 100 = \frac{525582}{1943873} * 100 = 27\%$$

Therefore, 73% of the lumber production goes directly to the market while 27% of that, according to the low market price, transfers to chips. Note that in this example, total outputs (lumber production and chips, table 11) are higher than the input rate, even though we have setup times in each period so we were expecting less output than input. The reason is that for lumber output, nominal outputs of the campaigns are considered. On the other hand for chip output, one minus target yield is assumed. The nominal output is always greater than the target output, so total output, which equals to nominal output, plus one minus target output, is always greater than one.

$$Total\ Output\% = Nominal\ Output\% + (1 - Target\ Output\ \%)$$

$$Total\ Output = Lumber\ Production + Chips$$

$$Total\ Output = 1943873 + 1837291 = 3781164\ ft^3$$

$$Total\ Input = Number\ of\ Periods * Input\ Rate$$

$$Total\ Input = 13 * 276987 = 3600831\ ft^3$$

4.3.3 Sensitivity Analysis and Discussion

Many parameters affect the model. These include chips selling price and inventory holding cost. In 4.3.3.1 and 4.3.3.2, the effects of changing these parameters are illustrated. In the last section of sensitivity analysis, 4.3.3.3, market demands are deliberately changed in different periods to examine how the model will react to these changes by choosing proper campaigns.

4.3.3.1 Scenario One: Changing the Chips Selling Cost

In the assumed model, the chips price for species one and two are 3 and 2 dollar per cubic feet respectively. In this section, we reduce these prices by one half. Then, for both species, selling the lumber, even in the lower level of the market, is more beneficial than turning them into the chips. Therefore, the model will avoid producing ToChips. On the other side, market lower level sale is increased significantly. Total revenue is decreased by 31.8% because almost half of the logs still become chips during the sawing process. The new objective function, within less than 4% gap from the upper bound, is 5,719,358 when the upper bound is 5,936,409.

$$Objective\ Decrease = \frac{8{,}385{,}011 - 5936409}{8385011} * 100 = 31.8\%$$

Figure 36 displays the total sale in each level and amount of lumber turned into chips.

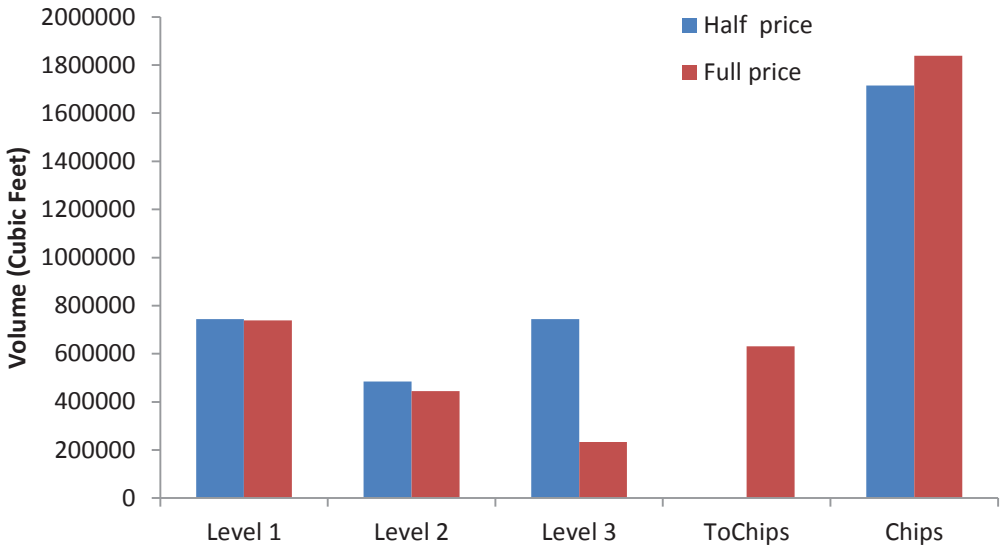


Figure 36- Scenario 1 Results

Table 12- Scenario 1 Results

	Half Chips Price	Full Chips Price
Level 1	743943.8 ft^3	739283.7 ft^3
Level 2	484622.9 ft^3	445215.8 ft^3
Level 3	743742.8 ft^3	233791.3 ft^3
ToChips	0 ft^3	630809.1 ft^3
Chips	1714533 ft^3	1838052 ft^3
Total	3686842 ft^3	3887152 ft^3

Table 12 compares these two models with different chip prices. Both models are almost selling the same amount at the first level, because it is the most beneficial market level. At the second and third market levels, with lower chip prices, we attempt to get higher yields and thus we see that we can sell more. Table 13 shows that we need more variety

in our campaigns to deal with the low chip prices. The total running time and utilization are lower in the low priced chips case because of necessary setups to change campaigns. Also this model tends to choose campaigns with higher lumber volume yields. Table 13 displays the number of campaigns, classes and utilization of each model. All chosen classes, campaigns and running times are available in appendix E

Table 13- Scenario 1, Total Campaigns

	Half Chips price	Full Chips price
Campaign setups	91	67
Class setups	44	27
Total setup time (min)	2230	1480
Total running time	28970	29720
Utilization	92.85 %	95.25 %

4.3.3.2 Scenario Two: Changing the Inventory Holding Cost

In this run, the inventory holding cost is reduced to half of the previous value. A new objective function, within less than 3% gap from the upper bound, is 8,555,043 when the upper bound is 8,806,884. The new objective function is just around 2% higher than the old one, which means that inventory holding cost is not the substantial part of the objective function.

$$Objective\ Increase = \frac{8{,}555{,}043 - 8{,}385{,}011}{8{,}385{,}011} * 100 = 2.03\%$$

Figure 37 demonstrations the total inventory in each period for both cases. As expected, less inventory holding cost results in keeping more inventory. Note that in both cases, we imposed the constraint that final inventory must equal ending inventory.

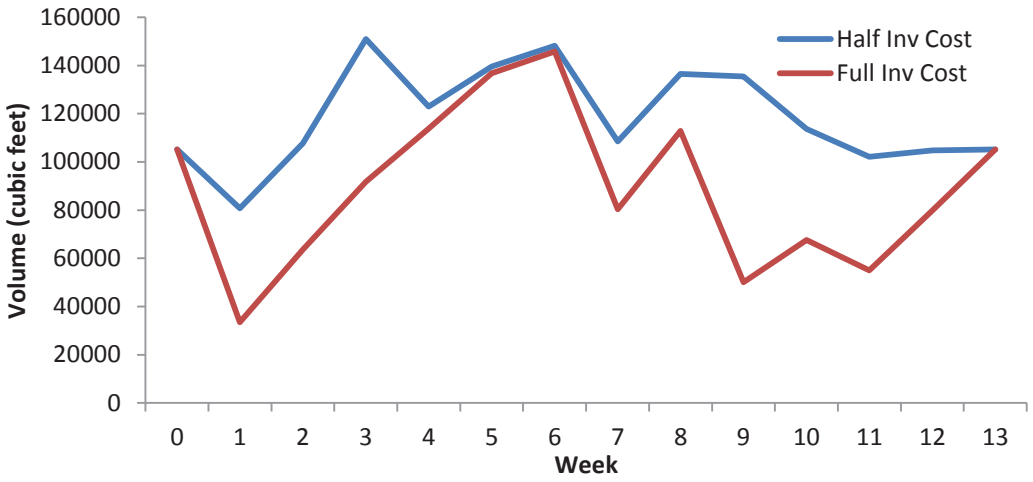


Figure 37- Scenario 2, Inventory Comparison

The other major difference is the chosen classes in each week. In the base case scenario, since the demand for species 2 just occurs in periods 5 to 9, classes 8 and 9 (producing species 2) only runs in those periods. In the current scenario, the mill runs class 9 at period 2 and keeps the inventory of species 2 for the next three periods because of the low inventory holding cost. Appendix E illustrates the all running classes and campaigns for each model. One of the main effects is the reduction in campaign setups. The base case incurred 67 campaign setups while only 53 were used in scenario 2, resulting in higher availability and higher production.

4.3.3.3 Scenario Three: Changing Demand Pattern

To explore the relation between the market demands and the chosen classes and campaigns in each period, the demands of some specific products were changed for periods 2, 3, 4. Table 14 illustrates the periods with different demands and running classes and campaigns for each of them. In the following table W stand for width, T for thickness and L for length. For example in period two, W = 4 means products with the

width four inch. + (-) 75% shows 75 percent increase (decrease) in the upper bounds of the market levels and maximum sales for those specific lumber pieces. The total demand in all 13 weeks did not change, since any change in the market levels recovered after 8 weeks for the same products.

Table 14- Scenario 3 Comparing Resulting Classes and Campaigns

		Current Case		Base Case	
Period	Demand changes	Classes	Campaigns	Classes	Campaigns
2	W = 4: + 75% W = 6: -75%	1, 2	3, 17, 36	1, 2	8, 12, 17, 35
3	T = 3: -75% T = 4: +75%	1, 6	3, 11, 90, 94, 105	4, 7	64, 107, 119, 121, 124, 125
4	L = 10 : +75% L = 14 : -75%	4	53, 56, 61	5, 6	78, 81, 82, 86, 96, 98,99, 100

As can be seen in tables 14, the schedule can be very different from the base case according to the different demand patterns. In the second period, since all classes can produce lumber with width 4(in), there is no limitation on chosen classes. Campaign 3 from class 1, which emphasizes larger widths, is selected. As logs in class 1 rarely produces 6x6 lumber pieces, campaign 3 which focusses on 4x4 lumber pieces, is chosen to cover the extra demand for 4x4 products. Table 9 provides details of the campaigns price lists.

At the third period, demands for the products with thickness 4(in) increased and at the same time demand for thickness 3(in) decreased. Among the selected campaigns, campaign 11, emphasizing thickness 4(in), is a proper choice for this period. Other

campaigns are selected by the model to produce the required products, while none of the campaigns in the base case emphasize the 4 (in) thickness.

A more interesting case happens at period 4, when extra demand for specific length is required. While classes 3 to 7 are sorted just by the length of the logs, the model has the option to select the suitable class according to the related demand for specific length. In period 4, class 4 which just produces length 8 and 10 foot lumber (mostly 10) is selected to satisfy the extra demand for length 10 (ft). Thus we see that the model is quite responsive to the specific length by choosing the appropriate class. In the base case, classes 5 and 6, mostly 12 and 14 foot lumber pieces were chosen.

The output fraction of each campaign is available in appendix D. You can also find all the chosen classes, campaigns and running times in appendix E.

4.4 Conclusion

In this chapter of thesis, we showed that the proposed mathematical model for scheduling the classes and campaigns for the sawmill can be computed effectively. A realistic scale problem with 70 types of products, 2 species, 162 campaigns and 9 log classes, over 13 periods has been formulated and solved as a large mixed integer programming model. This problem includes 2223 binary variables (for each campaign and each class in each period (162+9)*13=2223).

Although we do not reach provably optimal solutions, good solution within less than 5% gap than the upper bound are found in less than two hours (running on MacBook Pro laptop with core i5 CPU). Given that faster computers are readily available, particularly if one wants to use the cloud computing option discussed in Gurobi [15], this suggests that this approach can be readily used in industry.

In order to demonstrate that the model is sensitive to the data, we created three different scenarios from the base case. In scenario 1, we reduced chip value and observed that the model responded by attempting to get higher yields and selling more lumber. In scenario 2, we reduced inventory carrying costs and observed that the model scheduled longer campaign runs with fewer setups. More inventory was carried but more product was also produced and sold. In scenario 3, we created different demand patterns in certain periods, by decreasing demand for some types of lumber and increasing it for others. In all three scenarios, the model responded to the data changes by computing appropriate new solutions.

Note that in this study, we scheduled the campaigns which are previously created using the methods of chapter 3 according to the provided default price lists. However, there are other possible methods to create additional price lists and consequently new campaigns. By solving the linear programming problem, without requiring an integer solution, we can very quickly get a set of shadow prices (see Williams[37]) on each product (using the shadow prices of the inventory balance set of constraints, constraint set 1 in the model) for each period. This thus gives us 13 sets of shadow prices. Now we can feed the campaign generator with these shadow prices as new price lists; this will result in new potential campaigns which can be added to the existing set of campaigns. Shadow prices are only applicable to linear programs so the integer solution is not necessary. Alternatively, one can solve the model for an integer solution and fix the integer variables at the solution attained. Then solving the linear program with these variables fixed will produce new sets of shadow prices. This idea of using shadow prices to generate new campaigns is in the spirit of the column generation procedures discussed in chapter 4 of Williams and similar to the method was used by Maness and Norton [22]. Since we had enough campaigns to meet the demand profiles we were working with, we did not pursue this here but it might be a useful future research direction.

As we discussed in Chapter 2, the implementation of a planning model like the one we have been examining here is usually in the context of a rolling planning horizon approach, where only the first period is actually implemented. In the next chapter we look briefly at a rolling planning version of our work.

Chapter 5

Rolling Planning Horizons

In this chapter, we look at a rolling planning horizon approach for the campaign production planning model discussed in chapter 4. In a rolling planning horizon approach, once the model is solved, the resulting schedule is just applied to the immediate period. At the start of the next period, after the actual production yields and sales have been realized, and any new data on demand obtained, the initial inventory and planned sales data are updated and the model solved again while the planning horizon is moved forward for one period. This process, moving the horizon forward one period, updating data and replanning is repeated for each period.

Figure 39 illustrates the sample rolling planning applied to our model. As previously discussed, we generated the first four periods demand randomly and considered them as known demands; for the remaining periods, the expected value of the demand is used as a forecasted demand. As you can see in this figure, the first run's horizon is from week 1 to 13 and demand is known for weeks 1 to 4 and forecasted as constant for weeks 5 to 13. While, for the second run we rolled the horizon 1 week ahead and the new planning horizon is from week 2 to 14 and the demand for week 5 becomes known (simulated). Random numbers, distributed uniformly between 0.75 and 1.25, are multiplied by the average demand of each product to simulate the demand for week 5. Note that inventory at the end of the first period for the first run, is assumed as the initial inventory for the second run.

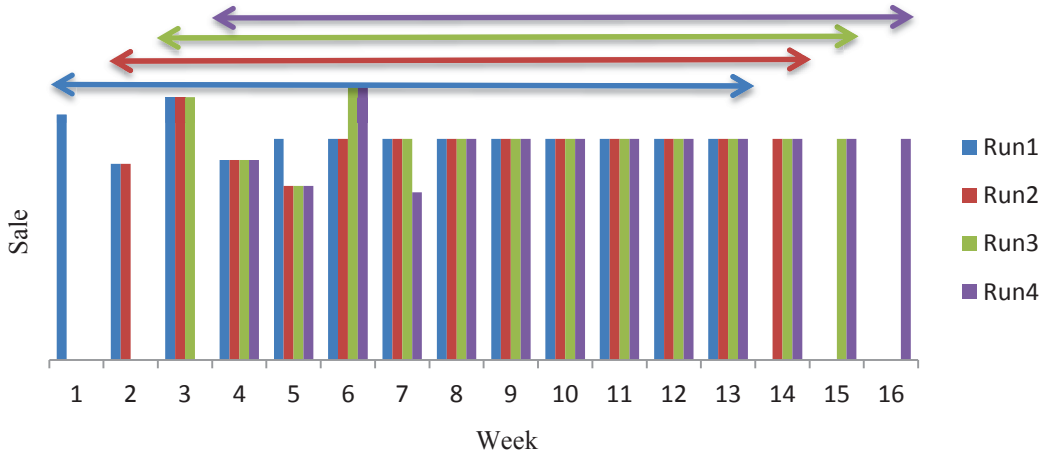


Figure 38- Sample Rolling Planning Horizons

5.1 Results

Table 15 compares the results of 4 runs by applying the concept of rolling horizon. As can be seen, the differences between the solutions are less than 1%, and they are all similar to each other. The solution time for each run was about two hours.

Table 15- Comparing Results of 4 Runs

Run	1	2	3	4
Best Solution	8385011	8443796	8383015	8381776
Upper Bound	8762668	8741632	8729947	8658081
Gap	4.5%	3.53%	4.14%	3.30%

Table 16 illustrates the detailed campaigns schedules of four runs by applying rolling planning horizons. It shows which campaigns were calculated to be run in each week and also running time durations in terms of proportion of week. It is notable that the campaigns schedules are quite different for these runs. For instance at week number 4, eight different campaigns are used in run1. In runs 2, 3 and 4 the number of campaigns run in period 4 are only 4, 5 and 3 respectively. None of the campaigns originally chosen

in run 1 are used in run 2, 3 or 4. Campaigns 110 now is used in runs 2, 3 and 4 while it was not used in run 1. With 162 possible campaigns we have considerable flexibility and can respond as demand becomes better known.

Table 16- Campaigns Schedules of 4 Runs

Run1			Run2			Run3			Run4		
Campaign	Week	Duration	Campaign	Week	Duration	Campaign	Week	Duration	Campaign	Week	Duration
9	1	0.9833	1	2	0.3774	3	3	0.1074	110	4	0.3358
8	2	0.4367	13	2	0.1734	15	3	0.2454	112	4	0.0383
12	2	0.2184	17	2	0.2271	90	3	0.1727	123	4	0.0332
17	2	0.1954	22	2	0.0397	106	3	0.1152	78	5	0.1655
35	2	0.1078	23	2	0.1366	107	3	0.0725	84	5	0.1803
67	3	0.4135	56	3	0.1598	124	3	0.2243	86	5	0.1801
107	3	0.2205	61	3	0.2954	44	4	0.1337	127	5	0.0815
119	3	0.0483	90	3	0.111	45	4	0.1816	128	5	0.1103
121	3	0.1452	102	3	0.1148	55	4	0.131	145	5	0.0669
124	3	0.0445	105	3	0.1685	107	4	0.0921	155	5	0.0478
125	3	0.0779	145	3	0.0795	110	4	0.4033	156	5	0.0462
78	4	0.1343	161	3	0.0042	55	5	0.2141	157	5	0.0463
81	4	0.0949	2	4	0.5916	66	5	0.4418	1	6	0.2009
82	4	0.146	107	4	0.0865	145	5	0.1384	9	6	0.6095
86	4	0.0838	110	4	0.2284	157	5	0.0763	91	6	0.1521
96	4	0.2549	112	4	0.0518	161	5	0.0835	17	7	0.3532
98	4	0.083	1	5	0.2503	17	6	0.1762	36	7	0.3636
99	4	0.0764	88	5	0.1468	18	6	0.3837	145	7	0.0646
100	4	0.0684	100	5	0.1157	35	6	0.2554	157	7	0.0927
44	5	0.127	102	5	0.1477	131	6	0.0633	161	7	0.0801
46	5	0.0671	104	5	0.0481	134	6	0.0756	53	8	0.2848
47	5	0.1022	105	5	0.0819	70	7	0.1208	65	8	0.3378
111	5	0.1494	161	5	0.1427	71	7	0.4799	101	8	0.195
112	5	0.0613	1	6	0.0399	135	7	0.1659	135	8	0.1282
120	5	0.1028	9	6	0.2048	155	7	0.0501	9	9	0.5706
145	5	0.1458	107	6	0.2111	156	7	0.0523	88	9	0.1135
149	5	0.1501	126	6	0.0764	157	7	0.0685	102	9	0.2784
150	5	0.0195	134	6	0.0767	17	8	0.2754	1	10	0.2112

70	6	0.3417	135	6	0.1008	45	8	0.6913	9	10	0.768
75	6	0.0671	144	6	0.1507	50	9	0.6441	88	11	0.2139
86	6	0.313	160	6	0.0564	53	9	0.1981	100	11	0.0831
127	6	0.0605	70	7	0.2669	63	9	0.1203	101	11	0.1601
135	6	0.1718	76	7	0.3705	72	10	0.1612	105	11	0.2167
45	7	0.8163	83	7	0.3376	78	10	0.1934	107	11	0.1593
156	7	0.0395	53	8	0.1526	81	10	0.1593	126	11	0.1169
161	7	0.1066	61	8	0.378	82	10	0.1431	1	12	0.1248
19	8	0.1753	111	8	0.1212	88	10	0.1703	4	12	0.4372
26	8	0.0607	125	8	0.1616	106	10	0.1227	9	12	0.413
29	8	0.1286	135	8	0.1282	107	10	0.2701	1	13	0.1613
30	8	0.1562	9	9	0.9833	112	10	0.0372	9	13	0.6027
31	8	0.3523	1	10	0.1899	116	11	0.0418	110	13	0.1985
40	8	0.0042	9	10	0.2636	119	11	0.1388	70	14	0.123
44	8	0.0687	109	10	0.1003	120	11	0.2186	71	14	0.2012
45	9	0.6972	111	10	0.1288	121	11	0.2559	86	14	0.1761
53	9	0.0707	125	10	0.2716	3	12	0.2457	107	14	0.1764
62	9	0.0233	65	11	0.3737	9	12	0.7334	108	14	0.2147
63	9	0.0606	72	11	0.2893	45	13	0.9833	112	14	0.028
130	9	0.0664	74	11	0.2996	8	14	0.194	116	14	0.0264
134	9	0.0194	53	12	0.1503	15	14	0.2931	1	15	0.2029
8	10	0.1411	66	12	0.3345	90	14	0.1061	9	15	0.6538
16	10	0.4078	107	12	0.1587	100	14	0.1034	106	15	0.1058
90	10	0.0743	125	12	0.1802	104	14	0.1651	70	16	0.1511
106	10	0.1611	126	12	0.1305	106	14	0.0882	71	16	0.4566
109	10	0.1574	9	13	0.6606	9	15	0.9833	88	16	0.1919
53	11	0.0527	102	13	0.3061				89	16	0.1588
61	11	0.1139	1	14	0.1244						
66	11	0.4514	9	14	0.8548						
68	11	0.3528									
53	12	0.1817									
57	12	0.4326									
88	12	0.1115									
91	12	0.1533									
105	12	0.0751									
71	13	0.5697									
72	13	0.0974									
107	13	0.0822									
108	13	0.2091									

Rescheduling the campaigns in each period affects the production and consequently total sales and inventory in a way that sawmill respond better to the current market situation. In each run, constraints set 11 require the model to have an ending inventory equals to the starting inventory; therefore total productions equal to total sales over the planning horizon. Note that the starting inventory for each run of the rolling planning horizon is different from others, therefore the ending inventory, which equals to starting inventory, will be different; while, total productions still equal to total sales in each run.

To be more realistic in dealing with an evolving knowledge of demand, the concept of rolling production planning scheduling was applied. Although the results for a whole season are obtained from solving the model, we assume that only the first week is implemented. By updating the inventory, receiving more accurate data about the future, and solving the model again, we can produce production plans that respond to the new data. This updating of the data and rescheduling occurs at the end of each period to achieve a better result, given a current state of the system.

Chapter 6

Conclusion and Future Research

This thesis presents a modeling framework for sawmill scheduling and control problem that consists of two main levels: operational and management. At the operational level, first different log classes and price lists are combined to create potential campaigns, then mill management schedules these campaigns to maximize the total benefits at the sawmill.

To make the problem tractable we assume logs are truncated cones and that they are without internal defects. Considering the real shapes of logs adds substantial complexity to our model but does not change the priciples behind our approach. Further, accurate data of scanned logs is required, and we need to model how the curve sawing equipment will cope with the irregular shapes. Previously similar problems to the one treated in this thesis have been modeled and solved without above assumptions, but these models were limited in the number of cutting patterns they can consider, and they take a long time to solve. In our approach, more than 8000 different patterns are generated and the best pattern is quickly obtained for each log. The main difference between our work and previous ones is that in our work, all possible patterns are a priori generated, sorted and stored. Then, for each log, the best pattern can be chosen quickly from among these stored patterns. In contrast, in most other works, the patterns for each log are generated individually. The given price lists are used to provide us with flexibility for creating a variety of campaigns. They are also practical because price lists, known as internal prices

in some studies, can easily be implemented in industry. As mentioned in chapter 4, other ways are of generating price lists are possible such as using the shadow prices from the optimization model.

The mixed integer programming model, introduced in chapter 4, determines the campaigns to be run, when they will be run, and for how long they will be run. The integer variables in the model represent the setup times, while, in most related studies, setup times are ignored. We considered two different setup times: campaign setup time and class setup time. Without binary variables, an optimal solution to the model can be found in a few seconds. When these variables are introduced the solving process becomes more complicated. To get better solutions faster, some strengthening constraints have been added to the model. With a model with 71 products, 2 species and 162 campaigns over 13 periods, a model solution time of less than two hours with less than a 5% gap is workable. Future research may suggest ways of reducing solution times by further strengthening of the formulation. To ensure the results are reliable, randomly selected products have been checked for congruence in the context of the supply chain. Additionally, the model has been run for different scenarios and the results obtained are responsive to the differences. Also, the model runs performed with a rolling planning horizon resulted in solutions with similar objective functions values.

Models that include a rolling planning horizon replan at the end of each planning period, after the previous period solution has been implemented. Results of the four runs employing a rolling planning horizon are discussed in chapter 5. Similar objective function values for these runs were obtained, while the schedules generated from each run differed as new information on demand became available.

The flexibility of our model is demonstrated by the fact that different combinations of campaigns lead to reasonable solutions. So for instance if there is not enough supply for the model to run a specific class or campaign, alternative classes and campaigns can be used instead.

Future work on the model could focus on making it more realistic. For instance we assumed that a sawmill is capable of running the entire planning horizon without any down time. If we were to model mill downtimes as a stochastic process, it would be worthwhile studying how the proposed rolling planning horizon could respond to this uncertainty. Other work might involve considering setup times that reflect the dependency of present decisions, regarding which classes and campaigns to run, on previously made class and campaign decisions.

The ability to produce smaller lumber from larger pieces could also be included. For example lumber with dimensions of 4x4x16 can be broken down by length into 2 pieces of 4x4x8 or by width into 4 pieces of 1x4x16. This would give the model additional capability to satisfy the arriving demand.

We have used fairly simple three level demand curves to model market demand. It may be worthwhile to consider more complex demand processes both within a period and over the planning horizon.

In this study internal prices are used as the control policy for guiding the cutting pattern optimizer to generate sets of cutting patterns that can meet the required output. These prices are predetermined according to 20 different types of price functions independent from market conditions and the current inventory level. Although these 20 price lists resulted in a reasonable variety of campaigns that made it possible to meet the types of

demand examined here, it would be possible to generate customized price lists based on the current market and inventory situations, for instance using the shadow prices from the scheduling model, to obtain an output of campaigns that is tailored to the current situations.

Sorting logs, sawing, and scheduling are important considerations for sawmill managers. While, the model proposed in this thesis can still be improved, our results show that this approach is capable of implementation at an industrial scale.

Bibliography

[1] Anthony, R. N. (1965). Planning and control systems: a framework for analysis. Division of Research, Graduate School of Business Administration, Harvard University.

[2] Baker, K. R. (1977). An experimental study of the effectiveness of rolling schedules in production planning. *Decision Sciences*, *8*(1), 19-27.

[3] Bowater Mersey Oakhill Sawmill, (2012). Scanned logs data. Supplied to Eldon Gunn by Hans Peterson, Halifax, Canada.

[4] Carlson, R. C., Beckman, S. L., & Kropp, D. H. (1982). The effectiveness of extending the horizon in rolling production scheduling. *Decision Sciences*,*13*(1), 129-146.

[5] Carnieria, C., Mendoza, G.A., Luciano, G.G. (1994), Solution procedures for cutting lumber into furniture parts. *European Journal of Operational Research*, 73(3), pp. 495-501.

[6] Carnieria, C., Mendoza, G.A., (2000), A fractional algorithm for optimal cutting of lumber into dimension parts. *Annuals of Operations Research*, 95, pp.83-92.

[7] Chand, S., Hsu, V. N., & Sethi, S. (2002). Forecast, solution, and rolling horizons in operations management problems: A classified bibliography. *Manufacturing & Service Operations Management*, *4*(1), 25-43.

[8] Funck J., Zeng Y., Bruner C., Butler D., (1993), SAW3D : A real shape log breakdown model. *5th International Conference on Scanning Technology and Process Control for the Wood Products Industry*, October 25-27, Atlanta, GA, Wood Technology, San Francisco CA. 19p.

[9] Funck J., Zeng Y., (1995), Integrating an expert system and dynamic programming approach to optimize log breakdown while considering lumber grade. *Proceedings of the 49th Annual Meeting of The Forest Products Society*, June 28, Portland

[10] Gaudreault, J., Forget, P, Frayert, J.M., Rousseau A., Lemieux S., D'Amours S., (2009), Distributed operations planning in the softwood lumber supply chain: models and coordination. *Bibliothèque et Archives nationales du Québec*.

[11] Gilmore, P.C., Gomory, R.E., (1961), A linear programming approach to the cutting stock problem. *Operations Research* , 9, pp. 849-859

[12] Goulet, P. 2006. Optitek: User's Manual. Document E-4130. Forintek

[13] Gunn, A. E. 1996. Some aspects of hierarchical planningin forest management. Proceedings of a Workshopon Hierarchical Approaches to Forest Management in Public and Private Organizations. Toronto, ON, May 25-29, 1992

[14] Gunn, A. E., MacDonald, C., Cameron, A. and Caissie, G., (2012) Scotsburn Dairy Group Uses A Hierarchical Production Scheduling and Inventory Management System to Control its Ice Cream Production, Department of Industrial Engineering, Dalhousie University (submitted for publication to Interfaces).

[15] Gurobi: Gurobi Optimization. Website: http://www.gurobi.com/

[16] Harry Freeman and Son Ltd., (2012). Nominal, target and actual dimensions. Supplied by e-mail to Eldon Gunn by Richard Freeman. Personal communication, Halifax, Canada.

[17] Hax, A. C., & Meal, H. C. (1973). Hierarchical integration of production planning and scheduling (No. TR-88). MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESEARCH CENTER.

[18] Kadipasaoglu, S. N., & Sridharan, V. (1995). Alternative approaches for reducing schedule instability in multistage manufacturing under demand uncertainty. *Journal of Operations Management*, *13*(3), 193-211.

[19] Lumber & Plywood Estimating Price Guide. (2011) Website: http://www.acehardware.net/estimate/

[20] Makorin, A. 2010. Modeling Language GNU MathProg: Language Reference for GLPK Version 4.45 (DRAFT, December 2010), Free Software Foundation, Inc Boston, MA, USA.

[21] Maness, T.C. and Adams D.M., (1991) The combined optimization of log bucking and sawing stratégies. *Wood and Fiber Science*, 23, 296-314.

[22] Maness; T.C and Norton, S.E.,(2002) Multiple-period combined optimization approach to forest production planning. *Scandinavian Journal of Forest Research* , 17, 460-471.

[23] Marinescu, M. V., & Maness, T. C. (2010). A Hierarchical Timber Allocation Model to Analyze Sustainable Forest Management Decisions. Mathematical and Computational Forestry & Natural-Resource Sciences (MCFNS), 2(2), Pages-117.

[24] Maturana, S., Pizani, E., & Vera, J. (2010). Scheduling production for a sawmill: A comparison of a mathematical model versus a heuristic. *Computers & Industrial Engineering*, *59*(4), 667-674.

[25] Mendoza, G.A., Meimban, R.J., Luppold, W.J. and Arman, P.A.,(1991) Combined log inventory and process simulation models for the planning and control of sawmill operations. In: *Proceeding 23rd GIRP International Seminar on Manufacturing Systems*; Nancy, France.

[26] Mesarović, M. D., Macko, D., & Takahara, Y. (1970). Theory of hierarchical, multilevel systems.

[27] Reinders M.P., Hendriks Th.H.B., (1989), Lumber production optimization. *European Journal of Operational Research*, 42, pp. 243-253.

[28] Rönnqvist, M. (2003). Optimization in forestry. *Mathematical programming*,*97*(1), 267-284.

[29] Schmidt, G., & Wilhelm, W. E. (2000). Strategic, tactical and operational decisions in multi-national logistics networks: a review and discussion of modeling issues. *International Journal of Production Research*, *38*(7), 1501-1523.

[30] Sohrabi, P., (2012), A Three-stage Control Mechanism for the Lumber Production Process of a Sawmill Based on a Powers-of-two Modelling Approach. Master Thesis, Dalhousie Universtiy, Canada.

[31] Suter WC., Calloway JA., (1994), Rough mill policies and practices examined by a multiple-criteria goal program called ROMGOP. *Forest products journal*, 44, pp. 19-28.

[32] Tejavibulya S., (1981), Dynamic programming sawing models for optimizing lumber recovery. *Dissertation Abstracts International Part B: Science and Engineering [DISS. ABST. INT. PT. B - SCI. & ENG.]*, 42, 70 pp.

[33] Todoroki C., (1997), Developments of the sawing simulation software AUTOSAW: linking wood properties, sawing and lumber end-use. *Proceedings of the Second Workshop,* IUFRO S5.01–04, South Africa, August 26–31, 1996, 241–247, Nepveu, G. Ed. Publication Equipe de Recherches sur la Qualité des Bois. INRA-Nancy, France.

[34] Todoroki C., Ronnqvist M., (2001), Log sawing optimisation directed by market demands. *NZ journal of forestry*, 45, pp. 29-33.

[35] Todoroki C., Rönnqvist M., (2002), Dynamic Control of Timber Production at a Sawmill with Log Sawing Optimization . *Scandinavian Journal of Forest Research*, 17, pp. 79-89

[36] Todoroki C., Rönnqvist M., (1999), Combined primary and secondary log breakdown optimization

[37] Williams, H.P. 1991. Model Building in Mathematical programming (3rd Edition), Wiley, New York.

[38] Xie, J., Zhao, X., & Lee, T. S. (2003). Freezing the master production schedule under single resource constraint and demand uncertainty. *International Journal of Production Economics*, *83*(1), 65-84.

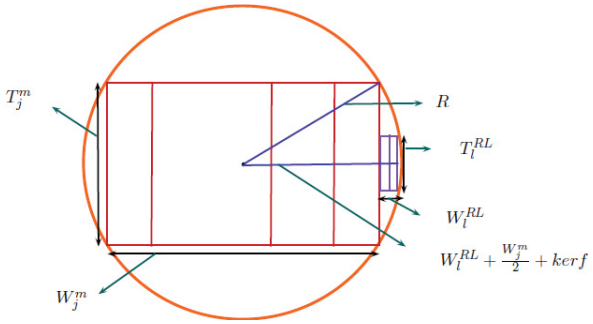
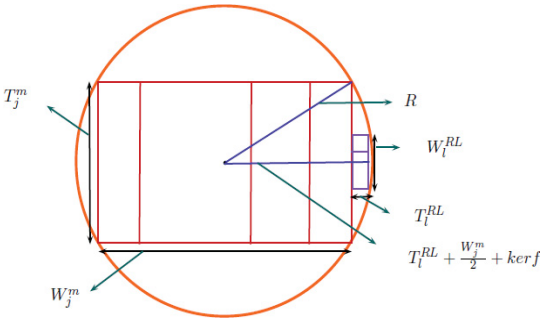
[39] Zanjani, M. K., Kadi, D. A., & El Fath, M. N. (2007). A stochastic programming approach for production planning in a manufacturing environment with random yield. CIRRELT.

[40] Zanjani, M. K., Nourelfath, M., & Ait-Kadi, D. (2010). A multi-stage stochastic programming approach for production planning with uncertainty in the quality of raw materials and demand. *International Journal of Production Research*,48(16), 4701-4723.

[41] Zhang, S.Y., Tong, Q.J., (2005). Modeling lumber recovery in relation to selected tree characteristics in jack pine using sawing simulator Optitek. *Annals of Forest Science*. 62(3), 219-228.

Appendix A: Details of Cutting Patterns

Categories 3 and 4:

Vertical right-left sub-cuts	Horizontal right-left sub-cuts:
For each T_l^{RL} where: $$T_l^{RL} \leq T_j^m$$ We calculate all the combinations of possible widths and findW_l^{RL} . $W_l^{RL} = (\sum_i Z_{i,l} * (W_i + \text{Kerf})) - \text{kerf}$ Where: $$W_l^{RL} \leq \sqrt{R^2 - \left(\frac{T_l^{RL}}{2}\right)^2} - \frac{W_j^m}{2} - kerf$$  Figure 39- Vertical right-left sub-cuts	For each T_l^{RL} where: $$T_l^{RL} \leq R - (\frac{W_j^m}{2} + \text{kerf})$$ We calculate all the combinations of possible widths and findW_l^{RL} .$W_l^{RL} =$ $(\sum_i Z_{i,l} * (W_i + \text{Kerf})) - \text{kerf}$ Where: $$W_l^{RL} \leq 2 * \sqrt{R^2 - (T_{RL} + \frac{W_j^m}{2} + \text{kerf})^2}$$  Figure 40-Horizontal right-left sub-cuts

Horizontal Above-below cut:

- 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 the W_{min}= Min $[Seq_j^{AB}]$

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

Step 3: Calculate rWaneUD, rWaneSide and rWane

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

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

$$rWane = Maximum(rWaneUD, rWaneSide)$$

Step 4: Calculate Lwane:

$$Lwane = \left\lfloor Min\left\{L, L * \left(1 - \frac{rWane - R_S}{R_L - R_S}\right)\right\}/2\right\rfloor * 2$$

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 $W_j^{AB} = W_j^{AB} - (W_{min} + kerf)$

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.

Vertical Above-below cut:

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 the W_{min}= Min $[Seq_j^{AB}]$

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

Step 3: Calculate rWaneUD, rWaneSide and rWane

rWaneSide

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

rWaneUD

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

$$rWane = Maximum(rWaneUD, rWaneSide)$$

Step 4: Calculate Lwane:

$$Lwane = \left\lfloor Min\left\{L, L * \left(1 - \frac{rWane - R_S}{R_L - R_S}\right)\right\}/2 \right\rfloor * 2$$

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.

<u>Vertical right-left cut:</u>

- Seq_j^{RL}: Set of sub-cuts' widths of the above-below 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 the W_{min}= Min $[Seq_j^{RL}]$

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

Step 3: Calculate rWaneUD, rWaneSide and rWane

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

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

$$rWane = Maximum(rWaneUD, rWaneSide)$$

Step 4: Calculate Lwane:

$$\text{Lwane} = \left\lfloor \text{Min}\left\{\text{L}, \text{L} * \left(1 - \frac{\text{rWane} - \text{R}_\text{S}}{\text{R}_\text{L} - \text{R}_\text{S}}\right)\right\}/2 \right\rfloor * 2$$

Step 5: If $Lwane = L$, set L as the length of remaining cuts ($\text{Z}_{\text{ijk}} = \text{Z}_{\text{ijk}} + \text{Z}_{\text{i,j}}$ where $\text{L}_\text{k} = \text{L}$) and STOP.

Otherwise go to Step 6.

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

Step 7: Let $\text{W}_\text{j}^{\text{RL}} = \text{W}_\text{j}^{\text{RL}} - (\text{W}_{\text{min}} + \text{kerf})$

Step 8: Update the set of width, $[\text{Seq}_\text{j}^{\text{RL}}] = [\text{Seq}_\text{j}^{\text{RL}}] - [\text{W}_{\text{min}}]$

Step 9: If $[\text{Seq}_\text{j}^{\text{RL}}] \neq \emptyset$, go to Step 1

Otherwise Stop.

<u>Horizontal right-left cut:</u>

For each thickness $\text{T}_\text{j}^{\text{RL}}$ when main cut with thickness T_f^m and width W_f^m is fixed:

Set $\text{W}_\text{R} = \text{W}_\text{L} = \frac{\text{W}_\text{j}^{\text{RL}}}{2}$, $\text{B}_\text{R} = 1, \text{B}_\text{L} = 0$ (starting from the right half), $\text{Z}_{\text{ijk}} = 0$

Step 1: Find the $\text{W}_{\text{min}} = \text{Min}\ [\text{Seq}_\text{j}^{\text{RL}}]$

Step 2: Keep i if $\text{W}_{\text{min}} = \text{W}_\text{i}$

Step 3: Calculate rWaneUD, rWaneSide and rWane

rWaneSide

$$= \sqrt{\left((\text{W}_\text{R} * \text{B}_\text{R}) + (\text{W}_\text{L} * \text{B}_\text{L})\right)^2 + \left(\left(\frac{1 - wanePrSide}{2}\right) * \text{T}_\text{j}^{\text{RL}} + \text{kerf} + \frac{W_\text{f}^\text{m}}{2}\right)^2}$$

rWaneUD

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

$$rWane = Maximum(rWaneUD, rWaneSide)$$

Step 4: Calculate Lwane:

$$\mathrm{Lwane} = \left\lfloor \mathrm{Min}\left\{\mathrm{L}, \mathrm{L} * \left(1 - \frac{\mathrm{rWane} - \mathrm{R_S}}{\mathrm{R_L} - \mathrm{R_S}}\right)\right\}/2 \right\rfloor * 2$$

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

Otherwise go to Step 6.

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

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

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

Step 9: If $[\mathrm{Seq_j^{RL}}] \neq \emptyset$, go to Step 1

Otherwise Stop.

Appendix B: Python 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 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 can be fit into the remaining area of the circle, “List” (including area and combinations of widths) will be pushed into a heap called CattingPattern. The same method is used for the 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”. The specifications of the created patterns include radius of the pattern, thickness of the main cut, widths of the sub-cuts in the main cut, and width and thickness of above below and right left 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 with radius between the small and large radius of the log are chosen. The length of each lumber piece 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 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 pickle 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 fuorth code called “RunSort”, first loads the pickled files of the first 2 log classes, combines and sorts the logs to create more classes.

Then the same as “Run” code, it calculates and writes the outputs of the new campaigns in an excel file.

Main cutting pattern

```
from xlwt import Workbook, Formula, easyxf, Utils
import pickle
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[l]) #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[l])
#Area calculations considering kerf
                    List=AreaCutRLVer, seqRLVer[t], TRLVerHor[l], "Vertical RL"
                    heapq.heappush(CuttingPattern,list(List)) #push the "List":(area, sequence
and T) into a heap called CattingPattern

        if TRLVerHor[l]<=maxTRLHor: # horizontal cuts
           Pcheck=pow(R,2)-pow((TRLVerHor[l]+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[l]) #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[l])
#Area calculations considering kerf
                    List=AreaCutRLHor, seqRLHor[t], TRLVerHor[l], "Horizontal RL"
                    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 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)))*U
D+(sqrt(pow(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 HORIZONTL 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=getIndx(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 legth f 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=l-(r-rs)*l/(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/(rl-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[ithick],LengthSell[ilength])
                ptyp=PieceSizeDict[piece]
                if M==1:
                    npcs=1
                else:
                    npcs=2
                volSaw+=npcs*PieceTargVolume[ptyp]
                for iobj in range(len(PriceLists)):
                    objVal[iobj]+=npcs*PriceLists[iobj][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.*l
            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 8-10 inside the class
PS2=0.4 # probability of having logs with length 10-12 inside the class
PS3=0.2 # probability of having logs with length 12-14 inside the class

PL1=0.0118 # probability of having logs with length 8-10 inside the class
PL2=0.1242 # probability of having logs with length 10-12 inside the class
PL3=0.2365 # probability of having logs with length 12-14 inside the class
PL4=0.1307 # probability of having logs with length 14-16 inside the class
PL5=0.4968 # probability of having logs with length 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,T
otWid) # 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=getIndx(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],S
Logs[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],S
Logs[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*l/144.
      elif dim==1:
         vol=pow(t,1.5)*w*l/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*l/144.)*(1.0468*w+ 0.19851*t + 0.0487*l +2.49827 - (0.0357*w*t)-
(0.00002781*w*t*l))
   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*l/144.
```

```
        else:
            vol = w*t*l/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 "PercentYiled:", 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,ValueE
,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,Le
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 C: GLPK Code

```
  /*Bi-Level Approach to Sawmill Price*/
param m, integer, >0;
/* m for number of product types */
param n, integer, >0;
/* n for number of campaigns */
param pr, integer, >0;
/* pr for number of periods */
param lev, integer, >0;
/* for number of market levels */
param cNo, integer, >0;
/*for number of classes*/
param ProdRate, integer, >0 ;
set I := 1..m;
/* I set of product types*/
set S := 1..2;
/*S set of species */
set K := 1..n;
/* K set of campaigns*/
set T := 1..pr;
/* T set of periods */

set C := 1..cNo;
/*C set of Class*/
set L := 1..lev;
set TI := 0..pr;
set Q dimen 3;
/* holding cost */
set ZZ dimen 3;
/* output */
set mx dimen 3;
set mn dimen 3;
set smx dimen 3;
set smn dimen 3;
set inp dimen 2;
set res dimen 2;
set KK dimen 1;
set KKK dimen 1;
set ccc dimen 4;
set ppp dimen 4;
set inIn dimen 2;
set KC within K cross C ;
```

```
param PC:= 20;
/*Penalty Cost for out of range inventory */

param ST >=0, default 1.0/240. ;
/*setup time for campaign 10min */

param STC>=0, default 1.0/80.;
/*setup time for class 30min*/

param mem{k in K, c in C},>=0;
/*shows campaigns of each class*/
table tab_membership IN "CSV" "Data5.csv":
  KC <- [Campaign, Class], mem ~ member;

param CountMem{c in C}, default sum{k in K:(k,c) in KC} 1;

param h{i in I, s in S, t in T}, >= 0;
/* inventory holding cost for product i in period t */
table tab_holdingcost IN "CSV" "Data.csv" :
  Q <- [Product, Species, Period], h ~ holding;


param sMin{i in I, s in S, t in T}, >= 0;
/* the minimum allowable sales of product i in period t */
table tab_smin IN "CSV" "Data.csv" :
  smn <- [Product, Species, Period], sMin ~ saleMin;

param sMax{i in I, s in S, t in T}, >= 0;
/* the maximum allowable sales of product i in period t */
table tab_smax IN "CSV" "Data.csv" :
  smx <- [Product, Species, Period], sMax ~ saleMax;

param iMin{i in I, s in S, t in T}, >= 0;
/* the minimum allowable inventory of product i in period t */
table tab_invmin IN "CSV" "Data.csv" :
  mn <- [Product, Species, Period], iMin ~ InvMin;

param cost{i in I, s in S, t in T, l in L}, >= 0;
/* Selling price for product i in period t at level L*/
table tab_costmx IN "CSV" "Data1.csv" :
  ccc <- [Product, Species, Period, Level], cost ~ Price;

param X{i in I, s in S, t in T, l in L}, >= 0;
/* product i upper bound for level L at period t */
table tab_Lmax IN "CSV" "Data1.csv" :
  ppp <- [Product, Species, Period, Level], X ~ Ubound;
```

```
param V{t in T, k in K}, >= 0;
/* the maximum amount of input run of campaign k in period t */
table tab_maxinput IN "CSV" "Data2.csv" :
  inp <- [Period, Campaign], V ~ Maxinput;

param y{k in K}, >= 0;
/* volume yield of each campaign */
table tab_CampaignYeild IN "CSV" "Data3.csv" :
  KK <- [Campaign], y ~ Yield;

param LogsCost{k in K}, >= 0;
/* Log cost per tone */
table tab_SupplyCost IN "CSV" "Data3.csv" :
  KKK <- [Campaign], LogsCost ~ LogCost;

param O{i in I, s in S, k in K}, >= 0;
/* output of product i from campaign k per unit volume input run of campaign k */
table tab_outputs IN "CSV" "Data4.csv" :
  ZZ <- [Product, Species, Campaign], O ~ output;
param inInv{i in I, s in S}, >=0;
/* sets the initial inventory */
table tab_initial IN "CSV" "Data7.csv" :
  inIn <- [Product, Species], inInv ~ initial;

/*****************************************************/

var Sale{i in I, s in S, t in T, l in L}, >= 0;
/* amount of product i sold in market level L at period t */

var Inv{i in I, s in S, t in TI}, >= 0;
/* inventory of product i at the end of period t */

var lostInv{i in I, s in S, t in T}, >=0;
/* Lost Invenotry amount */

var TP{k in K, t in T}, >= 0, <=1;
/* proportion of time that campaign k is running during period t */

var P{i in I, s in S, t in T}, >= 0;
/* production of product type i in period t */

var yP{k in K, t in T}, binary;
/*binary for setup time for campaigns within each class*/

var Z{c in C, t in T}, binary;
```

```
/*binary for setup between classes*/

var toChips{i in I, s in S, t in TI}, >=0;


/*****************************************************/

/* constraints */

/*1-production */
s.t. production{i in I, s in S, t in T} :
        P[i,s,t] = sum{k in K}TP[k,t]*(O[i,s,k]*V[t,k]);


/* 2-inventory balance */
s.t. inventory{i in I, s in S, t in T: i < 71}:
        Inv[i,s,t] = Inv[i,s,t-1]+ P[i,s,t]- sum{l in L}(Sale[i,s,t,l]) - lostInv[i,s,t] -
toChips[i,s,t];

s.t. inventoryChips{s in S, t in T}:
        Inv[71,s,t] = Inv[71,s,t-1]+ P[71,s,t]- sum{l in L}(Sale[71,s,t,l]) - lostInv[71,s,t]
+sum{i in I: i<71}(toChips[i,s,t]);

/* 3-time limit */
s.t. timeProportion{t in T}:
        sum{k in K,c in C:(k,c) in KC} (TP[k,t]+ yP[k,t]*ST) + sum{c in C}(Z[c,t]*STC)
<=1;

/*4- Sales upper bound */
s.t. saleUB{i in I, s in S, t in T}: sum{l in L}(Sale[i,s,t,l]) <= sMax[i,s,t];

/*5- Sales lower bound
s.t. saleLB{i in I, s in S, t in T}: sum{l in L}(S[i,s,t,l]) >= sMin[i,s,t];

#/*6- Inventory upper bound */
#s.t. InvUB{i in I, s in S, t in T}: Inv[i,s,t]<=iMax[i,s,t]+extraInv[i,s,t];

/*7- Inventory lower bound */
s.t. InvLB{i in I, s in S, t in T}: Inv[i,s,t]>=iMin[i,s,t]-lostInv[i,s,t];

/* 8. Starting stock */
s.t. starttoChips{i in I, s in S}:toChips[i,s,0] = 0;

/*9- Market maximum price */
s.t. marketLevel{i in I, s in S, t in T, l in L}: Sale[i,s,t,l]<=X[i,s,t,l];
```

```
/*10- Setup time*/
s.t. setupUP{k in K, t in T}: TP[k,t]<=yP[k,t];
s.t. setupLB{k in K, t in T}: TP[k,t]>=yP[k,t]*ST;

/*11- Setup for class*/
s.t. setupclass{k in K, c in C, t in T:(k,c) in KC}:yP[k,t]<=Z[c,t];
s.t. TotClassSetup{c in C, t in T}: sum{k in K:(k,c) in KC}
yP[k,t]<=CountMem[c]*Z[c,t];
display KC;
s.t. AtLeastOneInClass{c in C, t in T}: sum{k in K:(k,c) in KC} yP[k,t]>=Z[c,t];
s.t. AtLeastOnePerPeriod{t in T}: sum{c in C} Z[c,t] >=1;

/*12- Maximum inventory */
s.t. maxmimumInv{t in T}: sum{i in I, s in S:i<71}(Inv[i,s,t])<= 4*ProdRate;

/*8.1- Start&finish Inventory */
s.t. startInv{i in I, s in S }: Inv[i,s,0]= inInv[i,s];

s.t. endInv{i in I, s in S}: Inv[i,s,13] >= Inv[i,s,0] ;

/*13- minimum running time for each class */

s.t. mintimeClass{t in T,c in C}:
        sum{k in K:(k,c) in KC} (TP[k,t])>=STC*Z[c,t];

/*14- number of campaigns*/
s.t. maxcam{t in T}: sum{k in K} (yP[k,t]) <= 20;
s.t. maxcls{t in T}: sum{c in C} (Z[c,t]) <= 6;

/*****************************************************/

maximize obj: sum{i in I,s in S, t in T, l in L} (cost[i,s,t,l]*Sale[i,s,t,l] - h[i,s,t] * Inv[i,s,t]
- PC * lostInv[i,s,t])- sum{t in T, k in K}( (TP[k,t]* V[t,k]) * 0.0242646 * LogsCost[k]) ;

/* the objective is to maximze revenue */

solve;

/*****************************************************/
printf "Category 1, Product sale  \n";
printf " product-period:production \n";
printf{i in I,s in S, t in T} "%5d %5d %10g\n", i, s,t,  P[i,s,t] ;

#printf{k in K, c in C:(k,c) in KC} "%5d %5d %10g\n",k,c,  mem[k,c];

printf "Time proportion for each campaign  \n";
```

```
printf{k in K, t in T} "%5d %5d %10g\n", k, t,  TP[k,t];

table tab_result{k in K, t in T } OUT "CSV" "Campaigns.csv" :
  k ~ Campaign, t ~ Period, TP[k,t] ~ Time;

table tab_result{i in I, s in S, t in T } OUT "CSV" "Products.csv" :
  i ~ Product, s ~ Species, t ~ Period, P[i,s,t] ~ Production;

table tab_result{i in I, s in S, t in T, l in L} OUT "CSV" "Sales.csv" :
  i ~ Product,s ~ Species, t ~ Period, l ~ Level, Sale[i,s,t,l] ~ Sale;

table tab_result{i in I, s in S, t in T } OUT "CSV" "Inventory.csv" :
  i ~ Product,s ~ Species, t ~ Period, Inv[i,s,t] ~ Inventory;

/*********************************************************/

data;

param m := 71;
param n := 162;
param pr := 13;
param lev :=3;
param cNo := 9;
param ProdRate := 128205 ;
/* weekly production Rate */
/*param Factor := 0.0242646;
 ft^3 to m^3 /1.167 (m^3/tone) */

end;
```

Appendix D: Campaigns Fractional Outputs

Cam\Prod	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	0.00990	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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

Cam\Prod	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 E: Campaigns Scheduling Results

Scenario 1			Scenario 2			Scenario 3		
Campaign	Period	Duration	Campaign	Period	Duration	Campaign	Period	Duration
44	1	0.2672	2	1	0.2343	9	1	0.9833
55	1	0.0779	8	1	0.2339	3	2	0.3127
61	1	0.1137	96	1	0.0740	17	2	0.2219
77	1	0.0703	109	1	0.0705	36	2	0.4279
78	1	0.0573	112	1	0.0787	3	3	0.2774
83	1	0.1855	120	1	0.2461	11	3	0.1969
88	1	0.0287	9	2	0.3668	90	3	0.0892
108	1	0.0426	70	2	0.2682	94	3	0.2005
109	1	0.0184	71	2	0.1787	105	3	0.1902
112	1	0.0343	145	2	0.0288	53	4	0.1086
65	2	0.2312	161	2	0.0992	56	4	0.4693
89	2	0.2058	88	3	0.1377	61	4	0.3971
90	2	0.0935	100	3	0.2484	109	5	0.2028
107	2	0.1077	102	3	0.1727	124	5	0.1046
108	2	0.2677	107	3	0.1187	125	5	0.2967
140	2	0.019	126	3	0.2767	143	5	0.1465
55	3	0.1626	9	4	0.7932	155	5	0.06
56	3	0.1891	53	4	0.1735	157	5	0.1394
70	3	0.1689	14	5	0.1676	38	6	0.8119
71	3	0.2757	107	5	0.3229	134	6	0.0383
127	3	0.0346	125	5	0.1508	135	6	0.1123
128	3	0.1065	129	5	0.0890	21	7	0.3041
40	4	0.2222	135	5	0.2113	22	7	0.0647
88	4	0.0948	70	6	0.1156	70	7	0.2143
100	4	0.0746	85	6	0.1550	86	7	0.2401
102	4	0.2096	86	6	0.3740	156	7	0.0426
107	4	0.1697	143	6	0.1173	161	7	0.0717
108	4	0.1667	144	6	0.1923	38	8	0.7868
55	5	0.069	8	7	0.2512	129	8	0.0495
70	5	0.1039	13	7	0.7280	139	8	0.1262
71	5	0.1888	90	8	0.3083	8	9	0.2002
108	5	0.255	97	8	0.0042	16	9	0.2894
143	5	0.076	106	8	0.1736	109	9	0.1961
148	5	0.0266	113	8	0.1110	110	9	0.2726
155	5	0.0219	121	8	0.3571	61	10	0.3844
157	5	0.0716	70	9	0.1345	91	10	0.2453
161	5	0.0998	75	9	0.0755	96	10	0.1516
3	6	0.1196	76	9	0.4079	105	10	0.177
9	6	0.4309	78	9	0.2722	45	11	0.833
90	6	0.1238	135	9	0.0641	48	11	0.1462
91	6	0.284	3	10	0.3611	76	12	0.1436
55	7	0.1407	9	10	0.6181	78	12	0.1707
70	7	0.2147	9	11	0.6665	82	12	0.0592

87	7	0.0999	109	11	0.1874	109	12	0.1334
107	7	0.052	119	11	0.1086	120	12	0.1757
108	7	0.1512	56	12	0.3457	125	12	0.2675
134	7	0.0421	61	12	0.3199	1	13	0.2095
140	7	0.0396	92	12	0.1774	9	13	0.7697
141	7	0.0277	101	12	0.1154			
143	7	0.0542	1	13	0.2085			
149	7	0.0737	13	13	0.7706			
37	8	0.1766						
43	8	0.1008						
107	8	0.1068						
110	8	0.1916						
135	8	0.1433						
143	8	0.0555						
146	8	0.0863						
161	8	0.0559						
1	9	0.2841						
15	9	0.0545						
83	9	0.2297						
90	9	0.1462						
102	9	0.094						
108	9	0.0951						
112	9	0.0172						
55	10	0.0875						
61	10	0.1803						
69	10	0.1389						
89	10	0.2076						
100	10	0.075						
107	10	0.1061						
108	10	0.1379						
41	11	0.3161						
70	11	0.1048						
71	11	0.2735						
110	11	0.2139						
116	11	0.0334						
90	12	0.1066						
93	12	0.0223						
94	12	0.1681						
95	12	0.0614						
101	12	0.1115						
107	12	0.135						
108	12	0.3089						
112	12	0.0279						
53	13	0.1149						
54	13	0.3205						
68	13	0.1394						
70	13	0.1208						
71	13	0.2586						