ORDER-PROMISING AND PRODUCTION-PLANNING METHODS FOR SAWMILLS

by

Md. Shariful Islam

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

at

Dalhousie University
Halifax, Nova Scotia
December 2013

© Copyright by Md. Shariful Islam, 2013

DEDICATION

This thesis is dedicated to my wife Rukhsana Liza,
for her love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis focuses on the development of order-promising and production-planning methodologies for sawmills. Two types of demands are considered: contract and spot demands. Contract demands are those known commitments to be filled over a period of time, whereas spot demands are collected from the market for the current week, two, and three weeks prior to the delivery date. Campaigns are developed that describe how the various classes of logs will be processed under a given price list. The campaign produces lumber outputs that represent the proportions the sawmill optimizers would produce given this price list. An MIP model, which includes order promising and campaign production planning, is formulated to maximize revenue over a planning horizon. A new solution technique is introduced to simulate the effectiveness of the rolling planning horizon over the full year. The effectiveness of the solution techniques is studied using different scenarios.

LIST OF ABBREVIATIONS AND SYMBOLS USED

AATP	Advanced Available-to-Promise
ATP	Available-to-Promise
CTP	Capability-to-Promise
DSS	Decision Support System
FIFS	First-in-First-Served
FCFS	First Come First Serve
GO	Global Optimization
LP	Linear Programing
MIP	Mixed Integer Programming
MPS	Master Production Schedule
MSP	Multi-Stage Stochastic Programming
MTO	Make-to-Order
RFQ	Request-for-Quotation
UPM	Upper Partial Moment
UPV	Upper Partial Variance
d^F	Contract demand
d^0	Current week variable demand
d^2	Product demand to be delivered in 2 weeks
d^3	Product demand to be delivered in 3 weeks
S^F	Promised order for contract demand
S^0	Promised order for current week delivery
S^2	Promised order to be delivered in 2 weeks

S^3	Promised order to be delivered in 3 weeks
v^F	Product price for the contract demand
v^0	Product price for the current week variable demand
v^2	Product price to be delivered in 2 weeks
v^3	Product price to be delivered in 3 weeks

ACKNOWLEDGEMENTS

First of all, I would like to sincerely thank my supervisor, Dr. Eldon Gunn, for giving me the opportunity to work in this area. I want to thank him for his knowledge, supervision, advice, guidance, and patience that made my research achievement possible.

I would also like to thank my committee member, Dr. Corinne A. MacDonald, for her advice and enormous support during this research especially on campaign generation module.

I would also like to thank my committee member, Dr. Peter Gregson, for his advice and comments.

I would also like to thank Dr. Uday Venkatadri for his advice and continuous support.

I would also like to thank NSERC Forest Value Chain Optimization Network for funding this research.

I would also like to thank all of our group members Pegah Sohrabi, Andrew B. Matrin, Sina Saadatyar, Narges Sereshti and Mohammad Mallahi for their helpful comments and cooperation.

Finally, I would like to express my deepest appreciation to my family. Their endless support and encouragement made all of this possible.

CHAPTER 1
INTRODUCTION

A sawmill processes valuable forest resources (tree stems) to produce various types of lumber products. Scientific management of a sawmill at its operational level will reduce production waste, better meet market demands, and eventually increase profitability for the sawmill. Order-promising is the collection of orders from customers with a specific delivery due date. The order-promising and optimal production-scheduling for the promised order is an important endeavor for the sawmill, and has received attention by many researchers in recent years. This thesis addresses the issue of order-promising based on the available resources (i.e. logs, cutting facilities) for the sawmill product, and also focuses on optimal production-scheduling to deliver the finished product on time. A mixed integer programming model is developed to address these issues, and optimized to determine the orders that can be promised and the optimal production schedule, by maximizing the revenue through delivering promised order on time. The orders are promised considering the variability in the product demand and prices in every week. Logs are generated randomly to produce lumber products which include the uncertainty in throughput in the optimization model. The solution methods represent the different requirements and concerns in day to day sawmilling business and operations.

1.1. BACKGROUND

A sawmill receives different species, grades and classes of logs from the forest and processes them to produce a wide variety of lumber products, differentiated by species, dimensions, and grades. Due to the variable nature of tree-growth dimensions, the process yield is correspondingly uncertain at every stage of the lumber production operation. This uncertainty and the wide variety of end products make the sawmilling operation highly variable and complex. Amidst this uncertainty in operation, a sawmill manager collects orders from the market and produces lumber to fill promised orders. However, an order promised at a low price for a high quality of wood at the early stage of collecting may preclude later profitable actions. Thus, the improvement in the order-promising methodology plays a vital role for the success of sawmill operations.

The sawmilling operation starts from the bucking operation. Felled trees are bucked to produce logs of different grades, classes and standard lengths and the bucking operation is done in such a way that the total potential market value is maximized. The logs are generally graded according to the lumber quality that will be produced through the sawmilling operation, and the average diameter of the logs determines their class. Different species of logs, such as spruce, fir, larch, hemlock, white pine, red pine, etc., are processed in a series of manufacturing operations.

Some mills can buck trees; other receives logs bucked in the forest. The input logs are sorted based on log species, diameter classes and grades, and processed in the same order to ease the lumber sorting process. Log bark is removed, and then the debarked logs undergo the three major manufacturing steps of sawing, drying, and planing (surfacing). In the sawing operation, the debarked logs are broken down into various sizes of rough pieces of lumber. A single log produces lumber of different sizes through a series of cutting steps. Green (rough) lumber is produced through sawing, re-sawing, and edging processes, and collected in different bins. Each of the bins holds lumber of the same size and species. Next, lumber from the bins is bundled and dried in a kiln dryer to reduce the moisture content in accordance with customer orders. The dried lumber is then passed through a planing process, after which the lumber is graded based on physical defects and moisture content. The defects are removed by trimming operations, and dimensioned lumber of a specific length is produced. Fig. 1.1 shows the schematic flow diagram for the sawmill production operation. Generally, a portion of the produced lumber is transported to lumber suppliers to fulfill customer orders, while the rest is reserved as inventory. Inventory is required because sawmill demand is highly variable both seasonally and in the short-term between products. Lumber suppliers accept orders from customers, quoting quantity and due dates.

The decision to accept or reject an order and determine a due date for delivery on an accepted order is generally known as order promising. The manufacturer needs to fill the order on time to protect the manufacturer's reputation, as otherwise it may cause a loss of future orders. Thus, the dual processes of order promising and fulfillment should not be considered as two separate issues. Rather, these two activities are closely related to each other and involve sharing information among customers, suppliers, and the manufacturer.

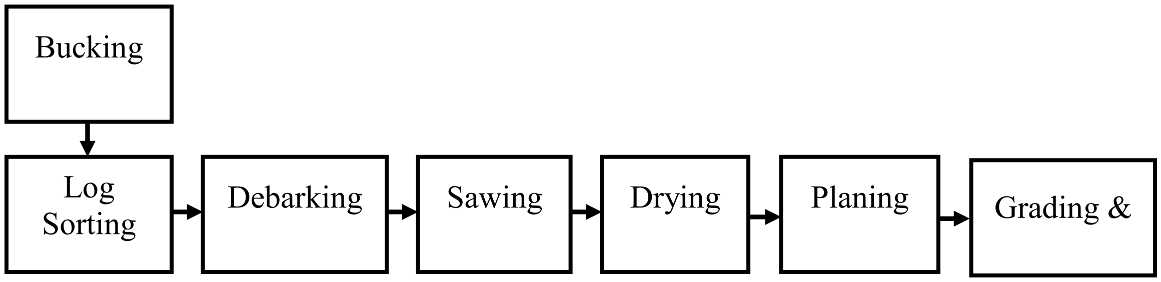

Figure 1.1: Lumber production flow diagram

Problems involved in order promising for lumber products have been specifically addressed by Azevedo et al. (2012), while those related to sawmill production planning and control have been the focus of numerous studies (Maness & Norton, 2002; Zanjani, 2009a; Saadatyar, 2012). Saadatyar (2012) maximized the revenue using a push manufacturing environment through proper production scheduling. This research is an extension of Saadatyar's (2012) research work. However, to the best of the author's knowledge, no study has yet been carried out that considers order-promising and sawmill production-planning to fulfill the promised order as an integrated problem. This research aims to fill this gap by integrating a sawmill's order-promising and production-scheduling operations.

1.2. MOTIVATION AND THESIS CONTRIBUTIONS

North American lumber industries' production aim has been to maximize the volume throughput (Gaudreault et al., 2009). The basic idea behind this type of production planning is to produce product efficiently and build inventory without short term consideration of customer demand. This results in a large inventory but low agility. The motivation for this type of production planning is to take advantage of economies of scale (i.e., large-scale production reduces variable unit cost) and is commonly known as the 'push-production' strategy. Although efficient in production, the push-production approach creates a large inventory, which may compromise the industry's main objective of making maximum revenue. Due to the lack of agility in this approach, product diversity is limited and the lumber industry cannot take advantage of a changing demand pattern.

On the other hand, a ‘pull-production’ strategy authorizes production based on current system state, which includes customer orders and inventory levels. i.e. “Produce-to-order” is one type of pull. In an assembly manufacturing environment, “pulling” means pulling a bill of materials in response to a demand. In the case of sawmill production planning, a system that is purely ‘pull’ is untenable because of the divergent nature of production throughput. In other words, it is not practical or economical for a sawmill to operate only when an order is received (Gunn, 2013). The motivation of this thesis is to move towards a pull-production environment for the sawmill industry and at the same time to increase the agility of the production.

Due to the characteristic non-homogeneity of tree stems, process yields are uncertain at every stage of sawmill production, from stem bucking to log sawing. The uncertainty in process yields and the wide variety of lumber demands make every stage of the management decision complicated. There are numerous challenges involved in managing a sawmill, as sawmills process logs of different species and produce an extensive range (e.g., different sizes and species) of lumber products, all the while collecting orders from customers on a weekly basis and endeavoring to deliver these orders on time. The challenges include: 1) deciding which customer order will be accepted that will earn maximum revenue by using available resources (i.e., raw materials, production facilities); and 2) determining the optimal campaign (production plan of different species and classes of logs) schedule that will fulfill the promised order and maintain a safe inventory level.

The thesis contribution is as follows:

1. Demonstrated that the shadow prices from the LP model can be used to generate campaigns that perform better in terms of both objective function and solution than the campaigns originally created in Saatadayar (2012) and Sohrabi (2012).
2. Developed an integrated MIP (mixed integer programming) model for order-promising and production-planning for sawmills.
3. Developed a dynamic solution technique to solve the model, where demand and price uncertainty are considered for every period.

1.3. THESIS ORGANIZATION

The rest of the thesis is organized as follows:

Chapter 2 presents a brief literature review. The literature is reviewed according to the following aspects: customer order acceptance, capacity loading, and sawmill production planning. The conclusion shows how the work in this thesis contributes to the existing literature.

Chapter 3 describes the generation of product pricelist based on shadow price. This chapter also describes how to generate new suitable campaigns that result in the reduction of solution time.

Chapter 4 introduces the mathematical model for order promising used in this work.

Chapter 5 describes a simulation approach that has been implemented within the Gurobi 5.5 optimization system. This allows testing the rolling planning horizon methodology as prices and demands vary over time. This chapter examines five different scenarios and presents model outputs illustrating the performance of the rolling planning horizon approach.

Chapter 6 draws conclusions and summarizes the results of this thesis. It also suggests possible future work.

CHAPTER 2

LITERATURE REVIEW

2.1. INTRODUCTION

Decisions regarding order promising and capacity loading involve the acceptance or rejection of customer orders, based on the availability of sufficient capacity to complete the orders prior to the delivery due date, which are given by the customers at the time of order placement.

In the context of a sawmill, due to the non-homogeneous and random characteristics of logs (i.e., diameter, number of knots, internal and external defects, etc.), the process yields are uncertain and at the same time the production process where many products are produced simultaneously from each log is divergent in nature. This means, it is not possible to produce product to meet a certain product demand without producing a lot of other products. Uncertainties in throughput and divergent nature of production make sawmill order promising and production planning complicated.

The production planning of a sawmill can be considered a combination of several classical production planning problems, which involve determining the optimal quantity of log consumption from different classes and the selection of corresponding cutting patterns to fit product demands (Zanjani et al., 2009a). Estimating the availability of sufficient capacity for the order is difficult because throughput and processing times are uncertain for the sawmill industry.

Order promising has two main issues on the sawmill. The first issue is that, if an order is accepted at a low price, this may preclude selling the lumber at a higher price later. Thus it may make sense to refuse an order, still produce lumber and wait until a better price is available later. The other issue is capacity utilization. By accepting orders and promising delivery in the future, this may lead to higher capacity utilization at the mill. However, if the mill is unable to meet these commitments, this may lead to customer dissatisfaction. The literature related to order promising and sawmill production planning is reviewed in the next two sections.

2.2. Customer Order Acceptance and Capacity Loading

In the literature, several authors addressed capacity loading as a vital issue for order promising. Wang et al. (1994) developed a workload-based policy for accepting or rejecting an order for an excessive demand non-MRP (material requirement planning) job shop under non-negotiable due-date requirements. The collected orders were prioritized based on the profit rate, which was defined as the profit per unit of time. More profitable orders were accepted as long as the entire scheduling period did not exceed the total available machine and manpower capacity. A neural network approach was proposed to solve the order-acceptance decision problem. Under the workload-based policy, orders are allocated to a planning period as long as for a specific resource (i.e., bottle neck of the system) the workload does not exceed the available capacity, and for all resources total workload does not exceed a specified maximum workload. Workload-based policy works well when the resources are in series. In the case of a sawmill, Products are produced in parallel. Therefore, workload-based order promising approach are difficult to apply for sawmill products.

Through a simulation experiment, Raaymakers et al. (2000a) investigated the performance of workload rules for order acceptance in batch chemical manufacturing with deterministic processing times. In the model, production orders were generated randomly and orders were accepted as long as the resulting work load per work center did not exceed a certain value. Considering the maximum work load of each work center, different scenarios were evaluated to find the variance of the makespan (the maximum completion times of a job in a job set), which were compared with the makespan obtained by simulated annealing (Raaymakers & Hoogeveen, 2000). The results showed that a capacity utilization of 53% can meet production needs on time. This work makes clear that simulation is a necessary tool to evaluate an order promising strategy.

Raaymakers et al. (2000b) investigated the performance of a regression-based makespan estimation policy and a workload-based policy against a detailed schedule policy (Raaymakers & Hoogeveen, 2000) for batch chemical manufacturing in a setting with deterministic processing times. The paper concluded that the regression-based makespan estimation policy performs better when there is high resource utilization and high job mix variety.

Ivanescu et al. (2002) extended the previous model (Raaymakers et al., 2000a, 2000b; Raaymakers & Fransoo, 2000c) for uncertain job processing times. Using a simulation experiment, the work compared the performance of the regression, scheduling and workload-based policy. The experimental result shows that regression and scheduling-based policies offer better performance than a workload-based policy. The performance of the regression-based policy increases with increasing uncertainty in processing time but decreases for the scheduling-based policy. A chemical processing industry produces various products from a single raw material, which bears resemblance to the sawmilling process. Raaymakers et al. (2000a, 2000b) studies were based on deterministic processing times. Although Ivanescu et al. (2002) considered the uncertain processing times for a batch processing chemical industry, the difference between a batch processing chemical industry and a sawmill is that, in a batch processing chemical industry, the raw material passes through various operational stages and processing steps that are different for each product, and sometimes the processing steps may have an overlap in time (two resources may be needed simultaneously). However, in a sawmill, multiple products are produced simultaneously although the drying and planning stages to deal with separate products. In Ivanescu et al. (2002)'s study, promised due dates for all products were the same and they had to be processed in one planning period. This study considers different due dates for different products, which can be processed in different planning periods.

Pibernik (2005) used the term Advanced Available-To-Promise (AATP), which includes Available-To-Promise (ATP) and Capability-To-Promise (CTP). He classified AATP into eight different generic types based on distinct criteria. The first criterion involves either finished goods inventory or inventory of supply-chain resources, including raw materials, work-in-process, finished goods, and production and distribution capacities. On the basis of finished goods inventory, AATP is better suited to a make-to-stock production environment, whereas the make-to-order production environment shows better performance on the basis of supply-chain resources. The operating mode of AATP is considered the second criterion in classifying AATP. AATP can be operated in real-time or batch mode. In real-time mode, order promising is made at the time of customer request. In batch mode, orders are collected within a batching-interval and then processed together. The third criterion is based on the interaction with the manufacturing resource planning and is classified into two categories: active and passive. An active AATP is integrated with the company's manufacturing resource planning that can be adjusted with the master schedule at the time of order promising, while passive AATP does not have any impact on manufacturing resource planning. Active AATP is suitable for make-to-order and passive AATP is suitable for make-to-stock manufacturing environments. Some add-on features were also considered with the generic AATP, such as AATP with substitute products, multi-location AATP, and AATP with partial delivery. After defining various generic AATP types, Pibernik (2005) developed a mixed integer programming model for batch AATP on the basis of finished goods inventory and a planning mechanism for real-time AATP. Partial deliveries were considered for both cases. Operating in real-time mode, AATP showed reduced performance compared to AATP operating in batch mode. The focus of this thesis is for a make-to-order production environment that differs from Pibernik's (2005) make-to-stock model.

Zhao et al. (2005) developed a mixed-integer-programming (MIP) model for available-to-promise with multi-stage resource availability and implemented in an assemble-to-order production environment. The model was formulated for a batch of customer orders that arrived within a pre-defined batching interval. The objective function of the model minimizes the due-date violation, inventory holding cost and a day-to-day production smoothness measure. The model considered multi-resource availability (including manufacturing orders, production capability and production capacity) and, from an order-promising point, allows the splitting of orders. The optimization-based ATP model was implemented at Toshiba Corporation to improve the performance over the company's historical performance of order promising where the ATP system collects orders over quarter-hour time intervals and returns commitments to the customer at the end of each ATP run. It was found that the optimization-based ATP model reduced due-date violations by about 18% and 2.3% of inventory holding costs. It also increased the resource utilization by 14%.

Venkatadri et al. (2006) developed an optimization-based decision support system (DSS) for order promising in supply chain networks of an assemble-to-order manufacturing industry. This model helps industry agents to promise orders in real time (e.g., quote due dates, prices, and quantity). Within this context, buyers and suppliers can constantly exchange information and negotiate product price, quantities and due date. The model considered the variation in product price with the lead time. All parties update production, warehousing, and distribution plans based on negotiations involving product price, features, and due dates. An order was defined as a single product or a combination of products, and early delivery was not desirable. The delivery of products could either be a single delivery or a span of deliveries over several time periods. The accepted order can be delivered in the next shipment if the formulated LP model provides a feasible solution. Otherwise, it will be delivered in the following or subsequent shipment. The authors recommend using a revenue management technique for further improvements.

Venkatadri et al. (2008) developed a linear programming algorithm for order-promising methods in a supply chain network adding new constraints such as penalties for late deliveries, and incentives for on-time deliveries with the previous work. The proposed model considered production and distribution data, initial inventories, committed orders and customer due date requests as an input, and production quantities in each period along with the sourcing path of each product in the supply chain network as an output. The proposed model also estimated the marginal cost and maximum availability of a customer-requested product in each time period in response to a request-for-quotation (RFQ) from a customer. The model described a multi-period, multi-commodity flow allocation problem whose objective is to minimize costs while at the same time planning and allocating its internal flows to allow for resources plans, utilization, and distribution plans. The model generates RFQs to the suppliers considering make-to-order production capacity constraints protocol. The model maximizes the revenue by minimizing the network cost and provides an estimation of the price to be quoted. This thesis does not consider any supply chain network. It only considers the production of a sawmill.

Applying revenue-management concepts to assemble-to-order (ATO) manufacturing environment, Harris and Pinder (1995) developed models for optimal pricing and capacity decisions. The general objective of revenue-management techniques is to maximize profit over the planning horizon by deciding whether to accept or reject a given order by anticipating future profitable orders. In the developed models, customer orders were segmented into two classes: class 1 customers were able to place their orders in advance, and class 2 customers placed their orders on a last-minute demand basis. The model was developed in two stages: optimal pricing and optimal reallocation. Allocation and reallocation of the optimal capacities were determined by the optimal protection level (i.e., the number of units reserved for a class). In this model, allocation for a particular class of customer was considered to be a function of the class rate and capacity. The expected profit was maximized through optimal pricing and capacity decisions.

Meyr (2007) introduced two alternative clustering models to classify customer classes on the basis of customer importance and profitability. Meyr (2008) upgraded this work and applied the advanced available-to-promise concept to the lighting industry, where bulbs, fluorescent lamps, etc. are produced. These are stocked to inventory on the basis of forecasts, and sold from the inventory as soon as customer orders arrive. A deterministic linear programming model for ATP allocation and consumption was proposed. Simulation results of the proposed method were compared with conventional real-time order promising such as First-In-First-Served (FIFS) and Global Optimization (GO). In the global optimization, all customer orders are collected first and a due date is promised for the batch. The results showed that customer segmentation substantially improved profits.

Fan and Chen (2008) developed a stochastic dynamic programming model by using a revenue management technique to address the order acceptance problem for a make-to-order (MTO) industry. In the model, it was assumed that both special purpose and flexible machines existed in the MTO system. A discrete time model was used to analyze the problem, where only one type of order could be placed at a time and only one type of machine could be assigned for each accepted order. Finally, with the use of the certainty equivalence approximation, the optimal order-acceptance and capacity allocation policies were proposed. In these policies, newly arrived order would be accepted as long as its revenue was not less than the minimum of the shadow revenue of the allocated machine.

Considering stochastic demand and exogenous replenishments, Pibernik and Yasdav (2009) developed a two-step AATP model that differentiates between two customer classes under a service level constraint for a single product industry in a make-to-stock environment. The target service level was ensured by determining a safety inventory level that guarantees a certain probability of not stocking out of inventory in the lead time.

Chen and Chen (2009) developed a two-phase order promising model for an assemble-to-order manufacturing industry. The reserved capacity is defined in the 1st phase, based on forecasted demand. Customer orders are promised in the 2nd phase, based on the phase-I decision, by giving customers commitments on delivery dates and quantity. The objective of phase-I is to maximize profit by allocating resources for certain customers. The priority rule in phase-I assumes that the unit profit for a product for each customer is known. The forecast reservation programming model for phase-I is developed using a fuzzy mathematical programming approach with different degrees of customer satisfaction regarding the reserved quantities. A mixed integer linear programming approach was used for the order promising method in phase-II.

Kilic et al. (2010) addressed an order-acceptance/rejection decision problem in a food processing system. Here, a single raw material is processed to produce several products that satisfy market demand. The demand is considered stochastic in nature. The problem was modeled using two heuristic approaches called the two-band heuristic and the first-come-first-served heuristic. The two-band heuristic was based on two arguments: 1) acceptance of an order when the resource level was “sufficiently high” would always be profitable, since preservation of resources for future orders was not necessary at this resource level; and 2) acceptance of an order with small resource requirements when the resource level was “sufficiently low” would always be profitable, since it was not possible to accept future orders with higher rewards due to their larger resource requirements. In the two-band heuristic approach, an order was accepted when the resource level was in one of these two bands (i.e., higher and lower resource levels). In the FCFS heuristic, an incoming order was attended according to their sequence of arrival and an order led to a nonnegative increment in the expected revenue was accepted. In Kilic et al.'s (2010) study, compared to the FCFS heuristic, the two-band method was found to be more effective in making an optimal acceptance decision.

Azevedo et al. (2012) developed an order promising model for a make-to-stock environment focusing on the Canadian softwood lumber industry using a revenue management technique. A three-step solution technique was proposed. In the first step, potential customers are segmented based on customers price sensitiveness. Price-sensitive customers who are willing to pay around the market price are considered as customer segment 1; customers who are willing to pay lower-than-market price are considered as customer segment 2; and customers who are not sensitive to the price considered as customer segment 3. The optimal allocation of the product is computed for different customer segments over a planning horizon by solving LP problems. After defining the product allocation for each customer segment, booking limits are defined. Booking limits includes the allocated volume for a specific customer segment, the volume that is not allocated to any customer segment, and a fraction of the volume that is allocated for a different consumption week for the same customer class or from different segments for more profit. After defining the booking limit, a Mixed Integer Programming (MIP) is developed to cope with the objective of net profit maximization. The product price is determined with reference to the market price of the past week. The planning horizon is four weeks. The optimization model was simulated considering orders received during the first week. Orders arriving at any given time are assumed to present an order delivery date of within four weeks. In their study, Azevedo et al. (2012) was able to demonstrate the benefit of the proposed technique compared to FCFS by analysing different scenarios.

2.3. Sawmill Production Planning

Mendoza et al. (1991) developed an operationally feasible comprehensive production schedule for the sawmilling process. In order to achieve this, a log inventory model and a real-time hardwood process simulation model were developed and combined. The system worked by first utilizing an optimization model to determine the best log input mix to process according to the periodic lumber demand given assumed breakdowns of each log class into products. Optimization of the log mix was performed under constraints based on average machine, human, and resource capacities. The log input mix was used as an input to the simulation model to find the production schedules of the entire sawmill, and SIMAN block diagram codes were used to model the event subroutines. As the output of the simulation model could measure different sawmilling systems' performances (i.e., the volume of logs processed, sawmilling time, lumber output, resources utilization, production delay status of buffer decks, etc.), the production schedule was developed based on the information of the sawmill simulation model. After different sawmill operating modes were examined via simulation, these could be used to re-examine the production schedule. Large scale implementation of the Mendoza et al. approach has not been reported.

Maness and Norton (2002) described a multi-period production planning model which had two components: a resource coordinator and a log-sawing simulation model.

In order to find the optimal level of activities satisfying the resource and market constraints, an LP algorithm was designed as the resource coordinator. Meanwhile, the shadow price from the LP model were used in the log-sawing simulation model to generate new sawing patterns which were then reinserted in the LP to determine a better combination of activities. In contrast to the plant models, which were designed to develop production strategies for current product prices and market demands, this model considered future implications of current decisions on inventory and future sales. The model was found to respond to market changes by altering sawing patterns and log consumption. The approach of the work reported in this thesis is very similar to that of Maness and Norton. Our emphasis is on a more flexible generation of sawmill patterns and on a framework for simulating the rolling planning horizon with order promising.

A multi-period, multi-product production planning problem was addressed by Zanjani et al. (2009a) by proposing a two-stage stochastic linear program model. The objective of the proposed program was to minimize material consumption cost as well as expected inventory and backorder costs. A deterministic linear programing (LP) model for the sawmill production planning problem was first formulated when the complete information about the process yield were unknown. Thus, the production plan was considered as the first-stage (planning stage) decision variable. The second stage (plan implementation stage) generated scenarios by taking a log samples from each log class, letting them be processed by each cutting pattern, and computing the average yield for the samples. This process was repeated for all cutting patterns, and the probability of process yield for each scenario was calculated. The first stage LP model is modified by the probability of process yield and formulates a deterministic equivalent model.

Next, a Monte Carlo simulation technique was used to obtain approximate solutions. The stochastic model was then compared with the deterministic one by considering the key performance indicators of a backorder gap. These indicators include the gap between the realized total backorder size of the deterministic and the stochastic models' plans after implementation. The plan precision (that is, the gap between the planned total backorder size determined by the production planning model and the realized total backorder size after implementation of the model plan) was also taken into consideration. Overall, in terms of the realized backorder size, the production plans in sawmills provided by the two-stage stochastic model were found to be more realistic than those provided by the mean-value deterministic model. This paper again illustrates the need to use simulation to test the performance of a production planning method under uncertainty.

Zanjani et al. (2009b) upgraded their previous model by adding the uncertainty of the product demand and proposed a multi-stage stochastic programming approach. As the product demand and process yield were considered uncertain in this model, they were first modeled separately and then integrated to obtain the multi-stage stochastic programming model. During the planning horizon, demand uncertainty was considered as a dynamic stochastic process. This was modeled as a scenario tree, which is a viable way of discretizing the dynamic stochastic data over time in a population, with each stage denoting the point in time when new information is available to the decision maker. Meanwhile, the uncertain yield was modeled as a scenario with a stationary probability distribution. A hybrid scenario tree was constituted by integrating the yield scenario in each node of the demand scenario tree, which was then considered as the basis for the multi-stage stochastic programming (MSP) model. In the model, the production plan was assumed to be flexible at different stages based on the demand scenarios. The results obtained by the multi-stage stochastic programming model were compared to those obtained by the optimal solution to the mean-value deterministic. and two-stage stochastic programming models. The multi-stage approach was found to be superior among the three.

Nourelfath (2010) studied a multi-period, multi-product production planning problem. Due to the uncertainty in machine breakdowns, the production rate and the customer service level were considered as random variables. Their robust production design ensures a pre-specified customer service level with high probability. A two-step optimization model was proposed, where in the first step, the mean-value deterministic model was solved, and in the second step, a method was proposed to improve the probability of meeting the service level.

Considering the uncertainty in the process yield, Zanjani et al. (2010) developed two robust production planning models for multi-period, multi-product sawmill production. The proposed models were developed based on the variability measures of recourse cost (inventory and backorder cost) and customer service level. The upper partial moment of order 2 (UPM-2) model measures the squared positive deviation of a scenario recourse cost from the target recourse cost, and the upper partial variance (UPV) model measures the squared positive deviation of a scenario recourse cost from the expected recourse cost. The optimal expected recourse cost is determined by a two-stage stochastic model without considering the penalty on recourse cost variability. The target recourse cost is determined as a percentage of the optimal expected recourse cost and the customer service level is defined as the portion of customer demand that can be fulfilled. In the simulation model, 3 classes of logs and 5 different cutting patterns were used for 27 different products. The production planning horizon was 30 days, and production demand in each period is assumed to be deterministic based on the received order. An unsatisfied demand is penalized by a unit backorder cost, which leads to a decreased customer service level. The solution aim is to reduce any anticipated backorder costs. The RO-UPV results in less backorder /inventory cost (size) variability. These several papers by Zamani and her colleagues illustrate how one might modify a production plan to deal with uncertainty. However, they do not actually test their method in a realistic sawmilling situation.

2.4. SUMMARY

The only study we found that considers profit-driven order-promising problems for the Canadian softwood lumber industry was carried out by Azevedo et al. (2012). In their optimization model, Azevedo et al. (2012) maximize profit and allow back orders, whereas Pibernik and Yasdav (2009) optimize the target service level and do not allow back orders. Furthermore, the Azevedo et al. (2012) model considers several locations of the manufacturing unit as well as different geographical customer segments and different classes of customer based on customers' willingness to pay. This is in contrast to other studies, where customers are statistically lumped into one category of willingness to pay (Meyr, 2007, 2008; Fan & Chen, 2008: Chen & Chen, 2009) and a single-product environment (Meyr, 2008; Pibernik & Yasdav 2009). Some other research considered that capacity loading (maximizing profit through high capacity utilization and maintaining a high service level) is one of the main issues in order promising (Wang et al., 1994; Raaymakers et al., 2000b; Raaymakers & Fransoo, 2000a; Ivanescu et al., 2002) for the chemical processing industry.

Available-to-Promise (ATP) means that the products are available either in storage or can be produced in a certain future period in accordance with the production promise. ATP can be calculated as the sum of the initial inventory plus the master production scheduled (MPS) quantity, minus the back order and already promised order. However, Azevedo et al. (2012) considered the ATP in a deterministic fashion, which is significantly different from the reality of a sawmill operation. Due to the uncertain nature of the raw material (classes of logs), the production amount is uncertain for a specific production period. In this thesis, ATP is determined by a sawmill operational MIP model that is capable of reacting to production uncertainty.

The sawmill production planning and scheduling are generally determined by the best way of cutting of logs to generate certain products, while minimizing the material consumption cost and the expected inventory and backorder costs for a certain period. Several studies considered the stochastic characteristics of logs and developed combined optimization and simulation models, where all logs were required to scan through before planning and also needed to be processed in the sawmill in the same order as processed in the simulation model (Mendoza, 1991; Maness & Norton, 2002). Zanjani et al., (2009a) addressed the sawmill production planning control and scheduling problem considering uncertain yield (2009b). These models need to generate scenarios, to assess the probability of each scenario and to adjust the production schedule based on the assessed scenarios.

Capacity loading and revenue management are two different aspects addressed in the literature regarding the formulation of an order acceptance or rejection decision. To the best of the author's knowledge, no work has yet been done specifically for the lumber industry combining both aspects. This research combines the two areas of work order promising and production planning for sawmills to fill this gap.

In this research, two types of demands are considered for order-promising: contract and spot demands. Contract demands are those promised by the sawmill manager to be filled over a period of time, whereas spot demands are collected from the market one, two, or three weeks prior to the delivery date. Orders are promised based on the production planning model solution. The production planning model is developed based on the availability of the raw materials (log classes), current inventory, and resources (cutting patterns, production capacity).This research upgrades the work presented by Saadatyar (2012) by creating a pull manufacturing environment based on order promising rather than his focus on multi-level demand. The thesis also upgrades the work presented by Azevedo et al. (2012) in several ways by determining production capacity and considering spot and contract demands as well as variations of product pricing and demand for each period. The prices are not necessarily determined by customer. In terms of production planning, the recent work presented by Zanjani et al. (2010) considered 3 classes of logs and 5 different cutting patterns. These were used for 27 different products for 30-day production planning. In the work reported in the remainder of this thesis, we consider more complex problems in an attempt to come closer to realistic sawmill situations. Most of the computations we report are carried out with 2 species, 8 different classes of logs, over 6,000 cutting patterns, 41 different product of each species, with a13-week rolling planning horizon simulated over one year.

2.5. Conclusions

In this chapter. literature was reviewed on the following aspects: customer order acceptance, capacity loading issues, and sawmill production planning issues This chapter attempts to show how the current work will contribute to the existing literature. In the next chapter, the campaign generation technique will be described.

CHAPTER 3

CAMPAIGNS GENERATION

3.1. INTRODUCTION

After reviewing the literature related to the thesis goal, it appears that order promising can be an important issue contributing to industry success. The order-promising process must be based upon actual constraints (such as available raw materials and production capacity) that an industry faces for its production planning. The divergent production and the uncertain nature of logs, with the consequent uncertainty in lumber production makes order promising difficult for a sawmill manager. To fulfill the promised order and meet related deadlines, advanced production planning may be required. A key part of sawmill production planning is the optimal scheduling of campaigns to satisfy customers' lumber demands.

A campaign is defined as a combination of log classes, a set of cutting patterns, together with a pricelist which contains the relative prices of the lumber and is used to choose the cutting patterns for each class of logs. As a result of simulating the behavior of the sawmill optimizers, the campaign definition results in certain proportional outputs for the set of lumber products that can be produced from this log class. The next three sections describe the generation of log classes, pricelist, cutting patterns, and campaigns. The generated campaign will be used for sawmill production planning in the next chapter. The fourth section describes how to generate more suitable campaigns to reduce computational time.

A key issue of campaign generation is the choice of the price-list used to generate the campaign. In Sadaatayar (2012) and Sohrabi (2012), the price-lists were generated based on heuristic ideas of trying to force the production of certain widths, thicknesses and/or lengths. In this chapter, we demonstrate for the first time, at a realistic scale of more than 40 products, that column generation procedures, based on the use of shadow prices of model constraints, can give superior results in campaign generation.

3.2. GENERATION OF LOG CLASSES

A typical sawmill processes logs of many different dimensions. Logs are generated randomly using a program developed by MacDonald (2013). In her program, MacDonald considered three parameters for generating random logs: small end radius, large end radius, and length. The logs are modelled as truncated cones and no defects in log were considered. The program can deal with various specifications of logs. The following data are considered to generate logs for our simulation:

Saadatyar (2012) collected data for the small end radius of logs from Bowater Mersey Oakhill sawmill and found a satisfactory fit with a lognormal distribution. Thus, the small end radius is modeled as coming from a lognormal distribution. The mean and standard deviation of the underlying normal distribution were 1.20 and 0.32 respectively; A relation between the small and large end is established. It was assumed that the logs were uniformly tapered, and that the taper rate is uniformly distributed in between 0.05 and 02 inches per feet of a log. Thus the large end radius for a log can be calculated as: the large end radius = small radius + uniform (5, 20)/100 (in/ft.)× length (ft.). Furthermore, using the Bowaters data, the probability of generating log lengths greater than 8ft. and less than 10ft. is 0.01; greater than 10ft. and less than 12ft. is 0.12; greater than 12ft. and less than 14ft. is 0.23; greater than 14ft. and less than16ft. is 0.13; and greater than 16ft. and less than18ft. is 0.49. Based on random small-end radius, length and taper 100,000 logs were generated. Different classes were created by sorting these 100,000 logs by length length. Log lengths less than 10ft. are considered class1; random lengths greater than 10ft. and less than 18ft. are class2; lengths greater than 10ft and less than 12ft. are class3; lengths greater than 12ft. and less than or equals 14ft are class 4; lengths greater than 14ft. and less than or equals 16ft. are class 5; and lengths greater than 16ft. and less than or equals 18ft. are class6. Log classes from 1 to 6 are used for pine and the log classes 1 and 2 are used for spruce.

These eight classes were used to test the approach in this thesis. It is possible to generate many more classes by sorting according to diameter as well as length. This will be left for future research. It should be noted that most sawmills will have extensive scan information on their log inputs. In this case it is unnecessary to simulate logs. The scanned information can be used correctly. The only reason that simulated log classes were used for this basis was due to the lack of real data from sawmills.

3.3. GENERATION OF THE PRICELISTS

A sawmill operation is controlled through the pricelist, a list that contains the relative prices of each lumber piece type. The pricelist plays a vital role in selecting cutting patterns. Typically, the sawmill is automated with various scanners and computers that decide which cutting pattern to use. The pattern chosen for an individual log is that which maximizes the value of the lumber produced where the price of each individual lumber product is given in the price list.

Saadatyar (2012) generated 20 different pricelists. Pricelist 1 was developed using actual lumber prices collected from http://www.acehardware.net/estimate/ in his research. The prices of products that are absent on that website were generated by developing a fit function using the “least squares” method. Pricelist 2 was generated by considering all types of lumber to have the same unit price. . Pricelist 3 created a function which generated unit price by emphasizing width; pricelist 4 considered used a function for unit price by emphasizing thickness; pricelist5 used a function for unit price, emphasizing length; and pricelists 6 to 20 repeated the emphasis on width, thickness and length by multiplying the unit price by 20 (assumed) for all generated products. Pricelist 6-9 generated giving emphasis on width, 10-15 on thickness and 15-20 on length. The details of these pricelists are given in Saadatyar (2012).

In section 3.6, we describe how more price list can be generated using the shadow prices of the LP model and these then used to generate more suitable campaigns.

3.4. GENERATION OF ALL POSSIBLE CUTTING PATTERNS

A new algorithm was developed by Sohrabi (2012) and Saadatyar (2012) by considering in advance the capability of a softwood sawmill to produce many combinations of patterns for a log of a particular diameter. MacDonald (2013) has improved their algorithm and documented the resulting Python code. The details can be found in Chapter 3 of Saadatyar’s (2012) MASc thesis, but a brief description is provided here. Patterns are generated in two or three steps of cutting (main, first, and second cut). Fig. 3.1 shows the schematic diagram of generation of all possible cutting patterns.

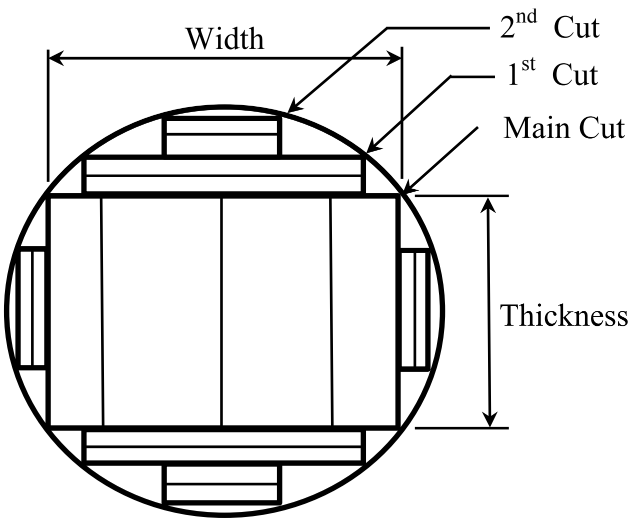

Figure 3.1: Schematic diagram of cutting pattern generation

The thickness of the main cut is determined by one standard thickness of lumber, and the maximum total width is determined by multiplying the standard thickness by a factor. All possible combinations of patterns are generated from the standard widths at that thickness of lumber, using kerf (width of the saw) and allowance (difference between "target" and "actual", within each dimension of the lumber). After the diameter of the main cut is determined, the largest available uncut portion remaining within that diameter is cut in the next step. The largest remaining uncut portion is sectioned into all possible combinations of standard-sized lumber using an algorithm similar to that for the main cut in the 1st cut. If any available area remains, that is cut using the similar algorithm in 2nd cut. Details of the process are given in the thesis by Saadatyar (2012) and Sohrabi (2012), and in MacDonald et al. (2013). Typically the algorithm produces 6000+ candidate sawing patterns. Not all patterns can be applied to a given log. For the subset of patterns which can be used, when that pattern is applied to the given log, the diameter, taper and length of the log are used to to calculate the lengths of the pieces that will be produced, given the wane allowances for the lumber.

3.5. GENERATION OF CAMPAIGNS

After generating the log classes, pricelist and cutting patterns, the next step is to generate campaigns. Each log in a sawmill is processed by a cutting pattern, the best of which, in terms of value output, is chosen by a pricelist. The campaign throughput is the cumulative throughput of lumber from a log class that includes the number of lumber pieces produced of each type, volume throughput in terms of percentage, and total lumber value.

MacDonald's updated campaign generation model, an extension of that developed by Sohrabi (2012) and Saadatyar (2012), can be described in the following steps:

Step 0: Setup input data files: The setup input data file contains all of the nominal values of the lumber to be produced, i.e., width (in), thickness (in), length (ft).

Step 1: Create the price lists: Use the pricelist that is described in section 3.3.

Step 2: Create the patterns: The generation of cutting patterns is described in section 3.4.

Step 3: Create the logs: The generation of logs is described in section 3.2.

Step 4: Find the best cutting option for each log: The best cutting option is chosen based on the value of produced lumber based on the pricelist.

Step 5: Generate the campaign: The generated campaign and relevant statistics, such as nominal lumber dimension, volume throughput, and number of logs processed, are stored in an Excel spreadsheet. Fig. 3.2 shows the campaign generation flow steps.

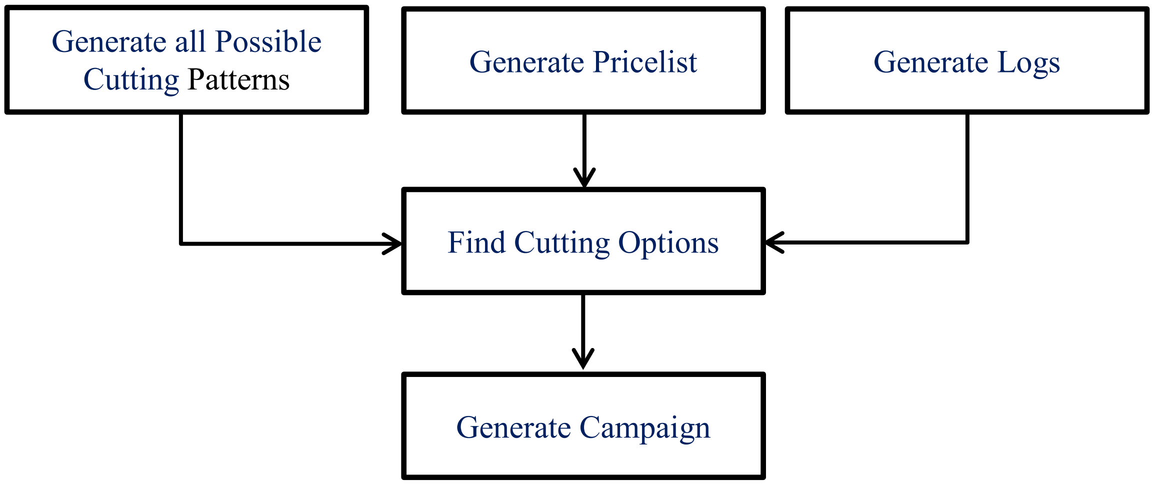

Figure 3.2: Campaign generation flow diagram (derived from MacDonald et al.'s (2013) developed program)

3.6. NEW CAMPAIGN GENERATION

At this point in our research, we are interested in reducing the solution time of Saadatyar's (2012) campaign scheduling model. This is because we want to develop an order-promising and production-planning model for 52 weeks instead of a 13-week model. In the literature, Maness and Norton (2002) demonstrated that a shadow price-generated cutting pattern introduced new columns into their master LP model and better met the market demand. A shadow price-driven cutting pattern can be more responsive to market demand than any other type of price. The shadow prices are generated after solving an LP model and represent the amount of additional value added to the objective value per unit variation of appropriate constraints. Maness and Norton's (2002) investigation drives us to generate a more suitable campaign by generating new pricelists using a shadow price technique that will introduce new columns in the campaign scheduling model and allow us to find a faster solution for the same market demand. Saadatyar's (2012) 13 weeks campaign schedule model is used to generate the shadow price of all products in this thesis. Fig. 3.3 and Fig 3.4 describe the mathematical model of Saadatyar (2012).

Sets

I	Set of product types (i_c is the index to the chip product)
S	Set of product species
K	Set of campaigns
T	Set of time periods
C	Set of classes
KC	Set of campaign class combinations
L	Set of market levels

Parameters

v_{ist}^{l}	$i \in I, s \in S, l \in L, t \in T$	Product selling price (\$/ft^3)
N_{ist}^{l}	$i \in I, s \in S, l \in L, t \in T$	Amount of product that can be sold (ft^3)
h_{ist}	$i \in I, s \in S, t \in T$	Inventory holding cost (\$)
O_{isk}	$i \in I, s \in S, k \in K$	Output percentage from each campaign run (percentage)
V_{tk}	$t \in T, k \in K$	The log volume input rate (ft^3 per period)
I_{ist}^{Min}	$i \in I, s \in S, t \in T$	The minimum allowable inventory (ft^3)
S_{ist}^{Max}	$i \in I, s \in S, t \in T$	The maximum allowable sales (ft^3)
S_{ist}^{Min}	$i \in I, s \in S, t \in T$	The minimum allowable sales (ft^3)
LC_k	$k \in K$	Cost of the logs (\$/tonne)
ST_k	$k \in K$	Campaign setup time (week)
STC_c	$c \in C$	Class setup time (week)
α		Conversion factor from volume to log

	weight (tonne/ft^3)
γ	Penalty cost for violating minimum inventory constraint (\$/ft^3)
$ICAP$	The maximum Sawmill storage (ft^3)

Variables

X_{ist}^{l}	$i \in I, s \in S, l \in L, t \in T$	Amount of product sold (ft^3)
I_{ist}	$i \in I, s \in S, t \in T$	Inventory of product (ft^3)
$\boldsymbol{\varepsilon_{ist}}$	$i \in I, s \in S, t \in T$	Violation of the inventory balance constraints (ft^3)
λ_{kt}	$k \in K, t \in T$	Campaign run time (week)
P_{ist}	$i \in I, s \in S, t \in T$	Production (ft^3)
$toChips_{ist}$	$i \in I, s \in S, t \in T$	Amount of chips produce from lumber inventory (ft^3)
Y_{kt}	$k \in K, t \in T$	Binary variable for campaign (0 if campaign k is run in t; 1 otherwise)
Z_{ct}	$c \in C, t \in T$	Binary variable for class (0 if class c is run in t; 1 otherwise)

Figure 3.3: Model MIP formulation: Sets, parameters and variables (Saadatyar, 2012)

Objective function:

$$Max \sum_{i}\sum_{s}\sum_{l}\sum_{t}((X_{ist}^{l} * v_{ist}^{l}) - (h_{ist} * I_{ist}) - (\gamma * \varepsilon_{ist})) - (\sum_{k}\sum_{t}((\lambda_{kt} * V_{kt})) * \boldsymbol{\alpha} * LC_{k}) \qquad 3.1$$

Subject to:

$$I_{ist} = I_{is(t-1)} + P_{ist} - \left(\sum_{l} X_{ist}^{l}\right) - toChips_{ist} \qquad \forall\, i \neq i_c, s, t \qquad 3.2$$

$$I_{cst} = I_{cs(t-1)} + P_{cst} - \left(\sum_{l} X_{cst}^{l}\right) + \sum_{i} toChips_{ist} \qquad \forall\, i = i_c, s, t \qquad 3.3$$

$$P_{ist} = \textstyle\sum_k \lambda_{kt} * (O_{isk} * V_{tk}) \qquad \forall\, i, s, t \qquad 3.4$$

$$\sum_{k \epsilon K} (\lambda_{kt} + Y_{kt} * ST_k) + \sum_{c} (Z_{ct} * STC_c) \leq 1 \qquad \forall\, t \qquad 3.5$$

$$S_{ist}^{Min} \leq \sum_{l} X_{ist}^{l} \leq S_{ist}^{Max} \qquad \forall\, i, s, t \qquad 3.6$$

$$I_{ist}^{Min} - \boldsymbol{\varepsilon_{ist}} \leq I_{ist} \qquad \forall\, i, s, t \qquad 3.7$$

$$X_{ist}^{l} \leq N_{ist}^{l} \qquad \forall\, i, s, l, t \qquad 3.8$$

$$\lambda_{kt} \leq Y_{kt} \qquad \forall\, k, t \qquad 3.9$$

$$Y_{kt} \leq Z_{ct} \qquad \forall\, k, c, t, (k, c) \in KC \qquad 3.10$$

$$\sum_{i} \sum_{s} I_{ist} \leq ICAP \qquad \forall\, i, s, t \qquad 3.11$$

$$I_{i,s,0} \leq I_{i,s,end} \qquad \forall\, i, s, t = 0/end \qquad 3.12$$

$$\sum_{k:(k,c) \in KC} Y_{kt} \geq Z_{ct} \qquad \forall\, c, t, \qquad 3.13$$

$$\sum_{c} Z_{ct} \geq 1 \qquad \forall\, t \qquad 3.14$$

$$\lambda_{kt} \geq Y_{kt} * ST_{kt} \qquad \forall\, k, t \qquad 3.15$$

$$\sum_{k:(k,c) \in KC} \lambda_{kt} \geq STC_c * Z_{ct} \qquad \forall\, c, t \qquad 3.16$$

$$0 \leq P_{ist}, \qquad 0 \leq X_{ist}^{l} \qquad 3.17$$

$$0 \leq \lambda_{kt} \leq 1 \qquad 3.18$$

Figure 3.4: Model MIP formulation: Objective and constraints (Saadatyar, 2012)

The objective function of Saadatyar's (2012) model shown in Equation 3.1 maximizes the total revenue, which includes the total revenue from selling products at different market level prices minus the product's holding cost, minus the minimum inventory violation cost, and minus the log costs for all periods.

Constraint 3.2 describes the inventory balance equations for all dimensioned lumbers. The left side of the equation is the current inventory, and the right side of the equation is the previous inventory, plus current production, minus current sales, and minus the converted amount of chips from the lumber inventory.

Constraint 3.3 is for the chips inventory balance equation. The current chip inventory is equal to the previous chip inventory, plus the current period chip production, minus chip sales, and plus the sum of chips converted from the different lumber inventories.

Constraint 3.4 represents the total production amount. The total production is the campaign run-time, percentage throughput of each product from that campaign, and the log input rate of that campaign.

Constraint 3.5 indicates that the sum of the campaign run-time, campaign setup time, and class setup time is less than or equal to 1 week.

Constraint 3.6 limits the upper and lower bounds of sales.

Constraint 3.7 sets the lower bound of the inventory level. There is a penalty cost if the inventory goes below the lower bound.

Constraint 3.8 limits the sales to market level.

Constraint 3.9 states that there is a setup time before running a campaign.

Constraint 3.10 shows that all campaigns in a class will run first before changing the class.

The total inventories are limited by constraint 3.11. The initial inventory and the final inventory are the same, as shown in constraint 3.12. Constraint 3.12 ensures the model's activity.

Constraint 3.13 ensures that before setting a new class, all campaigns of that class complete the processing. Constraint 3.14 ensures that at least campaigns from one of the classes of logs will run. Constraint 3.15 sets the lower bound of the campaign run-time. A campaign run-time must be greater than or equal to the campaign setup time. Constraint 3.16 sets the lower bound of the class run-time. The run-time of a class must be greater than or equal to the class setup time. Constraint 3.17 is for the non-negativity of the variables, and constraint 3.18 shows that each campaign run-time is less than a period (week).

For the purpose of this thesis Saadatyar's (2012) campaign scheduling MIP model has been simplified to 41 different dimensioned lumber products using for two different log species (spruce and pine), 8 different log classes, 20 price list and 160 campaigns. A list of these lumber products is shown in Table 4.1. The model was formulated for 13 weeks and solved using the Gurobi 5.5 solver. The solution of the model is as follows:

Best objective value: $\$ 9.43\times10^6$

Solution time: 12758 sec

MIP gap: 1.96%

The obtained solution is 1.96% close to best potential solution for MIP. The shadow price of the product can only be found after finding the solution of a linear programming

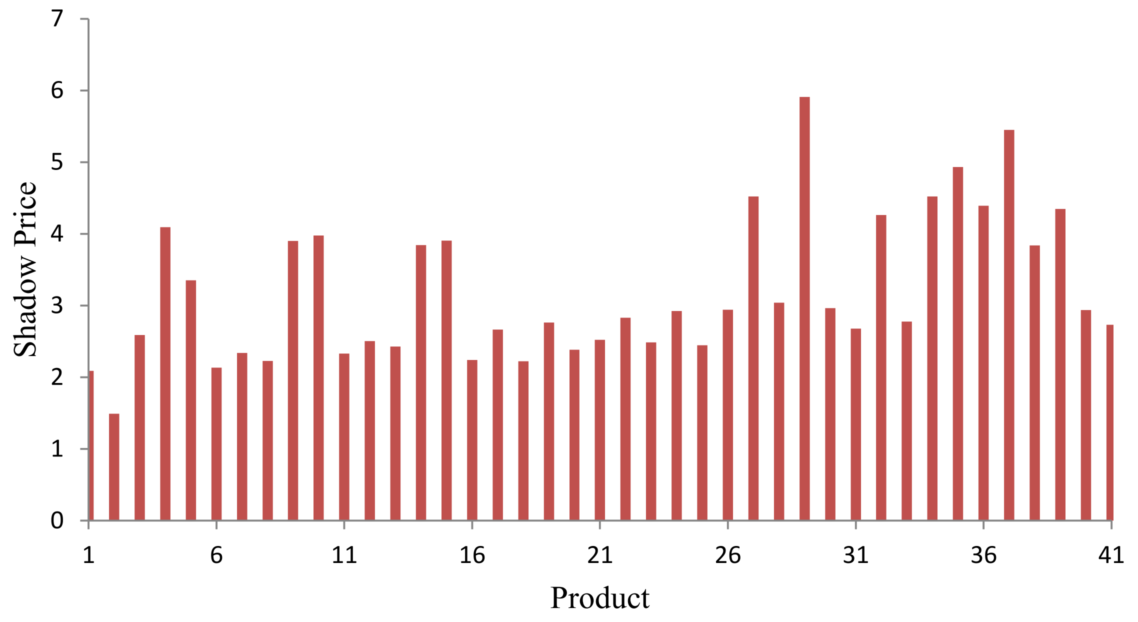

Figure 3.5: Shadow price generated from 160 campaigns for period 1

model. Shadow prices have no meaning for an MIP model, In our case, after solving approximately the MIP model, we fixed the integer value and solved the model as a linear programming model. The shadow prices of the product were taken from the inventory balance constraints 3.2 and 3.3.

The constraints 3.2 produced shadow price for 40 different dimensioned lumber and the constraints 3.3 produced the shadow price of the chips. Each period of solution produced a set of shadow prices for that period. Fig. 3.5 shows the generated shadow price of 41 different products for period 1. Thus, the 13-period solution produced 13 sets of new prices. These 13 sets of new lumber prices were added to the existing 20 sets of prices (used to generate 160 campaign from 8 log classes).

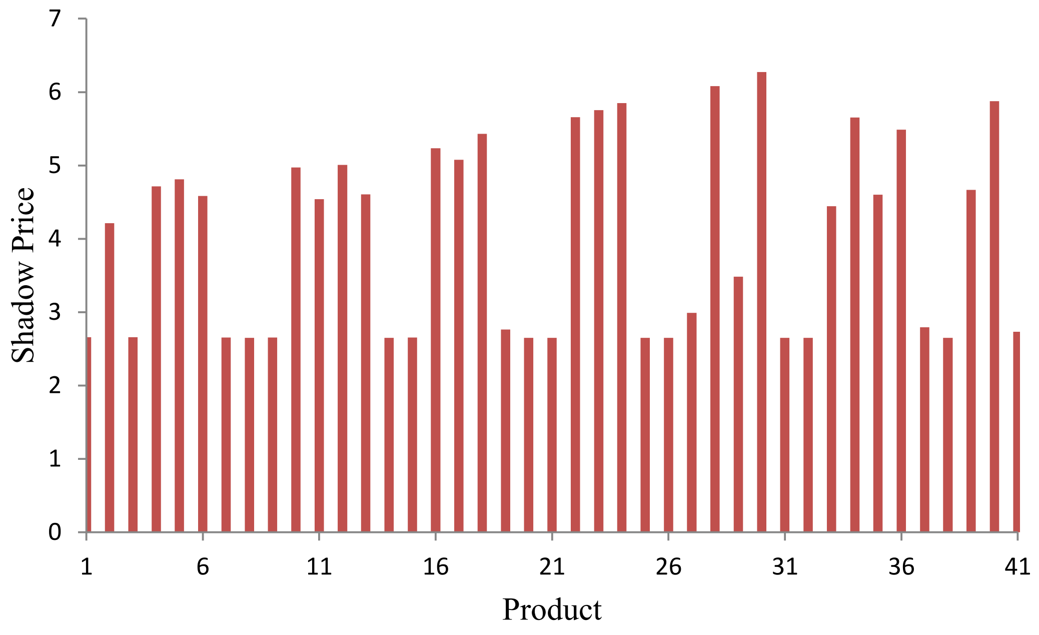

Figure 3.6: Shadow price generated from 238 campaigns for period 1

In MacDonald's campaign generation module, 20 set of pricelists are replaced by 33 set of pricelists and produced new campaigns. Using these 33 pricelists, the module produced 238 different campaigns for 8 classes of logs. The campaign scheduling model was solved again using 238 different campaigns. The solution findings are below using Gurobi 5.5 solver:

Best objective value: $ 9.26\times10^6$

Solution time: 9187 sec

MIP gap: 1.96%

Note that the solution time has decreased, even though we now have many more campaigns. We then repeated the same procedure to find a new set of pricelists. From the new solution run, we obtained 13 more pricelists, one of which is shown in Fig. 3.6.

Now, these 13 sets of pricelists are added with the previous 33 set of pricelists. This total of 46 different pricelists from 8 different classes of logs produces 366 different campaigns. The Gurobi 5.5 solver found the following solution after solving the 13-period production planning problem.

The solution findings are below:

Best Objective value: \$ 9.43×10^{6}

Solution time: 7684 sec

MIP Gap: 1.95%

3.7. Conclusions

Campaign generation techniques were described in this chapter, along with how the number of campaigns affects the solution time. Generating a new campaign based on the shadow price introduces new columns in the campaign scheduling model in a way similar to that Maness and Norton (2002). By providing more and better options to solve the schedule campaign and meet the market demands, this also reduces computational time. The solution shows that a higher number of campaigns results in a shorter computational time. It was demonstrated that increasing the number of campaigns from 180 to 366 reduces the solution time from to 12,758sec to 7,684sec. The shadow price generation code written in python 2.7 for Gurobi solver is given in Appendix A. The mathematical model of order promising and production using 366 campaigns to solve the problem is shown in the next chapter. Additional work could have been done in determining how many times the shadow price procedure should be run. This will be left for future research.

CHAPTER 4

MATHEMATICAL MODEL

4.1. INTRODUCTION

Order promising and delivering the promised order on time are important issues in the success of any manufacturer. Moreover, the order promising process must be based upon actual constraints (such as raw materials and capacity) that an industry faces for its production planning.

In the case of the lumber industry, the problem is greatly complicated by the divergent nature of sawmill production where a given production campaign will produce many different products as well as the additional problem due to the uncertain nature of logs and consequent uncertainty in lumber production. In this thesis we focus on the issues of divergent production and demand and price and uncertainty. The uncertainty in lumber yields and production can be dealt with via similar methods to those we will discuss.

Capacity constraints have been the focus of numerous studies in the literature (Raaymakers and Fransoo, 2000a; Ivanescu et al., 2002). Order promising without considering production capacity usually results in dissatisfied customers. This thesis considered the sawmill capacity constraints and divergent production in the process of promising an order. The problem description and the model formulation are described in the next two sections, and section 4.4 provides relevant data used in Chapter 5 for solving the mathematical model.

4.2. METHODOLOGY

The objective of this work is to maximize the revenue through order promising that takes into account the available resources and ensures that the resources are used optimally. In our study, we consider an example sawmill with an output capacity of 80 million board feet per year. The mill processes two different species (i.e., spruce, pine) of logs and produces 41 different products from each species, lumber of different dimensions, as well as chips. For simplicity we consider only a single lumber grade although in reality there will be several different quality grades. The produced lumber and chips are assumed to be sold on the market under two different situations. The first is to that of corporate customers with a large volume of demand and known, committed delivery contracts. The revenues from this contract demand are known. The second situation involves spot demand, where the mill experiences potential orders each week with delivery lead times of 0, 1, 2, 3, … L weeks and a price for delivery at that time for each product. In the work here we assume that the prices are higher for short lead times but in later work this assumption can be replaced.

A static MIP model has been formulated for order promising and production planning for the sawmill in this chapter. A summary of the input and output flow of the model is shown in Fig.4.1. The solution model is formulated in two steps. In the first step, all campaigns are generated based on pricelists, cutting patterns, and log classes. Descriptions for these have been given in Chapter 3. In the second step, the integrated order promising and campaign scheduling model are developed that include all of the generated campaigns, available inventory, inventory holding cost, promised demands and log price.

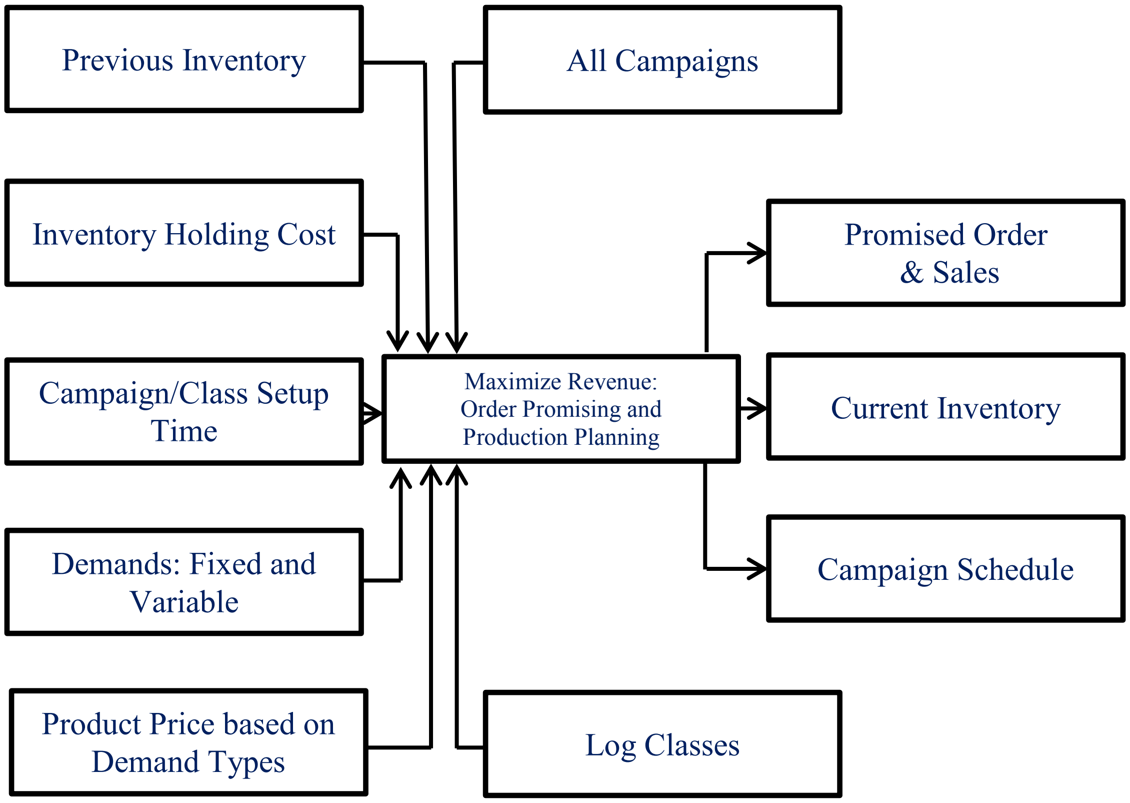
Previous Inventory
All Campaigns
Inventory Holding Cost
Promised Order & Sales
Campaign/Class Setup Time
Maximize Revenue: Order Promising and Production Planning
Current Inventory
Demands: Fixed and Variable
Campaign Schedule
Product Price based on Demand Types
Log Classes

Figure 4.1: Model input and output diagram

In the campaign scheduling model, two types of setup time are considered: campaign setup and class setup time. The mathematical model formulated in the next section is based on the following assumptions, all of which can be altered.

- A period is considered, as one week (8 hours a day, 5 days a week)
- The model is formulated for 52 periods. The operation hours for every period are same.
- No down-time is considered.
- Campaign outputs are based on simple log models (perfect truncated cones) and all products are the same grade.

4.3. Model Formulation

An MIP model is formulated by extending Saadatyar's (2012) model to solve the problem described in section 4.2. The output of the model maximizes the revenue through order promising and also creates a production plan through campaign scheduling to satisfy the promised orders on time. Fig.4.2 shows the Sets, Parameters and Variables of the model and Fig4.3 shows the model's objective and constraints. The GNU linear programing kit (GLPK) formulation for this model is found in Appendix B. The outputs and demands are in units of cubic ft. nominal . One can divide by 12 to get board ft. nominal.

Sets

I	Set of products (ic is the index to the chip product)
J	Set of species
K	Set of campaigns
T	Set of periods
C	Set of classes
KC	Set of all (k, c) where k is a campaign using log class c
L	Set of potential orders lead time

Parameters

h_{ijt}	$i \in I, j \in J, t \in T$	Inventory holding cost (\$/ft^3)
d_{ijt}^F	$i \in I, j \in J, t \in T$	Contract demands over the period (ft^3)
d_{ijt}^l	$i \in I, j \in J, t \in T, l \in L$	Potential orders to be delivered in l weeks (ft^3)
v_{ijt}^F	$i \in I, j \in J, t \in T$	Product price for contract demands (\$/ft^3)
v_{ijt}^l	$i \in I, j \in J, t \in T, l \in L$	Product price for l week advanced submitted demands (\$/ft^3)

Log_k	$k \in K$	Log cost (\$/tonne)
Set_k	$k \in K$	Campaign setup time as a fraction of one period (week)
Set_c	$c \in C$	Class setup time as a function of one period (week)
V_{tk}	$t \in T, k \in K$	Log volume input rate (ft^3 per period t)
$I_{i,j,0}$	$i \in I, j \in J$	Initial inventory (ft^3)
B_{ij0}	$i \in I, j \in J$	Initial Backorder (ft^3)
O_{ijk}	$i \in I, j \in J, k \in K$	Output of each campaign (lumber volume/log volume)
$iniv_{ij}$	$i \in I, j \in J$	Initial inventory product I, species j (ft^3)
$Liminv$		Inventory holding limit (ft^3)
α		Log volume to weight conversion factor (tonne/ft^3)
γ		Backorder penalty cost (\$/ft^3)
n		Number of products

Variables

I_{ijt}	$i \in I, j \in J, t \in T$	Inventory (ft^3)
λ_{kt}	$k \in K, t \in T$	Campaign run time (week)
P_{ijt}	$i \in I, j \in J, t \in T$	Production of lumbers (ft^3)
$ToCh_{ijt}$	$i \in I, j \in J, t \in T$	Production of chips form lumber inventory (ft^3)
S^F_{ijt}	$i \in I, j \in J, t \in T$	Contract promised order (ft^3)
S^l_{ijt}	$i \in I, j \in J, t \in T, l \in L$	Order promised to be delivered in l weeks (ft^3)
$Sale_{ijt}$	$i \in I, j \in J, t \in T$	Total Sales on promised order (ft^3)
B_{ijt}	$i \in I, j \in J, t \in T$	Backorder (ft^3)

Y_{kt}	$k \in K, t \in T$	Binary variable for campaign (0 if campaign k is run in t; 1 otherwise)
Z_{ct}	$c \in C, t \in T$	Binary variable for class (0 if class c is run in t; 1 otherwise)

Figure 4.2: Model MIP formulation: Sets, parameters and variables

Objective

$$Max \sum_{d \in F,0,2,3} \sum_{i} \sum_{j} \sum_{t} \left(\left(S^{d}_{ijt} * v^{d}_{ijt} \right) - \left(h_{ijt} * I_{ijt} \right) - \left(\gamma * B_{ijt} \right) \right) - \sum_{k} \sum_{t} \left(\lambda_{kt} * V_{kt} * \alpha * Log_{k} \right) \tag{4.1}$$

Constraints

$$p_{ijt} = \sum_{k \in K} \lambda_{kt} O_{ijk} V_{tk} \qquad \forall\, i,j,t \tag{4.2}$$

$$I_{ijt} - B_{ijt} = I_{ij(t-1)} - B_{ij(t-1)} + P_{ijt} - Sale_{ijt} - ToCh_{ijt} \qquad \forall\, i \neq ic, j, t \tag{4.3}$$

$$I_{ijt} - B_{ijt} = I_{ij(t-1)} - B_{ij(t-1)} + P_{ijt} - Sale_{ijt} + \sum_{i=1}^{n-1} ToCh_{ijt} \qquad \forall\, i = ic, j, t \tag{4.4}$$

$$\sum_{k,(k,c) \epsilon KC} \left(\lambda_{kt} + Y_{kt} * Set_{k} \right) + \sum_{c} \left(Z_{ct} * Set_{c} \right) \leq 1 \qquad \forall\, t \tag{4.5}$$

$$S^{F}_{ijt} = d^{F}_{ijt} \qquad \forall\, i,j,t \tag{4.6}$$

$$S^{l}_{ijt} \leq d^{l}_{ijt} \qquad \forall\, i,j,t \tag{4.7}$$

$$Sale_{ijt} = S^{F}_{ijt} + \sum_{l} S^{l}_{ij(t-l)} \qquad \forall\, i,j,t \tag{4.8}$$

$$\lambda_{kt} \leq Y_{kt} \qquad \forall\, k,t \tag{4.9}$$

$$\lambda_{kt} \geq Y_{kt} * Set_{k} \qquad \forall k,t \tag{4.10}$$

$Y_{kt} \leq Z_{ct}$ $\quad \forall k, c, t, (k, c) \in KC$ 4.11

$\sum_{k,c\epsilon KC} \lambda_{kt} \geq Set_c * Z_{ct}$ $\quad \forall\, c, t$ 4.12

$\sum_{k,c\epsilon KC} Y_{kt} \geq \; Z_{ct}$ $\quad \forall k, c, t, (k, c) \in KC$ 4.13

$\sum_{c} Z_{ct} \geq 1$ $\quad \forall\, t$ 4.14

$\sum_{i}\sum_{j} I_{ist} \leq Liminv$ $\quad \forall\, t$ 4.15

$B_{ijt} = 0$ $\quad \forall\, i, j, t = 0$ 4.16

$I_{ij0}= 0$ $\quad \forall\, j, i = ic$ 4.17

$I_{i,j,0} = iniv_{ij}$ $\quad \forall\, i \neq ic, j$ 4.18

$I_{i,j,end} \geq I_{i,j,0}$ $\quad \forall\, i, j$ 4.19

$\sum_{k} Y_{kt} \leq 46$ $\quad \forall\, t$ 4.20

$\sum_{c} Z_{ct} \leq 8$ $\quad \forall\, t$ 4.21

$0 \; \leq \lambda_{kt} \leq 1$ $\quad \forall\, k, t$ 4.22

$I_{ijt},\, B_{ijt},\, p_{ijt},\, S^{\,F}_{ijt},\, S^{0}_{ijt},\, S^{2}_{ijt},\, S^{3}_{ijt}, Sale_{ijt} \geq 0$ $\quad \forall\, i, j, t$ 4.23

Figure 4.3: Model MIP formulation: Objective and constraints

The objective function (Equation 4.1) maximizes the total revenue over the planning horizon and contains four terms. The 1st term sums all revenues generated from selling the products. The 2nd term is the sum of all inventories' holding costs. The 3rd term is the sum of all backorder costs and the 4th term is the sum of log costs, which is calculated by the log price per unit volume, multiplied by the total volume of logs consumed.

Constraint 4.2 describes the total production of each product in each period. The right hand sums over all campaigns k. The λ_{kt} is the time (fraction of a week) that campaign k runs. The V_{tk} is the input feed rate (tonnes per week) for campaign k in period t) and the O_{ijk} is the output of product i, species j per tonne of input of campaign k.

Inventory balance constraints are shown in Constraints 4.3 and 4.4. The left-hand side of the inventory balanced constraint consists of current inventory and back orders. Constraint 4.3 is for the entire lumber inventory, meaning for products numbered 1 to 40. The right-hand side of Constraint 4.3 shows previous inventory, previous backlog, current production, current sales, and chips produced while processing logs. The back order is deducted from the both of the sides so that the constraints hold for any positive value of the inventory and back order. That means that neither inventory nor backorder will ever be negative.

Constraint 4.4 is for the inventory balance constraint for chips. The left hand side of the constraint is the current inventories and backorder of the chips. The right hand side contains previous period inventories, backorders, current production, sales and the chips produced while making other products.

The unit of production time is a week. The total operation hours a week is 40 hours assumed to be same for every period. Constraint 4.5 ensures the total run-time and set-up time of a campaign and that classes of log should not exceed one week.

Constraint 4.6 shows that the amount promised to contract demand each period is equal to the contract demand.

Constraints 4.7 states that promised orders for delivery within any lead time should be less than or equal to the demand (known or forecasted) for that product and order lead time.

Constraint 4.8 states that total actual deliveries to sales in a period is the sum over l of the amounts promised l periods ago with a lead time of l.

Constraints 4.9 and 4.10 ensure that a campaign cannot run unless it has incurred a setup and that if run, it must be run for a time that is at least as long as setup. This lower limit can be increased if desired.

Constraint 4.11 requires that the log class c to which campaign k belongs must be set up if campaign k is setup.

Constraint 4.12 requires that the total campaign run-time in a class should be greater than or equal to the set-up time of the class.

Constraint 4.13 states that class set-up time will not be considered as long as a campaign of a class is running.

Constraint 4.14 ensures that at least one class of log is running. Constraint 4.15 shows that the maximum amount of inventory is limited by an inventory limit.

Constraint 4.16, 4.17 and 4.18 are for the initial back order, initial chip inventory, and initial lumber inventory, respectively.

Constraint 4.19 is a somewhat artificial constraint that requires total inventory in storage at the end of the horizon is equal to that at the beginning. This is an attempt at eliminating “free production” for testing purposes. In a real application, more realistic ending inventory constraints can be established for each product.

Constraints 4.20 and 4.21 are the limit of the number of campaigns and log classes per period. Constraints 4.22 ensure that campaign run time should not be more than a period. Constraints 4.23 are for non-negativity for the variables.

The model developed in this study is a combination of order promising and production planning for sawmills, which is an extension of Saadatyar’s (2012) production planning model. The model attempts to maximize revenue through order acceptance and campaign scheduling. A penalty cost is used for backorder.

The constraints 4.3 and 4.4 describe the inventory balance equations for the lumber products and the chips, respectively. The backorders are deducted from both sides of the two constraints (i.e., 4.3 and 4.4) so that the inventories and backorders never take negative values. Constraints 4.6, 4.7 and 4.8 are added to Saadatyar’s (2012) model to include order promising options. Constraint 4.5 ensures that the promised orders for the contract demand are equal to the contract demand. Constraint 4.6 represents that the promised orders for any lead time for a product is less than or equal to the demand of that product of that lead time. Constraint 4.8 ensures that the current sales should equal to the sum of promised order (i.e., contract and spot-demands) for the current period. The overall model promises order considering the availability of the resources (raw materials and production facilities) and ensures the best use of them.

4.4. DATA

The data related to campaign scheduling in the model such as sawmill capacity, species, log classes, set-up time, log cost, lumber price, etc., are taken from Saadatyar's (2012) MASc thesis. The solution technique is described in the next chapter.

Capacity: 80 M fbm (80,000,000) board feet of lumber are considered as the annual capacity of the modeled sawmill.

Weekly inventory holding cost: The inventory holding cost is the sum of fixed and variable costs. It is assumed that the fixed cost is \$1/ year and the variable cost is 25% of contract/fixed demand product. This can be calculated as

$$h_{ijt} = (v_{ijt}^{F} \times 0.25 \times \frac{1}{52}) + \frac{1}{52}, \quad \forall\, i, j, t$$

The inventory holding cost is given in Table 4.1 for each product.

Table 4.1: Product data (Saadatyar, 2012)

Pro-duct	Dimensions (WxTxL) (in*in*ft)	Average Demand $ft^3/week$	Unit Price (\$/ft^3)	Holding Cost (\$/ft^3) / week	Product	Dimensions (WxTxL) (in*in*ft)	Average Demand $ft^3/week$	Unit Price (\$/ft^3)	Holding Cost (\$/ft^3) /week
1	1x3x8	3449.388	4.422433	0.040492	21	2x6x8	3492.103	5.74146	0.046834
2	1x3x10	2073.339	4.519666	0.04096	22	2x6x10	3322.807	5.838193	0.047299
3	1x3x12	799.2673	4.616899	0.041427	23	2x6x12	3927.914	5.934925	0.047764
4	1x3x14	399.5905	4.714132	0.041895	24	2x6x14	4290.089	6.031658	0.048229
5	1x3x16	235.4637	4.811365	0.042362	25	2x6x16	4598.379	6.12839	0.048694
6	1x4x8	6046.227	4.58502	0.041274	26	2x8x8	1212.677	5.99479	0.048052
7	1x4x10	3616.337	4.682198	0.041741	27	2x8x10	1413.427	6.0913	0.048516
8	1x4x12	2400.034	4.779375	0.042209	28	2x8x12	1887.214	6.18781	0.04898
9	1x4x14	1232.175	4.876553	0.042676	29	2x8x14	2096.009	6.284321	0.049444
10	1x4x16	955.7446	4.97373	0.043143	30	2x8x16	2415.516	6.380831	0.049908
11	2x3x8	6879.644	5.361465	0.045007	31	2x10x8	731.84	6.24812	0.04927
12	2x3x10	2873.813	5.458531	0.045474	32	2x10x10	793.945	6.344408	0.049733
13	2x3x12	1760.281	5.555598	0.04594	33	2x10x12	947.0752	6.440696	0.050196

14	2x3x14	859.7346	5.652664	0.046407	34	2x10x14	890.4678	6.536983	0.050659
15	2x3x16	811.5066	5.74973	0.046874	35	2x10x16	966.5798	6.633271	0.051121
16	2x4x8	10924.27	5.48813	0.045616	36	2x12x8	367.9265	6.50145	0.050488
17	2x4x10	7921.274	5.585085	0.046082	37	2x12x10	447.0633	6.597516	0.05095
18	2x4x12	4771.97	5.68204	0.046548	38	2x12x12	601.3857	6.693581	0.051411
19	2x4x14	2605.536	5.778995	0.047014	39	2x12x14	595.679	6.789646	0.051873
20	2x4x16	2553.478	5.87595	0.047481	40	2x12x16	697.56	6.885711	0.052335

The contract/fixed demand is 30% of the average weekly demand shown in Table 4.1. Demands to be delivered within one, two, or three, weeks is 25% of the average weekly demand that is shown in Table 4.1 for each of the scenario.

In this example problem, we consider only orders promised for deliveries in the current week (l=0) or with two or three weeks of lead time (l=2,3). Product price for contract demands is given in Table 4.1 as a price per unit volume. The product price for the 0, 2 and 3-weeks lead time demands are 1.3, 1.2 and 1.1 × price of fixed demand product respectively. These demand and prices are used for the initial 52 weeks model. The first week solution is generated based on above demand and price setting. The variation in product prices and demand are considered from 2-52 weeks. The consideration technique is described in the next chapter.

Campaign and classes: Two different species are considered in this simulation: spruce and pine. Spruce is classified into 6 different classes of logs. Pine logs are classified into 2 different classes. 366 campaigns are considered from the combination of 46 pricelists and 8 different classes of logs.

Output of each campaign (O_{ijk}): The output of each campaign is lumber of different dimensions and chips. Considering the total campaign log input volume as a unit, the lumber produced for each product type is a fraction of the total volume. Chip production is calculated as sum over all of the fraction of the lumber volume produced.

Log cost: The prices considered for the 6 different classes of spruce and 2 different classes of logs are given in the table 4.2. The price includes the harvesting, bucking and transportation cost per tonne of logs.

Table 4.2: Log prices

Species	Spruce						Pine	
Class	1	2	3	4	5	6	7	8
Log Cost ($/tonne)	70	80	72	74	76	78	60	70

Campaign and class set-up time: The sawmill run-time is considered to be 8 hours per day, 5 days per week, and 40 hours per week. In the model in constraint 4.5, the total 40 hours is considered as 1 1 week. The class set-up time is considered to be higher than campaign set-up time which may involve no more that changing price lists. In the work we report and, class set-up time is 30 mins and campaign set-up time is 10 mins.

$$Set_k = \frac{10}{40 \times 60} = \frac{1}{240}\ week$$

$$Set_c = \frac{30}{40 \times 60} = \frac{1}{80}\ week$$

Initial Backorder: The initial backorder for all products is set to zero.

Initial Inventory: The initial inventory was calculated by multiplying a uniform random value between 0.5 and 1.5 to the weekly average demand. The initial inventory for pine was set to zero in this simulation.

Backorder penalty cost (γ): The back order penalty cost is $20 per (ft3). This is a high cost and should discourage backorder unless no other solution is possible.

Inventory holding limit ($Liminv$) : We set the sawmill inventory limits at a maximum of 4 weeks of its production capacity. Since the annual production capacity of the sawmill is 80,000,000 board feet (fbm), the weekly production capacity is:

$$= 80{,}000{,}000 \times \frac{1}{52} = 1538461.53 \text{ fbm} \times \frac{1}{12}\left(\frac{\text{ft}^3}{\text{fbm}}\right)$$

$$= 128205\ (\text{ft}^3/\text{week}).$$

and

$$Liminv = 4 \times 128205\ (\text{ft}^3)$$

Log volume input rate (V_{tk}) : The average yield of the campaigns obtained from the campaign-generation model is 52.91%. Thus, if we assume that the feed capability is, recognizing downtime and other interruptions, 20% higher than required for mill, the log volume input rate is:

$$\boldsymbol{V_{tk}} = \frac{128205}{0.5291} \times 1.2 = 290769\ (ft^3/week), \quad \forall\, t, k$$

The conversion factor (α) is used to calculate the total consumed-og price. The unit of log cost is \$/ton, and the unit of log-volume input is ft^3/week. The density of softwood lumber in Nova Scotia as 0.856898029 ton/m^3 is commonly used (Saadatyar, 2012)

$$\alpha = 0.0242646\ \text{tonne/ ft}^3$$

4.5. Solution Technique

The model is meant to be used in a rolling planning-horizon process. In a rolling planning horizon, the model is solved for a given time horizon. The intention is that the planning horizon is long enough to give an estimate of the future value of present production with the first period solution of each rolling planning-horizon meant to be implemented. Once the uncertain information in the first period is realized, the starting state and demand information at the beginning of the next period is updated and the model solved using the same time horizon going forward. In the next chapter, we describe an approach to simulating a 13 week rolling horizon approach that could be used for solving the order-promising and production-planning problem of a sawmill applied over 52 periods

Demands and prices vary each week. It is assumed that we know the current inventory information, and current accepted orders for each lead time, the potential orders in week 1 for each lead time l and the prices corresponding to those lead time orders. We assume that we have a forecast of the orders that are likely to occur in each week for each possible lead time at that week and again assume that we have a price for such orders. Once the model is solved, the first week's solution is considered to be implementable in terms of order acceptance and campaign schedule production to make the delivery on time. In rolling forward, we start week two as if it were week one, again knowing inventory, accepted orders, orders available to be accepted this week and updated forecasts for orders for the subsequent weeks up to a horizon of the same length as in the first model.

After solving the 2nd planning horizon, the 1st period solution of the 2nd planning horizon solution is implemented. At this stage, two implemented final solutions are obtained: one is from the 1st planning horizon solution, and the other is from the 2nd planning horizon. These two implemented solutions are also considered in the next planning horizon. In the next planning horizon, prices and demands for the coming 13 periods are updated and the two already-implemented solutions are also considered. The initial inventory for the current planning horizon is the inventory on hand at the end of the latest implemented period. This process is continued until the end period of the problem. In Chapter 5, we describe a simulation environment for testing this rolling planning approach.

4.6. CONCLUSIONS

This chapter described the research methodology of this thesis. An MIP model is formulated to solve the order promising and production planning issues in a sawmill. The data for the static model is also presented in this chapter. As order quantity and product price vary every week, a weekly optimal production schedule is required to resemble a real sawmill operation environment. In order to demonstrate the rolling planning horizon, we developed a dynamic solution technique using Gurobi 5.5 to solve the problem, where the demand and price are updated every week. The implementation and results of the solution techniques described in section 4.5 is shown in the next chapter.

CHAPTER 5

USING AN EMBEDED SIMULATION TO DEMONSTRATE THE ROLLING PLANNING-HORIZON

5.1. INTRODUCTION

In this chapter, we describe the solution techniques that we introduce to simulate the rolling planning-horizon approach in Chapter 4. A 52-week order promising and production- planning model is simulated as a rolling planning-horizon problem. In such an approach, a deterministic model is solved over a planning horizon, treating the random data as known. Only the first period of the horizon model is implemented. Once the random information for that period become known then the process is repeated with the planning horizon extended by one week. For the work presented here, the planning-horizon is considered as 13 weeks. Thus, to cover the 52 week period, we need to perform 40 optimization runs (weeks 1-13, 2-14, 3-15, …40-52). To perform these as individual optimization runs would be a very time-consuming task since, as shown previously, each run takes approximately 1100 seconds. The task we consider here is how to efficiently carry out the 40 runs necessary to simulate how the rolling planning-horizon approaches to order promising and production scheduling would perform. The approach we use takes advantage of the Python Shell provided with GUROBI 5.5 . Using this shell, the entire series of 40 optimization runs can be performed as a single task.

The models in this chapter were developed using the GUSEK environment (see Bettoni, 2013) for the GNU Linear Programming Kit (see Makhorin, 2013). The Gurobi Optimization solver GUROBI 5.5 was used to solve the models (see the website at Gurobi (2013) for information and documentation).

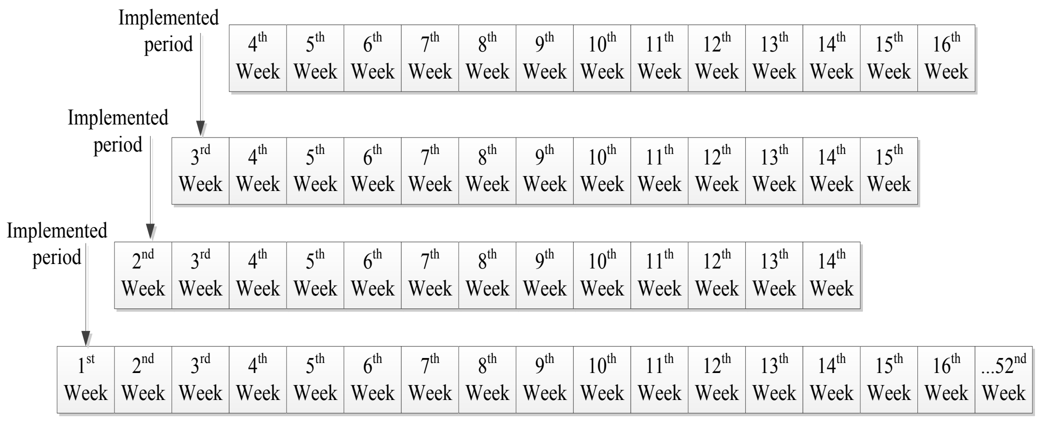

Figure 5.1: Rolling planning-horizon solution technique

First, the entire 52- week model is formulated in GMPL and read into the GUROBI 5.5 solver. The 52-week model is reduced to a 13-week model by putting all variable upper and lower bound zeros for weeks 14 to 52, and eliminating all constraints for weeks 14 to 52. Once the 1-to-13-week problem is solved, the 1st week solution is implemented.

In the second stage, week 2 to 14, all of demands and prices are updated and all of the constraints of 14 weeks are activated. The end inventory constraint is relaxed from the 13th period and set on 14th period. The 2nd week solution is implemented after solving the 2-14 week model. The process of updating solutions and moving forward to the next planning-horizon and continues throughout the 52 weeks. Fig. 5.1 indicates the solution technique for rolling horizon planning.

5.2. COMPUTATIONAL FRAME WORK

The mathematical model developed in Chapter 4 was coded using the GMPL modeling language of GLPK. In order to simulate the process of solving the model. using a rolling planning-horizon technique where variables, constraints right hand side and the objective function co-efficient value will be updated dynamically, we used the Python shell that comes with the Gurobi 5.5 solver. We used the GUSEK IDE to process the .mod file for the model. This then produces a linear programming model in the form of a *.lp file which can read by Gurobi 5.5. The initial s model was formulated as a 52 weeks problem for a fixed orders and prices. Using the Python shell within Gurobi, this 52 week problem was processed as a series of rolling planning horizon solutions as follows (the Python code to carry out this work can be found in appendix B):

Step 1: Read the 52-week *.lp model into two different model objects (i.e., m and m1).

Step 2: Reduce the 52-week model to a 13-week model. The model reduction is done in two stages. In the first stage, all variables from weeks 14 to 52 are eliminated by putting upper bound and lower bound variables to zero. The objective coefficients of those variables are also put at zero. In the second stage, all constraints from 14 to 52 weeks are relaxed based on the sense of the constraints. The three possible constraints senses, '≤', '≥' and '=', are relaxed in three different ways. The RHS value of a constraint is replaced by a big number M using the following rules:

Constraints sense	RHS value replaces by
≤	+M
≥	-M
= replaces by ≥	-M

Step 3: Solve the problem after eliminating all variables and constraints from 14 to 52 weeks.

Step 4: The first period solutions from the 13-period problem are saved and we consider those values as an implemented solution. The lower bound and upper bound of the 1st period variable values are set to the solution values.

Step 5: The next step for the rolling planning-horizon solution is to activate the next period's (i.e., period 14) variables and constraints. As well, the end-inventory constraints of period 13 need to be relaxed as well as the market demand and prices updated.

To activate the next period's (i.e., 14th period's) variables and constraints, we refer to the initial 52-week model, read previously as 'm1'. The 14th period variables' upper bound, lower bound, objective coefficient, constraints sense, and right-hand side values of model 'm' are replaced by the 14th period values of model 'm1'. The end inventory condition of the 13th period is relaxed using the step 2 and set as the 14th period constraints.

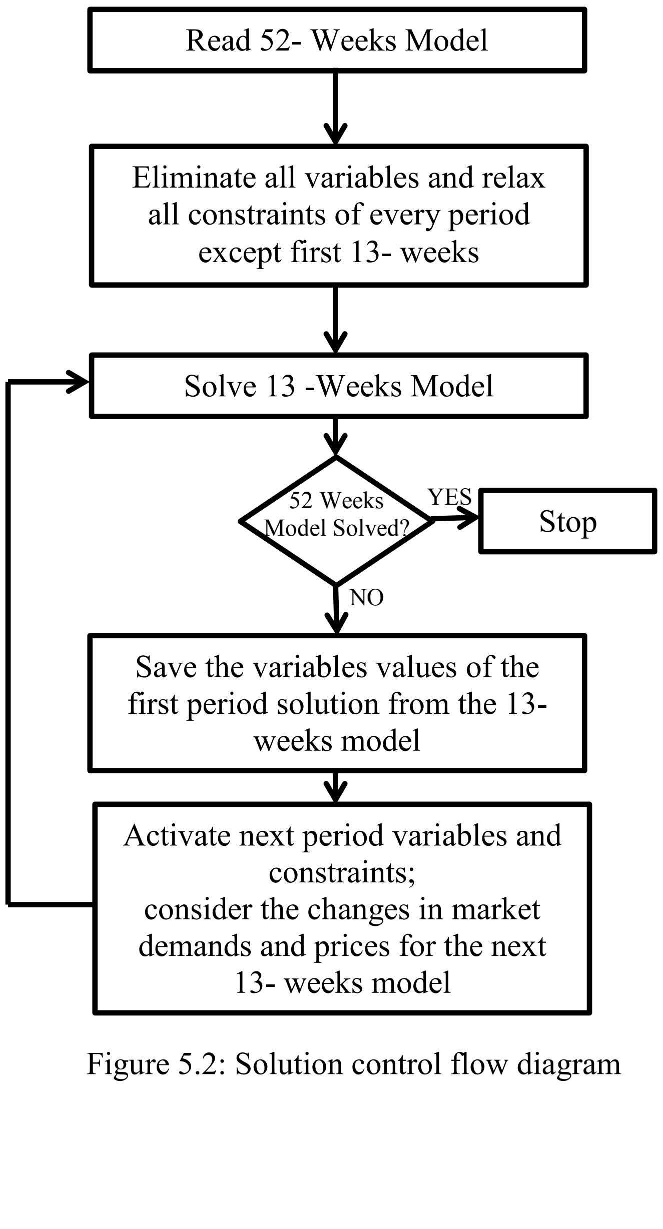
Read 52- Weeks Model
Eliminate all variables and relax all constraints of every period except first 13- weeks
Solve 13 -Weeks Model
52 Weeks Model Solved?
YES
Stop
NO
Save the variables values of the first period solution from the 13- weeks model
Activate next period variables and constraints; consider the changes in market demands and prices for the next 13- weeks model

Figure 5.2: Solution control flow diagram

The market demand and prices for the next 13 weeks are randomly generated updated for the 2- to 14-week model. Note that this is done to demonstrate the feasibility of our simulation process. More realistically, a forecasting model would be use to randomly generate anticipated markets and prices in a more formal way. The forecasting part of this process was not the subject of this thesis.

Step 6: Solve the 2- to 14-week problem and repeat step 4, step 5, and step 6. The solution control flow diagram is shown in Figure 5.2.

5.3. Results

The first 13-week model was solved considering static product prices and demands. The average weekly demands and the nominal price of each product (Table 4.1) are used to determine the product prices and demands. The prices shown in Table 4.1 are used for the contract demand. The prices of products delivered within the current week, two, and three weeks after accepting an order were treated as 1.3, 1.2, and 1.1 times the nominal price, respectively. The sum of the average weekly demand, shown in Table 4.1, is the weekly production capacity of the sawmill.

In order to test the ability of our process to cope with various random estimates of demand and price and to demonstrate the computational capability of her approach we carried out five different scenarios.

Scenario1 considered just the single13 period problem and solved once after the product demands were chosen randomly. The prices shown in table 4.1 were applied for the contract-demand and treated as constant. For the spot-demand, the nominal price is multiplied by 1.3, 1.2, and 1.1 to deliver within the current week, two, and three weeks, respectively to determine the spot-demand product price.

Scenarios 2, 3, 4, and 5 simulate the solution of a 52-week problem. The problems of those scenarios are solved for the 40 13-week planning-horizon problems.The demand and prices are randomly varied every week after the first week implemented solution.

Scenario 2 considered a ±10% price variation on the nominal price for spot-demand for weeks 2 to 52. Contract demand is 25% of capacity and spot-demand varies uniformly in between 15% and 35% of average weekly production (Table 4.1) for each lead time category. Scenario 2 has relatively small price variation and,on average, total demand less than capacity

Scenario 3 considered a ± 20% price variation on the nominal price for spot-demand for weeks 2 to 52. Contract demand is 30% of capacity and the three other spot-demands vary uniformly in between 25 and 35% of average weekly production in each category. Scenario 3 has relatively large price variation and told demand on average greater than capacity

Scenario 4 used a ±20% variation in product price for spot-demand for weeks 2 to 52.. Contract demand is 25%, the current week spot-demand varies from 20 to 30%, and the two other spot-demand vary from 30 -35% of average weekly production in each category. Scenario 4 has a quite large total demand relative to capacity with a relatively small portion of that being contract demand

In scenario 5, variation in product price is ±20%, contract demand is 25%, the current week spot-demand varies by 30-35% and the two other spot-demands vary by 35-40% of average weekly production in each category.

5.3.1. Scenario 1

The scenario 1 is generated by solving a 13-week order promising and production planning problem. The problem is solved in one step and does not consider the variation in product prices and demand for every week. The nominal prices of the product shown in Table 4.1 are used for the contract-demand. The spot prices of products delivered within the current week, two, and three weeks after accepting an order are considered to be 1.3, 1.2, and 1.1 times the nominal price respectively. The values are fixed for every week. The product demand for each period is chosen randomly. The resulting model contains 10632 continuous and 4862 binary variables. The following results are found by using the Gurobi 5.5 solver:

Best objective value: $\$1.08 \times 10^7$

Best bound: $\$1.11 \times 10^7$

The MIP gap for the 13-week solution was 2.67%, and the solution time was 1197.31 seconds.

Fig. 5.3 shows the 13-period solution for one product (product 1, species 1) under the scenario 1. It shows that the initial and final inventory is the same as limited by constraint 4.19. There is no backorder in this 13-period solution. No chips are produced from the product 1 (species1) inventory. Table 5.1 shows the inventory, production, sales, backorder and the chips that are produced from the product 1 (species1) inventory of scenario1 for the 13 week solution.

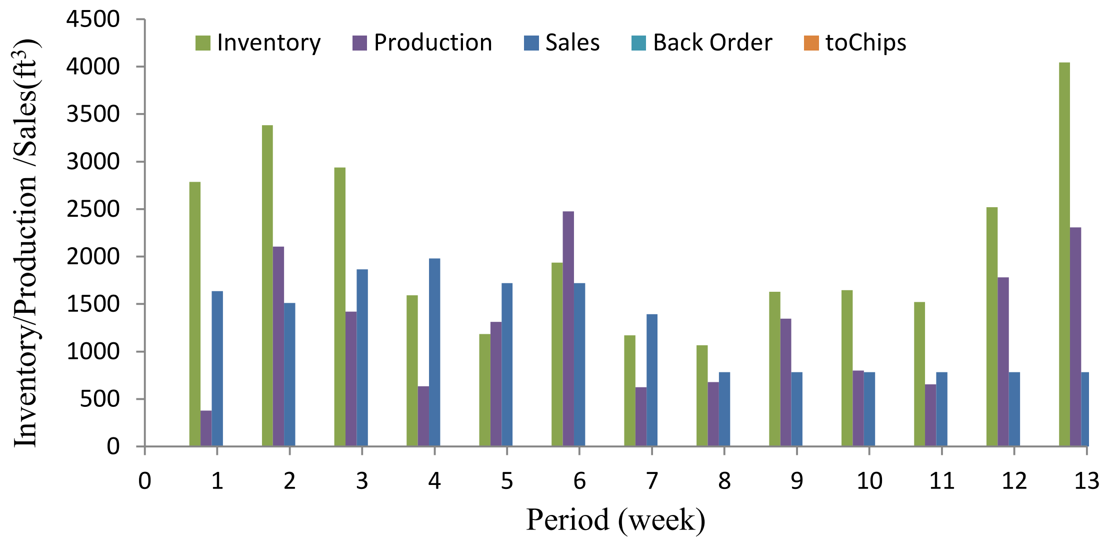

Figure 5.3: Production, sales and inventory of product1 (species 1)

The inventory-balance equation can be written as: Inventory (current period) = Inventory (previous period) + Production (current period) - Sales (current period) - ToChips (current period). This can be written as follows:

$$I_t = I_{(t-1)} + P_t - Sale_t - Bo_t - ToCh_t \tag{5.1}$$

To verify, using period 12 of scenario1- product1 (species1)

Inventory (12) = Inventory (11) + Production (12)- Sales (12)- Back order (12) - ToChips (12)

= 1520.87+1782.22-782.78- 0.00-0.0

= 2520.31

Table 5.1: Solution output for scenario1-product 1(species1)

Period	Production	Sales	Back order	ToChips	Ending Inventory
0	0	0	0	0	4044.94
1	377.66	1634.88	0	0	2787.73
2	2105.38	1511.07	0	0	3382.03

3	1420.21	1864.19	0	0	2938.06
4	634.38	1978.65	0	0	1593.78
5	1312.51	1722.11	0	0	1184.19
6	2475.11	1722.11	0	0	1937.19
7	624.20	1392.17	0	0	1169.22
8	679.42	782.78	0	0	1065.87
9	1346.51	782.78	0	0	1629.60
10	801.11	782.78	0	0	1647.94
11	655.70	782.78	0	0	1520.87
12	1782.22	782.78	0	0	2520.31
13	2307.41	782.78	0	0	4044.94

Table 5.2 below shows total sales in response to the contract and spot-demands for product 1 (species 1). There is no variation in product prices for any of the demand. The contract demand is satisfied 100%. The current week spot-demand is rejected by 48% and each of the two other spot-demands is rejected by 100%. The total sales in the current period = Sales corresponding to the promised order for the contract demand + Sales corresponding to the promised order for the current period + Sales corresponding to the promised order 2 periods prior + Sales corresponding to the promised order 3 periods prior. This can be written as follows:

$$Sale_t = S_t^F + S_t^0 + S_{t-2}^2 + S_{t-3}^3 \quad 5.2$$

As an other verification , period 4 sales from Table 5.2 is given below:

Sales (4) = Contract demand (4) + Promised Order (4)+ Promised Order (2)

+ Promised Order (1)

= 899.39+1079.27+0.00+0.00

= 1978.66 ft^3

Table 5.2: Demand, promised order and sales for scenario1-product1(species1)

Period	v^F	d^F	S^F	v^0	d^0	S^0	v^2	d^2	S^2	v^3	d^3	S^3	Total Sales
1	4.42	743.13	743.13	5.75	891.75	891.75	5.31	966.07	0.00	4.86	1114.69	0.00	1634.88
2	4.42	686.85	686.85	5.75	824.22	824.22	5.31	892.91	0.00	4.86	1030.28	0.00	1511.07
3	4.42	847.36	847.36	5.75	1016.83	1016.83	5.31	1101.57	0.00	4.86	1271.04	0.00	1864.19
4	4.42	899.39	899.39	5.75	1079.27	1079.27	5.31	1169.20	0.00	4.86	1349.08	0.00	1978.65
5	4.42	782.78	782.78	5.75	939.33	939.33	5.31	1017.61	0.00	4.86	1174.16	0.00	1722.11
6	4.42	782.78	782.78	5.75	939.33	939.33	5.31	1017.61	0.00	4.86	1174.16	0.00	1722.11
7	4.42	782.78	782.78	5.75	939.33	609.40	5.31	1017.61	0.00	4.86	1174.16	0.00	1392.17
8	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	0.00	4.86	1174.16	0.00	782.78
9	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	0.00	4.86	1174.16	0.00	782.78
10	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	0.00	4.86	1174.16	0.00	782.78
11	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	0.00	4.86	1174.16	1174.16	782.78
12	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	1017.61	4.86	1174.16	1174.16	782.78
13	4.42	782.78	782.78	5.75	939.33	0.00	5.31	1017.61	1017.61	4.86	1174.16	1174.16	782.78

Notation used in Table 5.2, 5.4, 5.6, 5.9, and 511:

v^F Product price for the contract demand;

v^0 Product price for the current week variable demand;

v^2 Product price to be delivered in 2 weeks;

v^3 Product price to be delivered in 3 weeks;

d^F Contract demand;

d^0 Current week variable demand;

d^2 Product demand to be delivered in 2 weeks;

d^3 Product demand to be delivered in 3 weeks;

S^F Promised order for contract demand;

S^0 Promised order for current week delivery;

S^2 Promised order to be delivered in 2 weeks;

S^3 Promised order to be delivered in 3 weeks.

The nature of this solution for product one was to accept all the spot-demand until no more can be used and still meet the ending inventory constraints. Note that it looks as if two week and three week ahead orders will be promised in weeks 12 and 13 but note that these will never be processed.

5.3.2. Scenario 2

In Scenario 2, a 52-week order-promising and production-planning model is simulated using the rolling planning-horizon technique. The problem is solved as 40 individual planning-horizon problem and 40 implemented solutions are obtained. The variation of product price of ±10% of the nominal price (price considered in scenario 1) for all three spot-demands is considered for every week of each planning horizon. The price for the contract demand is considered the same as the nominal price for each week. Contract demands are 25% of the weekly average production and spot-demands vary in between 15 and 35% of average weekly production in each category. The Gurobi 5.5 calculation produces the following solution using the developed solution algorithm:

Best objective value: $\$2.20\times10^{7}$

Best bound: $\$2.23\times10^{7}$

The solution time is 15.59 hours. Fig. 5.4 shows the solution output for the 52 weeks solution.

The 52-week solution shows that the end inventory, limited by constraint 4.19, is equal to the initial inventory of 4,044.94 (ft^3). As we are assuming that chips are sold at a lower price than lumber, no chips were produced from the lumber inventory as long as the maximum inventory is within the limit inventory. The constraints for the maximum inventory are set at equation 4.15.

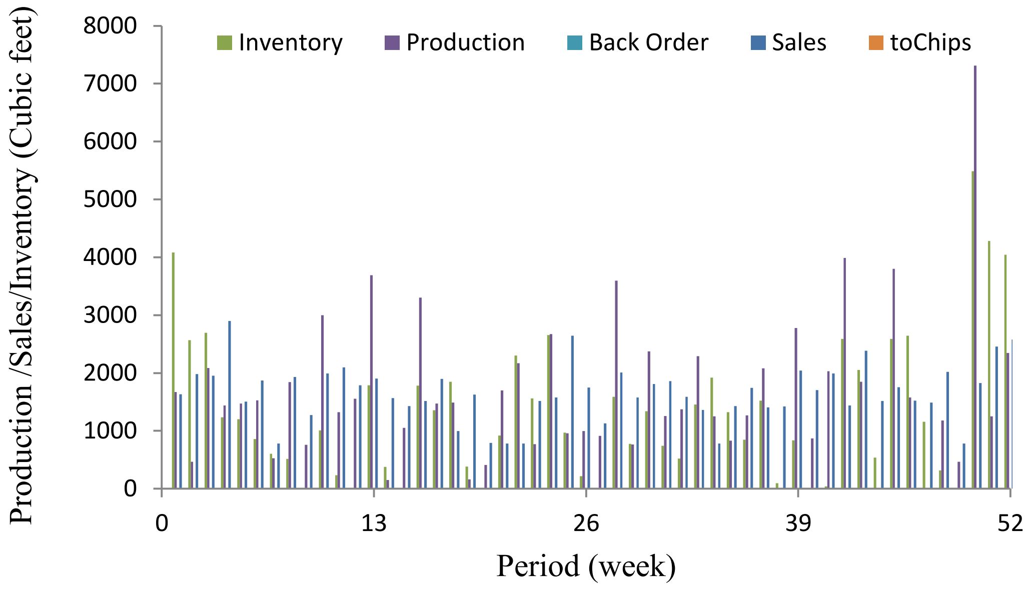

Figure 5.4: Production, sales and inventory of scenario2-product1 (species 1)

Table 5.3 shows the output of the 40 implemented periods of scenario2-product1 (species 1). The result satisfied the inventory balance equation 5.1 of every implemented period. Again, verifying, taking period 12 as an example.

Inventory (12) = Inventory (11) + Production (12)- Sales (12)- Back order (12) - ToChips (12)

= 232.73+1552.44-1785.17- 0.00-0.0

= 0

Table 5.3: Solution output of scenario2- product 1(Species 1)

Period	Production	Sales	Back order	ToChips	Ending Inventory
0	0.00	0.00	0	0	4044.94
1	1671.04	1634.88	0	0	4081.11
2	465.02	1981.71	0	0	2564.41
3	2086.26	1954.05	0	0	2696.63

4	1436.63	2899.99	0	0	1233.27
5	1469.91	1503.11	0	0	1200.07
6	1525.53	1869.40	0	0	856.21
7	527.76	782.80	0	0	601.17
8	1843.20	1929.73	0	0	514.63
9	759.64	1274.27	0	0	0.00
10	3000.24	1991.48	0	0	1008.75
11	1321.49	2097.51	0	0	232.73
12	1552.44	1785.17	0	0	0.00
13	3688.85	1900.69	0	0	1788.16
14	151.64	1563.72	0	0	376.08
15	1054.02	1430.10	0	0	0.00
16	3300.01	1518.03	0	0	1781.98
17	1471.56	1896.61	0	0	1356.93
18	1490.68	998.68	0	0	1848.93
19	163.89	1629.31	0	0	383.51
20	409.58	793.09	0	0	0.00
21	1701.32	782.80	0	0	918.52
22	2167.64	782.80	0	0	2303.36
23	771.38	1515.38	0	0	1559.37
24	2674.35	1577.54	0	0	2656.18
25	955.29	2645.54	0	0	965.94
26	996.45	1747.03	0	0	215.36
27	913.75	1129.11	0	0	0.00
28	3597.62	2009.78	0	0	1587.85
29	762.27	1575.40	0	0	774.71
30	2375.97	1809.57	0	0	1341.11
31	1257.99	1857.54	0	0	741.56
32	1370.83	1590.79	0	0	521.60
33	2292.65	1359.35	0	0	1454.89
34	1249.75	782.80	0	0	1921.84
35	829.60	1427.27	0	0	1324.17
36	1267.07	1744.35	0	0	846.89
37	2081.53	1407.16	0	0	1521.26
38	0.00	1424.46	0	0	96.81
39	2778.36	2039.96	0	0	835.21
40	868.25	1703.46	0	0	0.00

Table 5.4 shows that the total sales are the sum of orders promised for spot and contract demands for the product 1 (species 1). As we consider higher prices for earlier delivery,

most of the spot order is promised for the next week's delivery. If the next week's demand amount is less than the product availability, then it is promised for a 2-week delivery, and so on. The contract demand is satisfied by 100%. The current week spot-demand is rejected in 32.49% of cases, the two weeks advanced demand is rejected by 82.59% and the three weeks advanced demand is rejected by 100%.

As an example, period 12 sales from Table 5.4 is given below:

Sales (12) = Contract demand (12) + Promised Order (12)+ Promised Order (10) + Promised Order (9)

= 782.80+0.00+1002.37+0.00

= 1785.17

This was a case where the spot price in period 12 is low and the spot 2 period lead time price in period 10 was higher.

Table 5.4: Demand, order promised and sales for scenario 2- product 1(species1)

Per iod	v^F	d^F	S^F	v^0	d^0	S^0	v^2	d^2	S^2	v^3	d^3	S^3	Total Sales
1	4.42	743.13	743.13	5.75	891.75	891.75	5.31	924.68	100.42	4.86	747.47	0.00	1634.88
2	4.42	782.80	782.80	5.78	1198.91	1198.91	5.68	924.68	924.68	4.93	747.47	0.00	1981.71
3	4.42	782.80	782.80	5.43	1070.83	1070.83	5.23	1022.22	0.00	4.74	858.10	0.00	1954.05
4	4.42	782.80	782.80	5.66	1192.51	1192.51	5.16	1153.68	0.00	4.82	972.00	0.00	2899.99
5	4.42	782.80	782.80	5.36	1086.42	720.31	4.99	912.02	0.00	5.03	1062.74	0.00	1503.11
6	4.42	782.80	782.80	6.17	1086.60	1086.60	5.52	1146.93	1,146.93	4.63	1239.59	0.00	1869.40
7	4.42	782.80	782.80	5.39	597.26	0.00	5.14	712.08	0.00	4.92	1288.40	0.00	782.80
8	4.42	782.80	782.80	5.38	894.01	0.00	5.30	839.40	0.00	4.60	996.10	0.00	1929.73
9	4.42	782.80	782.80	5.80	1267.45	491.47	5.34	1057.31	309.49	4.71	1348.37	0.00	1274.27
10	4.42	782.80	782.80	6.19	1208.68	1208.68	5.64	1002.37	1,002.37	4.59	768.39	0.00	1991.48
11	4.42	782.80	782.80	6.18	1005.22	1005.22	5.56	624.64	0.00	4.78	1099.74	0.00	2097.51
12	4.42	782.80	782.80	5.40	625.60	0.00	5.02	1245.78	0.00	5.30	878.96	0.00	1785.17
13	4.42	782.80	782.80	6.19	1117.89	1117.89	5.29	785.55	0.00	5.12	1046.79	0.00	1900.69
14	4.42	782.80	782.80	5.74	780.92	780.92	5.23	925.82	0.00	4.56	1075.73	0.00	1563.72
15	4.42	782.80	782.80	5.99	742.19	647.30	5.22	1204.18	0.00	4.50	1015.45	0.00	1430.10
16	4.42	782.80	782.80	5.59	735.23	735.23	4.94	994.35	0.00	4.73	1251.13	0.00	1518.03
17	4.42	782.80	782.80	6.08	1113.81	1113.81	5.22	885.08	0.00	4.96	790.71	0.00	1896.61

18	4.42	782.80	782.80	5.77	1306.62	215.88	5.41	643.02	0.00	4.64	794.68	0.00	998.68
19	4.42	782.80	782.80	6.11	1110.93	846.51	5.01	725.80	0.00	4.53	604.07	0.00	1629.31
20	4.42	782.80	782.80	5.82	1013.66	10.29	5.10	999.31	0.00	4.62	819.27	0.00	793.09
21	4.42	782.80	782.80	5.41	1292.86	0.00	5.29	684.77	0.00	4.62	1002.38	0.00	782.80
22	4.42	782.80	782.80	5.35	958.95	0.00	5.21	1279.44	0.00	4.56	1362.28	0.00	782.80
23	4.42	782.80	782.80	5.68	732.58	732.58	5.33	591.57	591.57	4.54	898.86	0.00	1515.38
24	4.42	782.80	782.80	6.04	794.74	794.74	5.45	929.71	0.00	5.28	901.80	0.00	1577.54
25	4.42	782.80	782.80	5.78	1271.16	1271.16	5.23	700.96	0.00	4.95	846.67	0.00	2645.54
26	4.42	782.80	782.80	6.19	964.23	964.23	5.64	776.18	84.03	4.93	1357.08	0.00	1747.03
27	4.42	782.80	782.80	5.86	1073.87	346.31	5.39	656.62	0.00	4.90	1337.26	0.00	1129.11
28	4.42	782.80	782.80	5.95	1142.95	1142.95	5.50	1295.28	0.00	4.82	1170.47	0.00	2009.78
29	4.42	782.80	782.80	6.18	792.60	792.60	5.16	1329.22	0.00	4.58	683.61	0.00	1575.40
30	4.42	782.80	782.80	6.18	1026.77	1026.77	5.02	620.68	0.00	5.24	907.85	0.00	1809.57
31	4.42	782.80	782.80	5.57	1074.74	1074.74	5.40	589.09	0.00	4.81	840.42	0.00	1857.54
32	4.42	782.80	782.80	6.03	807.99	807.99	4.93	843.54	0.00	5.20	1124.13	0.00	1590.79
33	4.42	782.80	782.80	5.56	1102.50	576.55	5.08	679.09	0.00	4.62	940.38	0.00	1359.35
34	4.42	782.80	782.80	5.67	1350.00	0.00	5.23	1062.69	0.00	5.19	1080.92	0.00	782.80
35	4.42	782.80	782.80	5.46	644.47	644.47	5.56	624.36	624.36	5.30	1253.52	0.00	1427.27
36	4.42	782.80	782.80	5.85	961.55	961.55	5.21	1339.51	0.00	5.15	805.21	0.00	1744.35
37	4.42	782.80	782.80	5.35	1262.91	0.00	5.54	1126.84	545.07	5.00	792.98	0.00	1407.16
38	4.42	782.80	782.80	5.44	1250.21	641.66	4.98	702.72	0.00	4.94	1155.93	0.00	1424.46
39	4.42	782.80	782.80	5.45	1254.25	712.09	5.06	1256.67	0.00	4.76	1236.90	0.00	2039.96
40	4.42	782.80	782.80	5.94	1297.96	920.66	5.63	655.73	655.73	4.56	1175.87	0.00	1703.46

5.3.3. Scenario 3

Scenario three used a ±20% variation of the nominal price for each week's spot-demand. The price for the contract demand is the same as the nominal price for every week. The contract demand is 30% of nominal weekly production and the three other spot-demands vary from 25-35% of the nominal weekly production. The Gurobi 5.5 solver found the following results for the 52-week model using the designed solution technique:

Best objective value: $\$2.43 \times 10^7$

Best bound: $\$2.47 \times 10^7$

Time taken to solve the problem is 16.62 hours. Fig. 5.4 shows the generated revenue after solving each planning-horizon of a 40 planning-horizon problem.

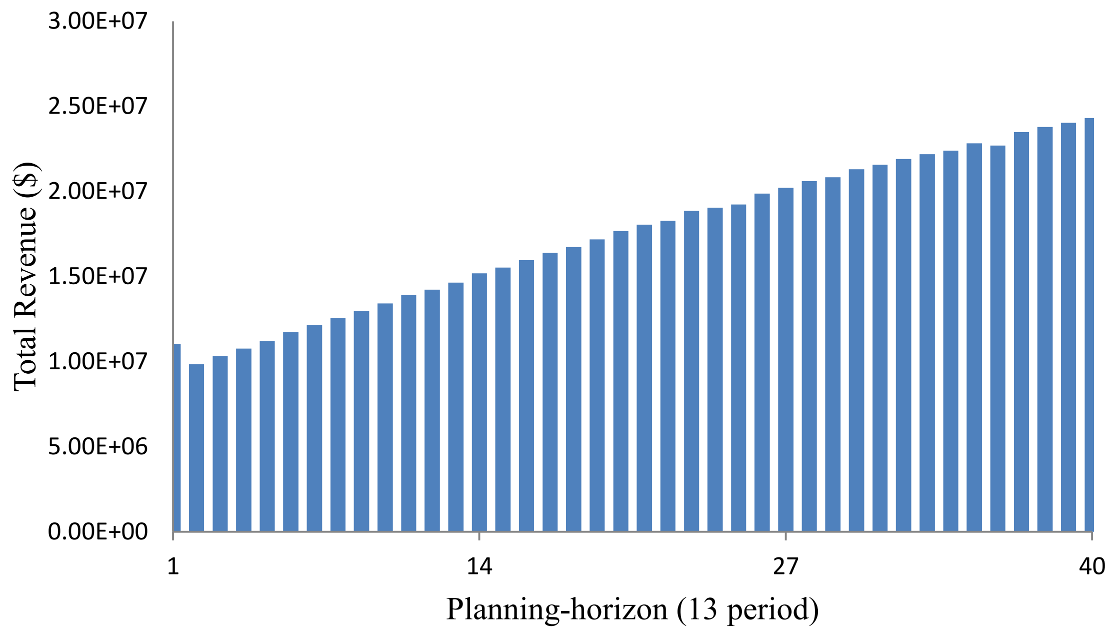

Figure 5.4: Revenue after solving each planning-horizon for scenario 3

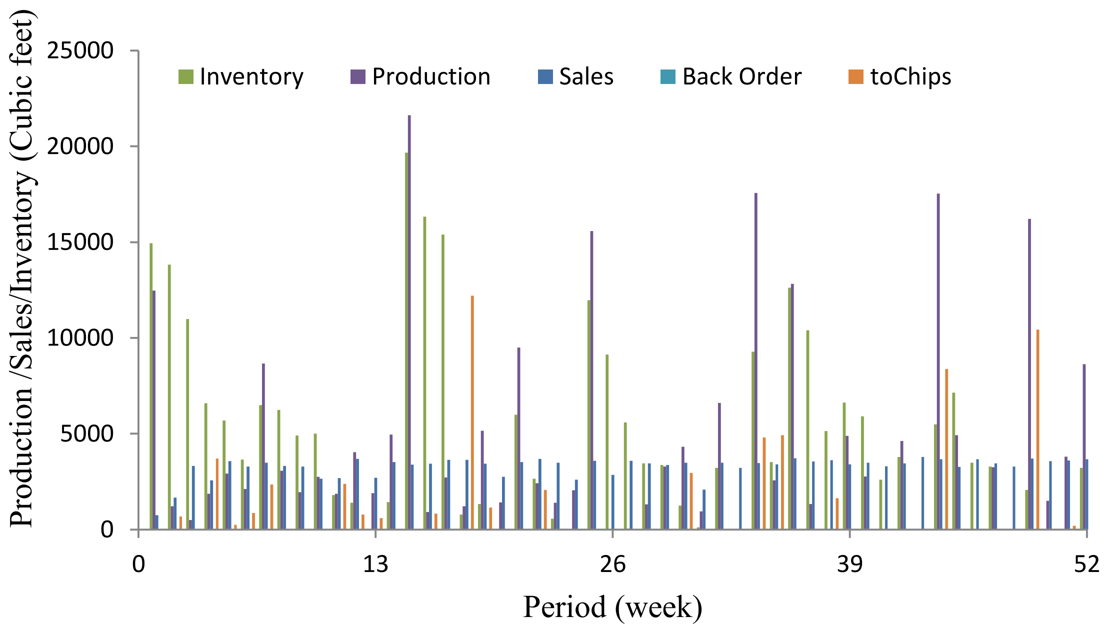

Figure 5.5: Production, sales and inventory of product5 (species 1) of scenario 3

Figure 5.5 shows the result of product 5 (species1) for the scenario 3. The initial and final inventories are same. No backorder is generated but chips are produced from the product 5 inventory over time. Although chips are sold at a lower price than the lumber, chips are produced from the lumber inventory when the sum of inventories exceeds a limiting inventory given by Equation 4.15.

Table 5.5 shows the 40 implemented 1-period solutions of product 5 (species 1) of scenario 3. The inventory balance equation 5.1 is satisfied in every period. As verifying example look at period 6 of Table 5.5

Inventory (6) = Inventory (5) + Production (6)- Sales (6)- Back order (6) - ToChips (6)

= 5696.95+2123.43-3293.57-0.00-869.30 = 3657.51

Table 5.5: Solution output of the scenario 3 for product 5 (species1)

Period	Production	Sales	Back order	ToChips	Ending Inventory
0	0.00	0.00	0	0.00	3218.40
1	12477.22	754.29	0	0.00	14941.33
2	1224.03	1664.88	0	678.31	13822.17
3	493.34	3325.13	0	0.00	10990.39
4	1874.45	2562.43	0	3710.99	6591.41
5	2917.72	3566.78	0	245.39	5696.95
6	2123.43	3293.57	0	869.30	3657.51
7	8670.56	3484.13	0	2347.62	6496.32
8	3070.81	3320.31	0	0.00	6246.82
9	1947.32	3291.12	0	0.00	4903.03
10	2757.13	2648.14	0	0.00	5012.02
11	1863.81	2686.86	0	2381.63	1807.34
12	4046.03	3682.75	0	775.09	1395.52
13	1896.03	2699.68	0	591.87	0.00
14	4959.85	3519.77	0	0.00	1440.09
15	21626.13	3387.75	0	0.00	19678.47
16	924.22	3438.76	0	833.83	16330.11
17	2714.53	3642.70	0	0.00	15401.93
18	1211.44	3634.38	0	12202.61	776.38

19	5158.95	3440.90	0	1157.49	1336.95
20	1409.99	2746.94	0	0.00	0.00
21	9506.92	3516.69	0	0.00	5990.23
22	2417.87	3694.20	0	2067.11	2646.80
23	1408.00	3492.00	0	0.00	562.80
24	2046.72	2609.52	0	0.00	0.00
25	15571.12	3594.23	0	0.00	11976.90
26	2.22	2852.44	0	0.00	9126.68
27	50.01	3585.15	0	0.00	5591.54
28	1321.73	3454.49	0	0.00	3458.78
29	3288.93	3372.02	0	0.00	3375.69
30	4323.90	3495.00	0	2955.53	1249.06
31	945.78	2082.02	0	0.00	112.82
32	6605.12	3489.35	0	0.00	3228.58
33	0.00	3228.58	0	0.00	0.00
34	17563.15	3469.70	0	4808.52	9284.93
35	2563.08	3400.53	0	4931.87	3515.62
36	12828.76	3714.89	0	0.00	12629.49
37	1335.16	3561.40	0	0.00	10403.26
38	0.00	3615.49	0	1641.67	5146.09
39	4882.86	3405.33	0	0.00	6623.62
40	2769.15	3482.25	0	0.00	5910.52

Table 5.6 shows variations in product price, demand, the promised order and sales for 40 implemented periods for the product 5 (species 1). It shows that at period 20 the model promised for a higher price order having a longer delivery date in the presence of shortage of inventory. Table 5.6 data shows an agreement with the sales equation 5.2. The contract demand is satisfied by 100%. The current week spot-demand is rejected by 8.48%, the two weeks advanced demand is rejected by 5% and the three weeks advanced demand is rejected by 11%.

An example is given below to see the agreement between the sales in Table 5.6 and equation 5.2 for period 23.

Sales (23) = Contract demand (23) + Promised Order (23)+ Promised Order (21) + Promised Order (20)

= 880.8+857.91+951.56+801.73 = 3492.00

Table 5.6: Demand, order promised and sales for scenario 3- product 5(species1)

Per iod	v^F	d^F	S^F	v^0	d^0	S^0	v^2	d^2	S^2	v^3	d^3	S^3	Total Sales
1	4.81	342.86	342.86	6.25	411.43	411.43	5.77	1945.7	1645.72	5.29	1645.7	0.00	754.29
2	4.81	880.80	880.80	5.86	784.08	784.08	6.36	870.48	870.48	4.92	960.25	960.25	1664.88
3	4.81	880.80	880.80	5.75	798.61	798.61	4.99	890.81	890.81	5.88	769.10	769.10	3325.13
4	4.81	880.80	880.80	6.68	811.15	811.15	5.80	737.13	737.13	5.36	942.40	942.40	2562.43
5	4.81	880.80	880.80	5.53	834.92	834.92	6.55	842.97	842.97	4.92	869.36	869.36	3566.78
6	4.81	880.80	880.80	7.08	906.54	906.54	6.36	804.42	804.42	5.13	788.88	788.88	3293.57
7	4.81	880.80	880.80	6.13	817.97	817.97	6.49	847.95	847.95	6.13	887.74	887.74	3484.13
8	4.81	880.80	880.80	5.83	765.72	765.72	5.14	849.84	0.00	5.36	764.01	0.00	3320.31
9	4.81	880.80	880.80	6.72	773.49	773.49	5.36	988.59	988.59	5.94	992.20	992.20	3291.12
10	4.81	880.80	880.80	6.49	879.60	879.60	5.35	909.62	909.62	4.57	894.43	0.00	2648.14
11	4.81	880.80	880.80	7.14	817.47	817.47	4.88	846.16	846.16	4.92	1015.83	1015.83	2686.86
12	4.81	880.80	880.80	5.86	900.13	900.13	6.41	765.08	765.08	5.50	831.80	831.80	3682.75
13	4.81	880.80	880.80	5.85	972.72	972.72	5.96	856.45	856.45	4.59	889.00	889.00	2699.68
14	4.81	880.80	880.80	7.10	858.05	858.05	5.50	826.99	826.99	4.71	1020.41	1020.41	3519.77
15	4.81	880.80	880.80	6.65	818.70	818.70	6.06	743.87	743.87	4.65	839.65	839.65	3387.75
16	4.81	880.80	880.80	6.88	841.97	841.97	5.01	886.72	886.72	5.87	978.79	978.79	3438.76
17	4.81	880.80	880.80	7.15	997.62	997.62	5.44	788.86	788.86	5.59	1004.90	1004.90	3642.70
18	4.81	880.80	880.80	7.10	1027.21	1027.21	6.48	861.24	861.24	5.28	881.30	881.30	3634.38
19	4.81	880.80	880.80	7.08	792.45	792.45	6.03	871.49	871.49	4.88	994.33	994.33	3440.90
20	4.81	880.80	880.80	6.26	802.18	0.00	6.55	970.66	970.66	5.32	801.73	801.73	2746.94
21	4.81	880.80	880.80	6.65	883.10	883.10	4.92	951.56	951.56	4.86	748.73	748.73	3516.69
22	4.81	880.80	880.80	6.80	848.42	848.42	6.49	775.70	775.70	4.77	834.36	834.36	3694.20
23	4.81	880.80	880.80	6.19	857.91	857.91	6.21	963.07	963.07	5.47	961.12	148.92	3492.00
24	4.81	880.80	880.80	6.56	842.24	204.29	6.51	951.59	951.59	5.43	964.79	964.79	2609.52
25	4.81	880.80	880.80	6.78	916.00	916.00	6.02	846.81	846.81	5.96	898.68	898.68	3594.23
26	4.81	880.80	880.80	6.54	871.12	871.12	5.61	849.57	849.57	4.78	822.17	822.17	2852.44
27	4.81	880.80	880.80	6.98	892.76	892.76	5.05	920.23	920.23	5.35	970.40	970.40	3585.15
28	4.81	880.80	880.80	6.91	825.44	825.44	5.51	738.01	738.01	5.22	946.17	946.17	3454.49
29	4.81	880.80	880.80	5.75	969.67	748.82	4.87	886.09	255.05	5.54	829.25	829.25	3372.02
30	4.81	880.80	880.80	7.07	905.79	905.79	5.74	873.43	873.43	5.02	795.33	795.33	3495.00
31	4.81	880.80	880.80	6.15	962.43	0.00	4.85	931.77	931.77	6.23	908.05	908.05	2082.02
32	4.81	880.80	880.80	6.93	905.88	905.88	6.64	747.35	747.35	5.16	798.24	798.24	3489.35
33	4.81	880.80	880.80	5.77	913.59	620.67	6.52	865.61	865.61	6.12	931.69	931.69	3228.58
34	4.81	880.80	880.80	6.75	933.50	933.50	6.37	915.44	915.44	5.41	998.70	998.70	3469.70
35	4.81	880.80	880.80	5.32	855.88	855.88	6.39	880.38	880.38	4.81	896.66	896.66	3400.53
36	4.81	880.80	880.80	5.81	986.96	986.96	6.40	864.89	864.89	4.49	936.14	936.14	3714.89
37	4.81	880.80	880.80	6.52	801.52	801.52	5.84	810.37	810.37	4.55	759.44	759.44	3561.40

38	4.81	880.80	880.80	7.17	973.14	973.14	5.53	988.61	988.61	5.65	899.34	899.34	3615.49
39	4.81	880.80	880.80	6.61	778.02	778.02	5.03	763.88	763.88	4.93	842.72	760.54	3405.33
40	4.81	880.80	880.80	7.21	853.40	853.40	6.11	931.63	931.63	5.41	943.81	943.81	3482.25

Different sets of campaigns are chosen for different periods of production runs, out of 366 campaigns of 8 different log classes.

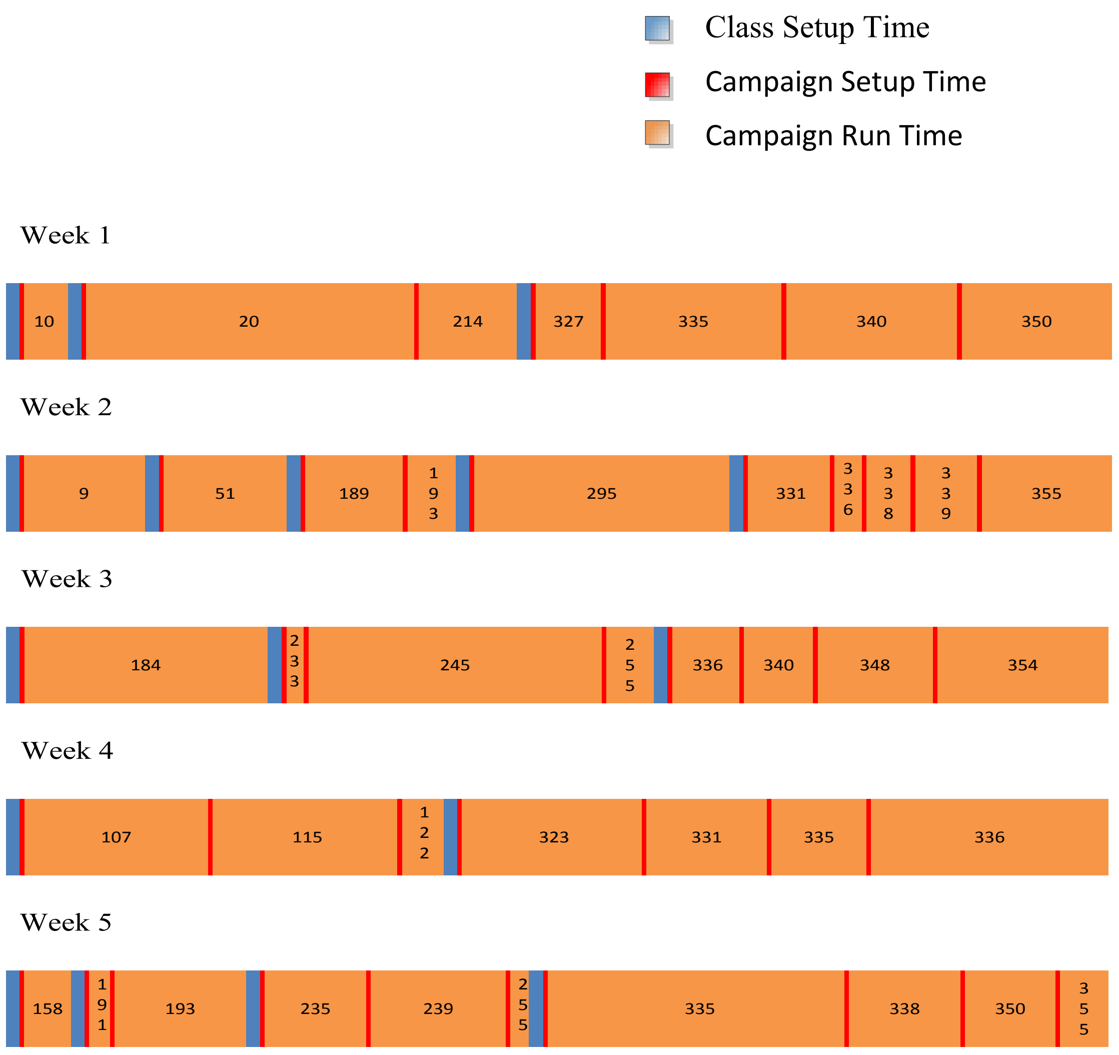

Figure 5.6: Campaign schedule for the first five implemented solution of scenario 3

In the first implemented solution, 7 campaigns are scheduled. These are campaign number 10, 20, 214, 327, 335, 340 and 350. The number of campaigns setup is 346 and class setup is 148 during the 52 weeks. The total campaign setup time is 86.25 hours. The total class setup time is 74 hours. The total of setup time is 158.25 hours. The total run time of the sawmill in a year is 2080 hours. A representative figure is shown in Fig 5.6 from the solution output of campaign scheduling. The first 5 of the 40 implemented periods of the third scenario is presented in Fig 5.6. The total setup time is 7.61%, and the resource utilization is 92.39%. Table 5.7 shows the campaign scheduling data for the first 12 implemented periods.

Table 5.7: Campaign schedule for 12 implemented periods of scenario 3

Implimented Period (week)	Class No.	Class Setup time (week)	Campaign No.	Campaign Setup time (week)	Campaign Run time (week)
1st week	1	0.0125	10	0.004167	0.0395530
	5	0.0125	20	0.004167	0.2962737
			214	0.004167	0.0891129
	8	0.0125	327	0.004167	0.0590578
			335	0.004167	0.1591718
			340	0.004167	0.1546645
			350	0.004167	0.1354994
2nd week	1	0.0125	9	0.004167	0.1093239
	2	0.0125	51	0.004167	0.1115516
	5	0.0125	189	0.004167	0.0881067
			193	0.004167	0.0437483
	7	0.0125	295	0.004167	0.2310293
	8	0.0125	331	0.004167	0.0739152
			336	0.004167	0.0250550
			338	0.004167	0.0398130
			339	0.004167	0.0559957
			355	0.004167	0.1172941
3rd week	5	0.0125	184	0.004167	0.2209512
	6	0.0125	233	0.004167	0.0158607
			245	0.004167	0.2662212
			255	0.004167	0.0429422
	8	0.0125	336	0.004167	0.0610417

			340	0.004167	0.0628116
			348	0.004167	0.1045706
			354	0.004167	0.1547670
4th week	3	0.0125	107	0.004167	0.1668509
			115	0.004167	0.1674682
			122	0.004167	0.0377708
	8	0.0125	323	0.004167	0.1630860
			331	0.004167	0.1093144
			335	0.004167	0.0862242
			336	0.004167	0.2151190
5th week	4	0.0125	158	0.004167	0.0422229
	5	0.0125	191	0.004167	0.0188961
			193	0.004167	0.1193695
	6	0.0125	235	0.004167	0.0922358
			239	0.004167	0.1223649
			255	0.004167	0.0170517
	8	0.0125	335	0.004167	0.2693975
			338	0.004167	0.1011253
			350	0.004167	0.0815802
			355	0.004167	0.0440889
6th week	4	0.0125	158	0.004167	0.2490817
			167	0.004167	0.0097653
	5	0.0125	189	0.004167	0.2700899
	7	0.0125	295	0.004167	0.4168962
7th week	1	0.0125	20	0.004167	0.2194615
	8	0.0125	336	0.004167	0.1762062
			338	0.004167	0.0641946
			340	0.004167	0.2881683
			350	0.004167	0.2061357
8th week	5	0.0125	188	0.004167	0.2982444
			193	0.004167	0.2450194
	6	0.0125	233	0.004167	0.1311059
			238	0.004167	0.2839634
9th week	3	0.0125	101	0.004167	0.105512
	4	0.0125	146	0.004167	0.0934151
	5	0.0125	184	0.004167	0.1147294
	8	0.0125	330	0.004167	0.2173151
			340	0.004167	0.1448942
			350	0.004167	0.0640390
			354	0.004167	0.1809282
10th week	4	0.0125	158	0.004167	0.4648595
			167	0.004167	0.0944932
	7	0.0125	295	0.004167	0.1826052
	8	0.0125	350	0.004167	0.2038753

11th week	3	0.0125	97	0.004167	0.0463855
			116	0.004167	0.0189331
			122	0.004167	0.1088439
	8	0.0125	322	0.004167	0.0724454
			331	0.004167	0.0599732
			335	0.004167	0.1521234
			336	0.004167	0.3029408
			340	0.004167	0.1545215
			355	0.004167	0.0213328
12th week	3	0.0125	114	0.004167	0.0755888
	6	0.0125	235	0.004167	0.4439730
	7	0.0125	290	0.004167	0.1507505
			295	0.004167	0.1386951
	8	0.0125	338	0.004167	0.1201591

Six runs of simulation were performed for scenario 3. The total revenue obtained by simulating scenarios 3 for six different runs were \$ 2.43×10^7, \$ 2.41×10^7, \$ 2.41×10^7, \$ 2.44×10^7, \$ 2.38×10^7, and \$ 2.38×10^7. If outputs are assumed to be normally distributed I, this corresponds to a 95% confidence interval of \$ $(2.40\pm0.02)\times10^7$.

The narrow confidence interval of the total revenue indicates that in each run of simulation almost same amount of revenue is generated. The revenue is generated through order promising and delivering the promised order on time. These activities (i.e., the order promising and delivering) depend on the market prices and demands in each period. To explain the activity of each simulation run, product-23 ($2\times6\times12$) is selected randomly and the sales activity in six different simulation runs is plotted in Fig. 5.7.

It shows that the sales pattern is very different in each run of each period. For an example, in period 9, the sales amount in run 1 is 4795.11 ft^3 and in run 4 is 1195.50 ft^3, in period 18, sales in run 2 is 1747.58 ft^3 and in run 3 is 4846.67 ft^3.

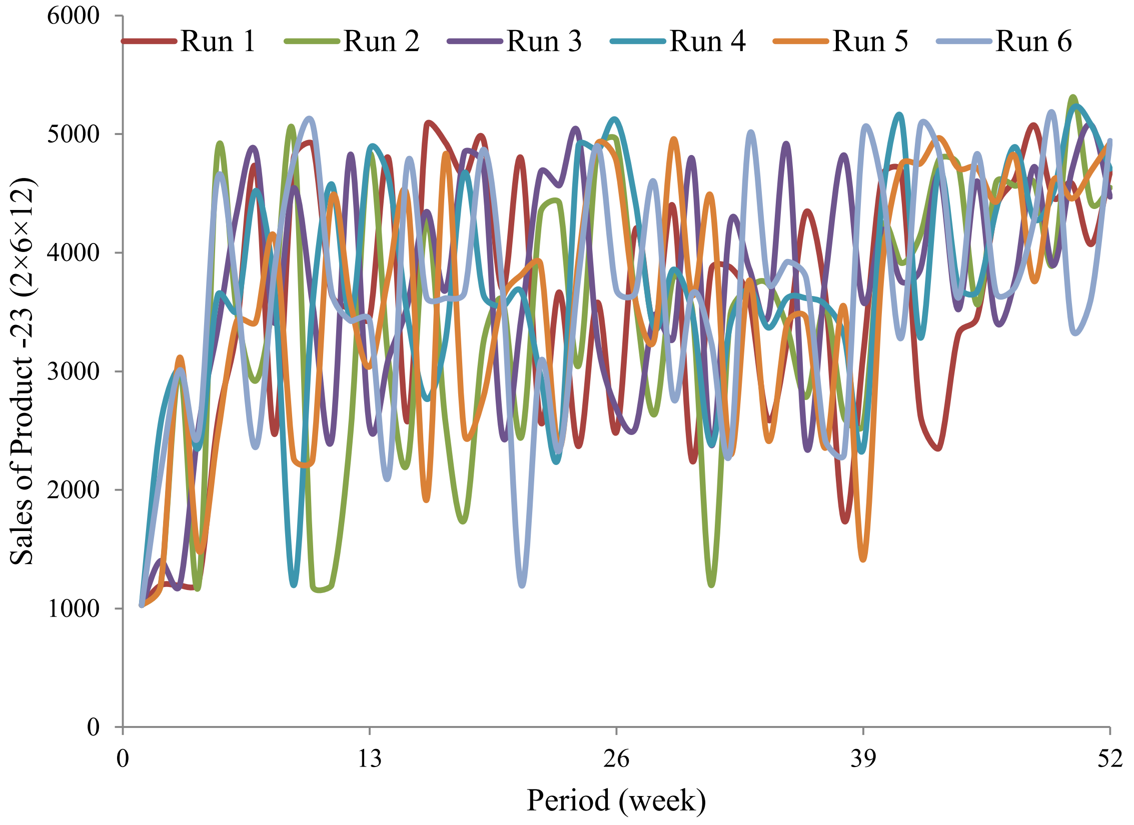

Figure 5.7: Sales of product-23 (species 1) of scenario 3 for different simulation runs

Thus, although the rolling planning approach achieves about the same result over each simulation, quite different production strategies were used to cope with the randomness.

Fig. 5.8 shows the promised order for the current period for the product-23. It shows that the order promising for each simulation run is again quite different from others in every period. Here the same period (i.e., period 9) is considered to explain, the promised order for the current period for run 1 is 1153.80 ft^3 and for run 4 is 0 ft^3. In period 18, promised order for the current period in run 2 is 246.63 ft^3 and for run 3 is 1258.39 ft^3.

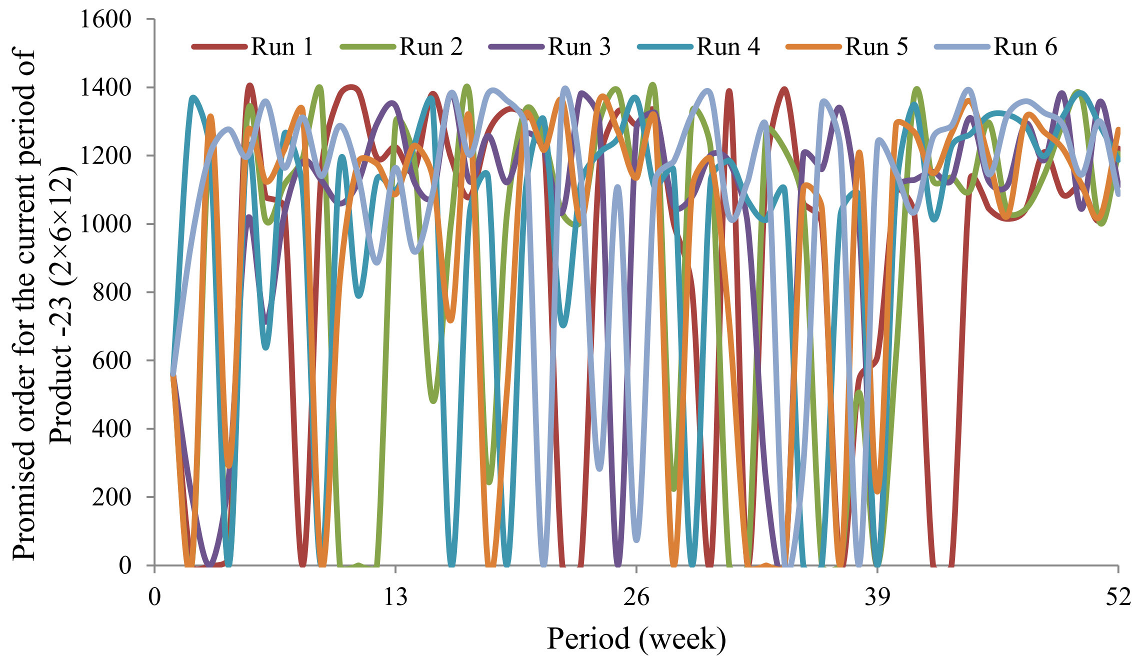

Figure 5.8: Order promised for the current period of product-23 (species 1) of scenario 3 for different simulation runs

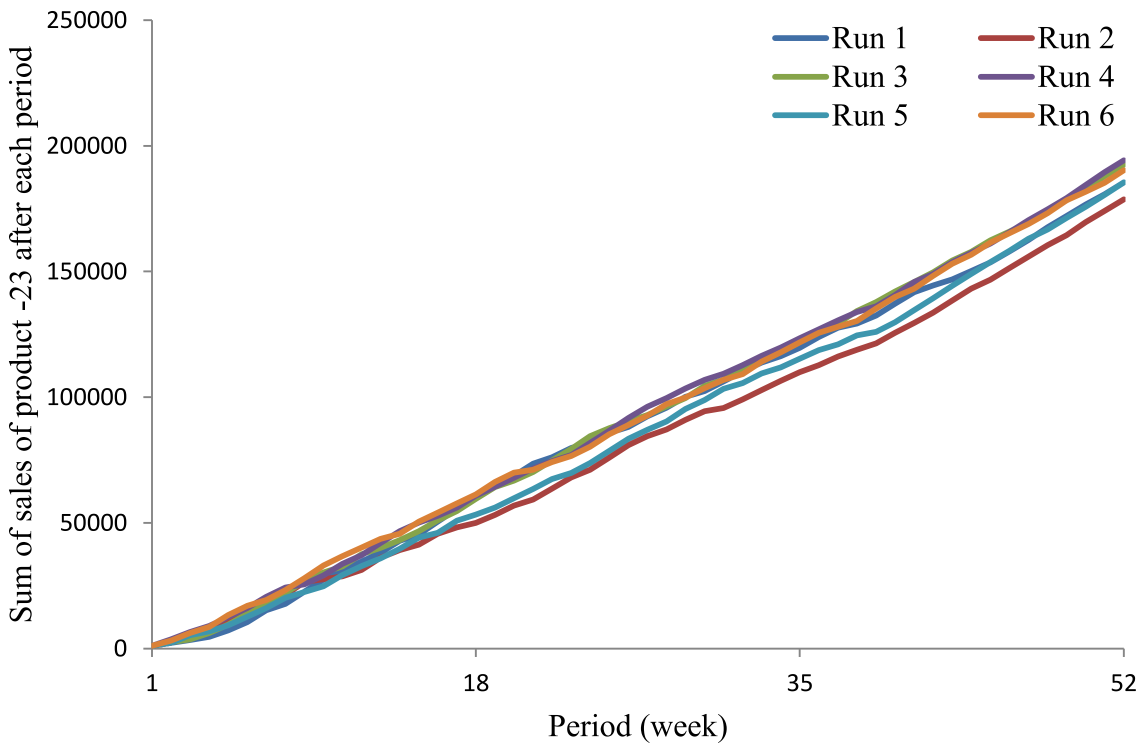

Figure 5.9: Sum of sales of product-23 for different simulation run

Fig. 5.9 shows the total sales at a particular period for the product-23. The mean of total sales in 52 weeks of different runs corresponds to a 95% confidence interval of 187754.26± 27478.22. The same level of total sales is achieved smoothly in different runs through using different sales strategy that is shown in Fig. 5.7.

5.3.4. Scenario 4

In scenario 4, the variation of product price is considered ±20% of the nominal price, and the contract demand is 0.25% of average weekly production capacity. Current week spot-demand varies from 20-30% and the two other spot-demands vary from 30-35% of the average weekly demand (Table 4.1). The Gurobi 5.5 solver produces the following results after solving the 52-week model using the designed solution technique:

Best objective value: $\$2.51 \times 10^7$

Best bound: $\$2.58 \times 10^7$

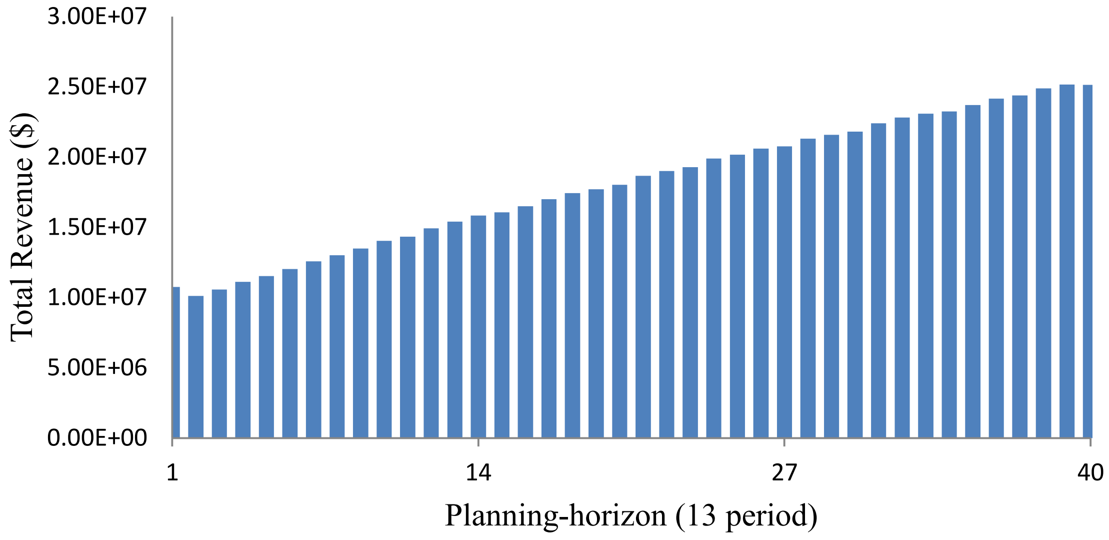

Figure 5.10: Revenue after solving each planning-horizon of scenario 4

The solution time was 13.55 hours. The generated revenue for each planning-horizon for scenario 4 is shown in Fig 5.10. Fig. 5.11 shows the 52 weeks solutions of the production, sales and inventory of product 1 (species 1) of scenario 4. The initial and final inventories are found to the same. No backorder and chips are generated over the period.

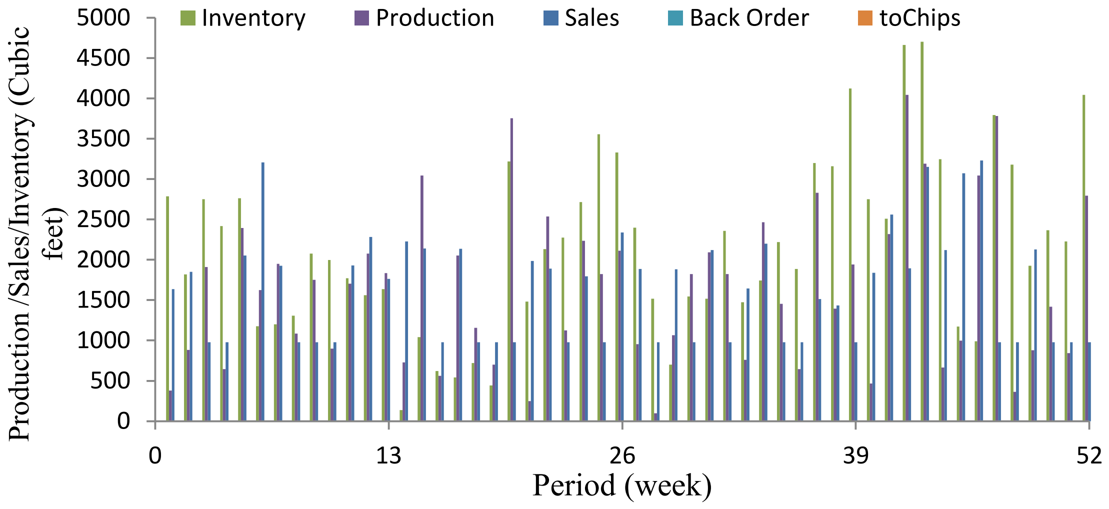

Figure 5.11: Production, sales and inventory of product1 (species 1) of scenario 4

Table 5.8 shows 40 implemented first period solutions of scenario 4- product 1(Species1). The inventory balance equation is met in every period. As an example of period 10 of Table 5.8 using equation 5.1 is given below:

Inventory (10) = Inventory (9) + Production (10)- Sales (10)- Back order (10) - ToChips (10)

= 2077.78+897.34-978.50-0.00-0.00

= 1996.62

Table 5.8: Solution output of scenario 4- product 1(Species1)

Period	Production	Sales	Back order	ToChips	Ending Inventory
0	0	0	0	0	4044.94
1	377.66	1634.88	0	0	2787.73
2	882.64	1849.86	0	0	1820.51
3	1908.92	978.50	0	0	2750.93
4	646.76	978.50	0	0	2419.19
5	2393.57	2051.83	0	0	2760.93
6	1623.25	3208.67	0	0	1175.50
7	1951.18	1927.86	0	0	1198.82
8	1085.21	978.50	0	0	1305.53
9	1750.75	978.50	0	0	2077.78
10	897.34	978.50	0	0	1996.62
11	1705.46	1931.55	0	0	1770.53
12	2076.09	2284.16	0	0	1562.46
13	1836.05	1761.85	0	0	1636.65
14	726.76	2227.40	0	0	136.01
15	3045.56	2139.98	0	0	1041.59
16	559.75	978.50	0	0	622.84
17	2054.29	2135.75	0	0	541.37
18	1157.96	978.50	0	0	720.84
19	701.73	978.50	0	0	444.07
20	3754.66	978.50	0	0	3220.24
21	247.52	1984.88	0	0	1482.88
22	2538.40	1890.45	0	0	2130.83
23	1123.59	978.50	0	0	2275.92
24	2233.83	1795.57	0	0	2714.18
25	1820.84	978.50	0	0	3556.52
26	2114.20	2339.26	0	0	3331.46
27	954.23	1888.23	0	0	2397.46
28	98.39	978.50	0	0	1517.36
29	1066.25	1884.15	0	0	699.45
30	1823.38	978.50	0	0	1544.34
31	2091.79	2119.55	0	0	1516.58
32	1821.16	978.50	0	0	2359.24
33	760.65	1644.57	0	0	1475.32
34	2465.61	2198.01	0	0	1742.92
35	1455.18	978.50	0	0	2219.60
36	643.23	978.50	0	0	1884.33
37	2830.45	1514.38	0	0	3200.40
38	1393.82	1435.55	0	0	3158.68

39	1943.11	978.50	0	0	4123.29
40	466.46	1837.10	0	0	2752.65

Table 5.9 shows variations in product price, demand, the promised order and sales for 40 implemented periods for scenario 4-product 1(species1). The Table 5.9 shows that the sales equation of 5.2 satisfied every period. The contract demand is satisfied by 100%. The current week spot-demand is rejected by 55.93%, the two weeks advanced demand is rejected by 88.44% and the three weeks advanced demand is rejected by 97.46%. As an example of period 6 sales from Table 5.9 is given below:

Sales (6) = Fixed demand (6) + Promised Order (6)+ Promised Order (4) + Promised Order (3)

= 978.50+945.05+0.00+1285.12

= 3208.67

Table 5.9: Demand, order promised and sales for scenario 4-product 1(species1)

Period	v^F	d^F	S^F	v^0	d^0	S^0	v^2	d^2	S^2	v^3	d^3	S^3	Total Sales
1	4.42	743.13	743.13	5.75	891.75	891.75	5.31	966.07	0.00	4.86	1114.69	0.00	1634.88
2	4.42	978.50	978.50	6.45	871.36	871.36	5.15	1187.06	0.00	4.18	1325.49	0.00	1849.86
3	4.42	978.50	978.50	5.12	1141.79	0.00	4.74	1186.89	0.00	5.62	1285.12	1285.12	978.50
4	4.42	978.50	978.50	5.63	938.42	0.00	5.24	1202.37	0.00	4.21	1294.43	0.00	978.50
5	4.42	978.50	978.50	5.73	1073.33	1073.33	5.40	1353.85	0.00	4.98	1261.03	0.00	2051.83
6	4.42	978.50	978.50	6.51	945.05	945.05	4.57	1277.23	0.00	4.65	1269.47	0.00	3208.67
7	4.42	978.50	978.50	6.48	949.36	949.36	5.43	1328.82	0.00	5.59	1229.48	0.00	1927.86
8	4.42	978.50	978.50	5.52	788.86	0.00	4.75	1184.14	0.00	4.52	1215.59	0.00	978.50
9	4.42	978.50	978.50	4.88	1101.13	0.00	4.58	1320.68	0.00	4.07	1360.02	0.00	978.50
10	4.42	978.50	978.50	5.69	1004.88	0.00	6.00	1305.66	1305.66	5.41	1232.71	0.00	978.50
11	4.42	978.50	978.50	6.14	953.05	953.05	4.72	1262.57	0.00	4.76	1268.97	0.00	1931.55
12	4.42	978.50	978.50	5.46	820.26	0.00	5.85	1187.52	723.98	5.40	1331.51	0.00	2284.16
13	4.42	978.50	978.50	6.39	783.35	783.35	4.66	1242.98	0.00	5.31	1302.14	0.00	1761.85
14	4.42	978.50	978.50	5.60	1131.20	524.92	5.67	1199.78	0.00	5.57	1329.99	0.00	2227.40
15	4.42	978.50	978.50	6.05	1161.48	1161.48	4.52	1353.03	0.00	4.40	1201.08	0.00	2139.98
16	4.42	978.50	978.50	5.26	1126.11	0.00	6.02	1275.39	0.00	4.65	1304.17	0.00	978.50
17	4.42	978.50	978.50	6.59	1157.25	1157.25	5.66	1198.77	0.00	5.12	1310.66	0.00	2135.75
18	4.42	978.50	978.50	5.06	1160.23	0.00	5.33	1318.67	0.00	4.96	1349.87	0.00	978.50
19	4.42	978.50	978.50	5.41	840.02	0.00	5.01	1308.53	0.00	5.51	1235.37	0.00	978.50

20	4.42	978.50	978.50	5.20	942.06	0.00	4.90	1192.61	0.00	4.85	1351.10	0.00	978.50
21	4.42	978.50	978.50	6.06	1006.38	1006.38	4.65	1356.21	0.00	5.53	1193.00	0.00	1984.88
22	4.42	978.50	978.50	5.78	911.95	911.95	5.23	1231.34	0.00	5.05	1205.73	0.00	1890.45
23	4.42	978.50	978.50	5.08	878.62	0.00	4.88	1214.17	0.00	4.67	1369.64	0.00	978.50
24	4.42	978.50	978.50	5.93	817.07	817.07	5.93	1360.76	1360.76	4.16	1217.98	0.00	1795.57
25	4.42	978.50	978.50	5.55	1061.81	0.00	5.29	1316.47	0.00	5.10	1340.03	0.00	978.50
26	4.42	978.50	978.50	5.28	964.41	0.00	4.57	1246.11	0.00	5.04	1359.21	0.00	2339.26
27	4.42	978.50	978.50	5.96	909.73	909.73	5.34	1177.09	0.00	4.37	1210.72	0.00	1888.23
28	4.42	978.50	978.50	5.05	986.23	0.00	4.78	1176.48	0.00	4.87	1184.60	0.00	978.50
29	4.42	978.50	978.50	5.66	905.65	905.65	4.87	1327.76	0.00	4.79	1294.83	0.00	1884.15
30	4.42	978.50	978.50	5.04	1096.45	0.00	5.52	1187.85	0.00	4.61	1326.46	0.00	978.50
31	4.42	978.50	978.50	6.24	1141.05	1141.05	5.07	1206.45	0.00	5.16	1274.25	0.00	2119.55
32	4.42	978.50	978.50	5.46	844.64	0.00	5.65	1308.63	1219.51	4.68	1223.87	0.00	978.50
33	4.42	978.50	978.50	6.04	1070.22	666.07	4.65	1346.64	0.00	5.22	1264.74	0.00	1644.57
34	4.42	978.50	978.50	5.30	1017.02	0.00	5.77	1268.72	0.00	5.52	1208.47	0.00	2198.01
35	4.42	978.50	978.50	5.34	946.13	0.00	5.70	1323.09	0.00	5.00	1250.39	0.00	978.50
36	4.42	978.50	978.50	5.69	807.00	0.00	5.86	1330.66	457.05	4.48	1178.12	0.00	978.50
37	4.42	978.50	978.50	5.74	976.52	535.88	5.48	1326.59	0.00	4.52	1190.50	0.00	1514.38
38	4.42	978.50	978.50	5.26	831.27	0.00	4.89	1323.02	0.00	4.13	1364.69	0.00	1435.55
39	4.42	978.50	978.50	5.48	907.08	0.00	5.95	1219.17	759.99	4.88	1195.69	0.00	978.50
40	4.42	978.50	978.50	6.30	858.60	858.60	4.43	1335.93	0.00	4.67	1265.18	0.00	1837.10

5.3.5. Scenario 5

The variation of product price of ±20% of the nominal price is considered in scenario 5. The contract demand is 0.25%, current week spot-demand varies from 30-35% and the two other spot-demands vary from 35-40% of the average weekly production (Table 4.1). Gurobi 5.5 solver produces the following results after solving the 52-week problem using the designed solution technique:

$$\text{Best objective value: } \$2.78\times10^7$$

$$\text{Best bound: } \$2.82\times10^7$$

The solution time is 5.87 hours. Figure 5.12 shows the generated revenue after solving each planning horizon.

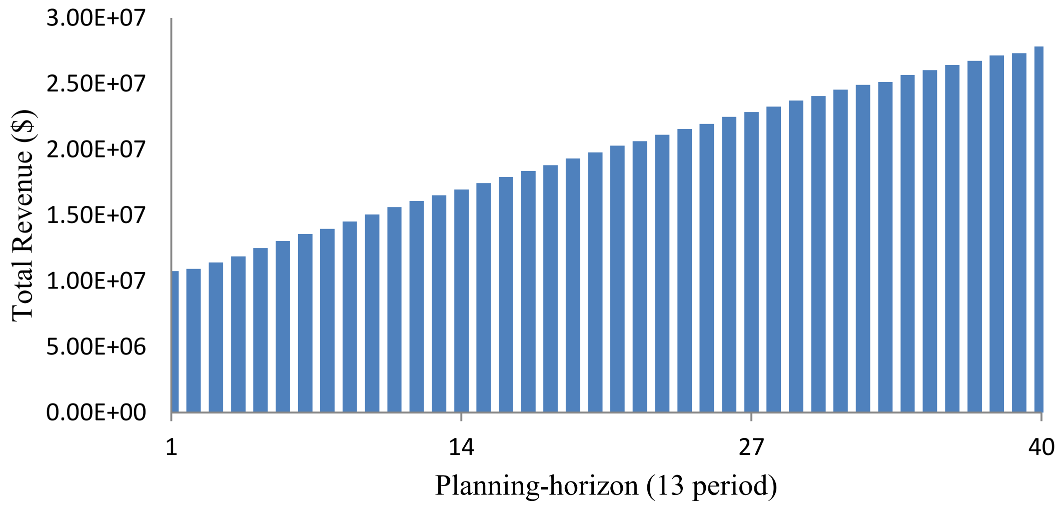

Figure 5.12: Revenue after solving each planning-horizon for scenario 5

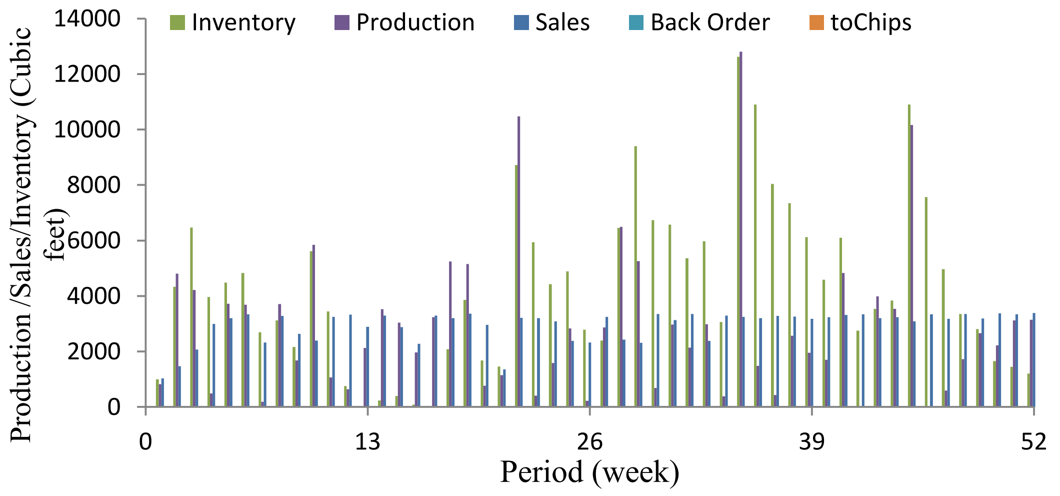

Figure 5.13: Production, sales and inventory of product 4 (species 1) of scenario 5

Figure 5.13 shows the solution of 52-week problem using the rolling planning-horizon technique for product 4 (species1). The initial and final inventories are the same. No backorder and chips are generated over the time.

Table 5.10 shows 40 implemented first period solutions of scenario 5 -product 4(species1). The inventory balance equation 5.1 is met for every period for this scenario. As an example of period 17 of Table 5.10

Inventory (17) = Inventory (16) + Production (17)- Sales (17)- Back order (17) - ToChips (17)

= 79.32+3234.49-3288.66-0.00-0.00

= 25.15

Table 5.10: Solution output of scenario 5 -product 4(species1)

Period	Production	Sales	Back order	ToChips	Ending Inventory
0	0	0	0	0	1199.34
1	821.66	1028.17	0	0	992.84
2	4806.28	1470.51	0	0	4328.62
3	4215.04	2071.65	0	0	6472.00
4	489.61	2995.43	0	0	3966.18
5	3721.09	3201.99	0	0	4485.28
6	3680.83	3334.48	0	0	4831.64
7	188.26	2323.22	0	0	2696.68
8	3707.18	3277.95	0	0	3125.91
9	1677.04	2640.08	0	0	2162.86
10	5845.18	2390.84	0	0	5617.21
11	1069.47	3243.86	0	0	3442.82
12	639.24	3324.42	0	0	757.64
13	2133.22	2890.86	0	0	0.00
14	3526.35	3289.12	0	0	237.24
15	3040.66	2878.74	0	0	399.15
16	1960.84	2280.68	0	0	79.32
17	3234.49	3288.66	0	0	25.15
18	5246.58	3195.71	0	0	2076.02
19	5148.20	3367.24	0	0	3856.98
20	770.28	2952.90	0	0	1674.36
21	1141.78	1357.98	0	0	1458.16
22	10478.73	3213.88	0	0	8723.01
23	410.84	3198.38	0	0	5935.48
24	1583.84	3089.44	0	0	4429.88

25	2834.23	2379.72	0	0	4884.39
26	223.47	2324.09	0	0	2783.77
27	2860.76	3247.90	0	0	2396.63
28	6492.69	2433.08	0	0	6456.25
29	5254.95	2310.02	0	0	9401.17
30	689.16	3353.80	0	0	6736.53
31	2966.55	3130.09	0	0	6572.99
32	2143.89	3352.77	0	0	5364.10
33	2981.98	2380.01	0	0	5966.07
34	389.45	3297.99	0	0	3057.53
35	12807.28	3245.85	0	0	12618.96
36	1480.77	3202.42	0	0	10897.31
37	427.23	3285.42	0	0	8039.12
38	2570.91	3262.02	0	0	7348.01
39	1950.39	3172.37	0	0	6126.02
40	1696.76	3234.22	0	0	4588.57

Table 5.11 shows variations in product price, demand, the promised order and sales for 40 implemented periods for scenario 5 -product 4(species1). The Table 5.11 data satisfy the sales equation of 5.2. The contract demand is satisfied by 100%. The current week spot-demand is rejected by 3.84%, the two weeks advanced demand is rejected by 9.45% and the three weeks advanced demand is rejected by 17.33%. As an example of period 17 sales from Table 5.11 is given below:

Sales (17) = Contract demand (17) + Promised Order (17)+ Promised Order (15) + Promised Order (14)

= 615.25+823.25+968.45+881.71

= 3288.66

Table 5.11: Demand, order promised and sales for scenario 5 -product 4(species1)

Period	v^F	d^F	S^F	v^0	d^0	S^0	v^2	d^2	S^2	v^3	d^3	S^3	Total Sales
1	4.71	467.35	467.35	6.12	560.82	560.82	5.65	607.55	607.55	5.18	701.02	701.02	1028.17
2	4.71	615.25	615.25	5.73	855.26	855.26	5.10	934.71	934.71	5.23	904.14	904.14	1470.51
3	4.71	615.25	615.25	6.31	848.85	848.85	5.78	871.64	871.64	5.58	980.30	980.30	2071.65

4	4.71	615.25	615.25	6.05	744.45	744.45	5.14	892.31	892.31	5.05	967.89	0.00	2995.43
5	4.71	615.25	615.25	6.71	810.95	810.95	5.95	905.96	905.96	4.92	964.49	959.01	3201.99
6	4.71	615.25	615.25	6.81	846.62	846.62	6.29	907.04	907.04	4.51	887.38	254.39	3334.48
7	4.71	615.25	615.25	6.81	802.01	802.01	5.37	948.18	948.18	4.61	947.10	0.00	2323.22
8	4.71	615.25	615.25	6.26	796.66	796.66	5.17	914.33	914.33	5.23	980.77	980.77	3277.95
9	4.71	615.25	615.25	6.24	822.27	822.27	5.78	869.64	869.64	4.54	922.92	922.92	2640.08
10	4.71	615.25	615.25	6.79	861.26	861.26	5.18	956.14	956.14	4.51	970.00	970.00	2390.84
11	4.71	615.25	615.25	6.91	778.20	778.20	5.17	899.19	899.19	5.36	877.47	877.47	3243.86
12	4.71	615.25	615.25	5.85	830.11	830.11	6.16	946.26	946.26	4.86	866.14	866.14	3324.42
13	4.71	615.25	615.25	5.56	858.71	406.42	5.22	905.37	604.32	4.28	950.34	788.67	2890.86
14	4.71	615.25	615.25	5.73	850.14	850.14	5.56	876.76	876.76	4.83	881.71	881.71	3289.12
15	4.71	615.25	615.25	6.50	793.03	793.03	6.29	968.45	968.45	6.03	880.93	880.93	2878.74
16	4.71	615.25	615.25	5.46	780.25	0.00	5.74	873.52	873.52	5.41	955.28	955.28	2280.68
17	4.71	615.25	615.25	5.38	823.25	823.25	6.33	970.15	970.15	4.87	873.28	873.28	3288.66
18	4.71	615.25	615.25	5.47	826.00	826.00	5.23	921.92	723.43	4.77	956.42	0.00	3195.71
19	4.71	615.25	615.25	6.47	826.57	826.57	5.78	975.38	0.00	5.50	911.17	911.17	3367.24
20	4.71	615.25	615.25	5.64	740.93	740.93	5.50	925.60	925.60	5.09	878.31	878.31	2952.90
21	4.71	615.25	615.25	5.67	742.73	742.73	6.03	908.73	908.73	4.78	951.56	951.56	1357.98
22	4.71	615.25	615.25	5.28	761.87	761.87	5.29	876.62	697.22	5.79	875.89	875.89	3213.88
23	4.71	615.25	615.25	5.77	796.08	796.08	5.52	915.88	27.34	4.43	873.52	0.00	3198.38
24	4.71	615.25	615.25	6.59	825.40	825.40	6.46	918.66	918.66	5.22	903.74	903.74	3089.44
25	4.71	615.25	615.25	5.76	861.24	861.24	5.55	874.97	874.97	4.39	896.64	0.00	2379.72
26	4.71	615.25	615.25	5.41	790.18	790.18	5.49	982.60	982.60	6.04	876.48	876.48	2324.09
27	4.71	615.25	615.25	6.18	853.93	853.93	5.17	925.82	0.00	5.98	900.50	900.50	3247.90
28	4.71	615.25	615.25	5.78	835.23	835.23	6.17	980.48	980.48	4.99	876.02	876.02	2433.08
29	4.71	615.25	615.25	6.32	818.29	818.29	6.38	882.45	882.45	5.02	968.32	968.32	2310.02
30	4.71	615.25	615.25	6.84	857.57	857.57	4.75	955.47	955.47	4.37	892.87	0.00	3353.80
31	4.71	615.25	615.25	6.38	756.36	756.36	4.75	936.13	936.13	4.79	970.40	970.40	3130.09
32	4.71	615.25	615.25	6.08	813.73	813.73	6.58	972.87	972.87	6.01	922.19	922.19	3352.77
33	4.71	615.25	615.25	6.60	828.64	828.64	5.53	959.45	959.45	5.69	865.33	865.33	2380.01
34	4.71	615.25	615.25	5.89	739.46	739.46	5.15	910.24	910.24	6.05	966.48	966.48	3297.99
35	4.71	615.25	615.25	6.42	748.96	748.96	5.11	959.37	959.37	4.29	875.21	875.21	3245.85
36	4.71	615.25	615.25	6.82	811.59	811.59	5.45	915.71	915.71	4.48	863.33	863.33	3202.42
37	4.71	615.25	615.25	6.04	744.32	744.32	5.80	916.43	916.43	4.39	970.77	970.77	3285.42
38	4.71	615.25	615.25	6.69	855.85	855.85	4.99	886.97	886.97	4.77	978.97	978.97	3262.02
39	4.71	615.25	615.25	5.74	777.35	777.35	6.16	938.41	938.41	5.01	982.56	982.56	3172.37
40	4.71	615.25	615.25	6.81	761.23	761.23	5.56	982.47	982.47	6.05	878.13	878.13	3234.22

5.4. DISCUSSIONS

Five different scenarios were studied to test the sensitivity of the designed model. Different product price demand patterns were also considered in five different scenarios. None of the scenarios generate any backorder because the order was promised within the production capacity. The initial and final inventories were thus the same for all scenarios. In the second scenario, as the inventory went down to zero, it did not promise for any variable demand for period 12.

In the third scenario, chips were produced in different periods from the lumber inventory. Although the lumber prices were higher than the chip price, constraint 4.15 limited the total lumber inventory capacity of the sawmill. As there was no capacity limit for specific product inventory, the optimization model decided which inventory product would be converted to chips.

In period 20, as inventory was low, the model did not accept the coming week's spot-demands. Instead, it accepted relatively higher price demands that would be delivered after 2 weeks. It also accepted an order that would be delivered three weeks later, though the price of the product is lower than that of the coming week's delivery product price but from the production planning model it is anticipated that promising order for a demand that would be delivered in three weeks is feasible. The same situation was also seen in period 24.

The scenario 3 is simulated for 6 different runs and it is found that the mean of total revenue corresponds to a 95% confidence interval of $\$(2.40\pm0.02)\times10^7$. The revenue is generated using different order promising strategies based on the demand and prices for that scenario. The sales in each period for different runs of scenario 3 are shown in Fig 5.7. The mean of total sales in 52 weeks of different runs corresponds to a 95% confidence interval of (187754.26 ± 27478.22) ft^3. Thus, although the order promising, rolling planning horizon technique achieves the same total result over 52 weeks, much different strategies of production and order promising are required in order to cope with the uncertainty.

In scenario 4, it is shown that the order is promised for higher price product in period 3. The computational time of scenario 5 is significantly less than the other scenarios. This is because this scenario was run on a different computer having 8 GB ram and quard core 3.40 GHz processor speed. All other scenarios run on a computer having 4 GB memory and duel core 3.17 GHz processor speed. Gurobi 5.5 is designed to take advantage of extra processors.

5.5. CONCLUSIONS

This chapter described the dynamic rolling planning-horizon simulation solution technique. This new approach allow us to conveniently carry out the many runs needed to evaluate the rolling planning horizon approach in a simulation. The sensitivity of the designed model was also presented by illustrating various scenarios by looking at how the production requirements are met for certain products.The solution code written in Python 2.7 for Gurobi 5.5 solver is given in Appendix C for all scenarios. In the next chapter, the research conclusion and future work are described.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This thesis describes the modeling framework of an order-promising methodology and production-planning technique for the sawmill industry. Order promising is a continuous process with ongoing sawmilling operations. Two types of customer demands are considered: contract demands and spot demands. Contract demands are submitted by a corporate customer and are intended to be delivered over a period of time. Three types of spot demands are considered in this thesis: promised demands to be filled current week; promised demands to be filled within two weeks; and promised demands to be filled within three weeks. A production-planning model that includes the scheduling of campaigns in such a way that the promised order can be delivered on time, is also included with the order-promising model.

In this thesis, we considered 2 different species of logs that were classified into 8 different classes. Each of the species produced 41 different dimensional lumber products. We considered 46 price lists and over 6,000 cutting patterns that produced 366 different campaigns. Log defects were not considered in generating cutting patterns and all output products were considered to be of the same quality. The order promising and production-planning length was 52 weeks, and the problem was solved using the Gurobi 5.5 solver and the rolling planning-horizon technique.

The newly developed solution algorithm can be seen as allowing one to solve a very large-scale optimization problem. For example, the 52-week model contains 43,934 continuous variables, as two types of setup time (campaign and class) are added with the problem, which includes more than 19,448 binary variables. The problem was attempted using a PC with 4 GB memory and 3.17 GHz processor speed. However, the computer ran out of memory after 10.39 hours. Here we are not just solving a 52 week problem but simulating how the solution process would work as the random information becomes available weekl.

The developed solution techniques solve the problem as a 13-week problem each time in 40 steps. This eventually reduces the continuous and binary variables to 10,454 and 4,862, respectively for each planning horizon. The total number of solution runs is 40 and took in between 13 and 16 hours, although faster results were obtained in scenario five on more capable computer. More to the point, the designed solution technique allows for updating of the parameter values before starting each 13-week planning-horizon. However, the new solution technique could be applied to any large scale dynamic optimization problem.

The combination of order promising and production-planning is the novelty of this work. In the literature, the order-promising methodology has been investigated (Azevedo et al., 2012), but without consideration of the production planning. Other researchers developed a production-planning strategy (Zanjani et al., 2009a; Zanjani et al., 2009b; Nourelfath, 2010) targeting fulfilling a demand. In this thesis, we maximize the total revenue via promising order and make a production schedule to fulfill the order. Chapter 3 describes the generation of new campaigns that resulted in a reduction in solution time. We also demonstrated that a new pricelist to generate the new campaigns can be usefully developed from the product shadow price, similar to the column generation technique discussed by Maness and Nelson. From the column-generation technique, more suitable campaigns allow the solver more options to schedule the campaign relatively in shorter time.

In Chapter 4, we formulated an MIP model for order promising and campaign scheduling. The objective is to maximize the revenue using available resources and ensure the proper utilization of the resources. Four different types of demand lead times were considered in the model. The model determines how many orders will be accepted within the sawmill capacity and other constraints. It also determines the campaign schedule to fulfill the promised order.

The dynamic solution technique, developed to carry out process of simulating the rolling planning horizon application of the model developed in Chapter 4, is described in Chapter 5. The new technique allows us to dynamically update the prices and demands for each product in each period within a single run in the Gurobi Python environment. The scenario 3 simulated 6 different runs. The variation in the total revenue obtained in 6 runs was found to be small (having the 95% confidence interval for the mean (\$$2.38\times10^7$, \$$2.42\times10^7$)).. The variation in total sales is also found to be small (having the 95% confidence interval for the mean (160276 ft.3, 215232 ft.3)). However, the order acceptance strategies and campaign production strategies needed to obtain these consistent results were of necessity greatly different. The results shown in Chapter 5 demonstrate the effectiveness with which the rolling planning horizon approach can be simulated as a tool of investigation.

In a real sawmill, we would have available scanners information for a large number of logs. Thus simulating the logs would be unnecessary. An interesting area of research would be to investigate various sorting strategies to generate alternative campaigns using the market prices and shadow prices as pricelists. What we have demonstrated here is the technical capability to do this at large-scale.

The idea of shadow price based campaign generation can be investigated in several ways, such as by adding more shadow prices to the existing pricelist, or by removing the old pricelist from the existing pricelist and using only the new shadow price in the pricelist, to generate campaigns and see the effect on solution time.

Using the revenue management technique is one of the options for order promising. The general objective of revenue management techniques is to maximize profit over the planning-horizon by deciding whether to accept or reject a given order by anticipating future profitable orders. Customers are classified into different segments, based on their willingness to pay. An allocation limit is defined for each class of customer. Different authors segment customers in different ways. For instance, Harris and Pinder (1995) segmented customers into two classes: class 1 customers were able to place their orders in advance, and class 2 customers placed their orders on a last-minute demand basis. Azevedo et al. (2012) segmented customers into 3 classes: class1 were price-sensitive customers who were willing to pay around the market price; class2 customers aimed to disburse a lower-than-market price; and class3 were customers to whom a product might be sold at a premium price.

In future research, the segmentation of customer classes can be done after consultation with the local sawmill industry. The booking limits of a product for each segment of customers are also an issue for future research, as is the pricing of products for each customer segment. Investigating the suitability of offering variable product prices according to demand lead-time (Venkatadri et al., 2006) for the lumber industry could likewise be interesting research.

On the other hand, to fulfill the promised demand, an optimal production schedule is required. A real-time production control model can be developed, consisting of two parts: 1) a campaign- scheduling model; and 2) a pricelist-based control model. The campaign-scheduling problem can be modeled separately and coupled with the sawmill control problem. The campaign- scheduling model schedules a campaign based on real-time data, and the pricelist-based feedback control will achieve the production target within the timeframe determined by the campaign- scheduling model.

The emphasis in this thesis has been mostly technical. We have demonstrated the ability to use shadow prices to generate better campaigns. We have created a framework within which the rolling planning horizon approaches can be simulated for a model which includes both the complexity of sawmill campaigns that produce multiple output products and an order promising framework where demands that are available are accepted or rejected based on current inventories current prices and anticipated future demands and price. A full investigation of how these technical capabilities can lead to better planning in real sawmills remains to be accomplished.

REFERENCES

1. Azevedo, R. C., D'Amours, S., Rönnqvist, M. (2012): Advances in Profit-Driven Order Promising for Make-To-Stock Environments – A Case Study With a Canadian Softwood Lumber Manufacturer. International Journal of Production Economics (submitted).
2. Bettoni, L. (2013) GUSEK (GLPK Under Scite Extended Kit). Accessed at http://gusek.sourceforge.net/gusek.html.
3. Bowater Mersey Oakhill Sawmill (2012): Scanned Logs Data. Supplied to Eldon Gunn by Hans Peterson, Halifax, Canada.
4. Chen, C.-Y., Zhao, Z.-Y., and Ball, M. O. (2001): Quantity and Due Date Quoting Available- To-Promise, Information Systems Frontiers 3(4), pp. 477–488.
5. Chen , J.-H. and Chen , C.-T. (2009): Using Mathematical Programming on Two-Phase Order Promising Process With Optimized Available-To-Promise Allocation Planning. International Journal of the Computer, the Internet and Management, Vol. 17. No.3, pp. 25 -40.
6. Gunn, E., MacDonald, C., Saatadayar, S., and Sohrabi, P., (2013): "Sawmill Manufacturing: Moving Towards Lean" , NSERC/FPInnovations Forest Value Chain Optimization Network Webinar of April 3, 2013.
7. Fan, L. and Chen, X. (2008): Order Acceptance and Capacity Allocation Policies based on Revenue Management, International Conference on Information Management, Innovation Management and Industrial Engineering, pp. 248-251.

8. Gaudreault, J., Forget, P., Frayret, J., Rousseau, M. A. and D'Amours, S. (2009): Distributed Operations Planning in the Lumber Supply Chain: Models and Coordination, Technical report, CIRRELT Working Paper CIRRELT.
9. Gurobi, (2013) Gurobi Optimization. Website: http://www.gurobi.com/
10. Harris, F. H., deB. and Pinder, J. P. (1995): A Revenue Management Approach to Demand Management and Order Booking in Assemble-To-Order Manufacturing, Journal of Operations Management, Vol. 13, No.4, pp. 299-309.
11. Ivanescu, C. V., Fransoo, J. C., Bertrand, J. W. M. (2002): Makespan Estimation and Order Acceptance in Batch Process Industries When Processing Times are Uncertain, OR Spectrum 24: pp. 467–495.
12. Kilic, O. A., Donk, D. P. V., Wijngaard, J., and Tarim , S. A. (2010): Order Acceptance in Food Processing Systems with Random Raw Material Requirements, OR Spectrum 32: pp. 905–925.
13. MacDonald, C., Gunn, E. A., Sohrabi, P., Saadatyar, S. (2013): Developing Sawmill Campaigns using Python, Technical Report, Department of Industrial Engineering, Dalhousie University.
14. Makhorin, A. (2013). GLPK (GNU linear programming kit). Accessed at http://www.gnu.org/software/glpk/ .
15. Maness, T.C. and Norton, S. E. (2002): Multiple-period combined optimization approach to forest production planning. Scandinavian Journal of Forest Research 17: pp. 460-471.
16. 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, Proceeding 23rd CIRP International Seminar on Manufacturing Systems, Nancy (France).

17. Meyr, H., (2007): Clustering Methods for Rationing Limited Resources. In: Lars Mönch, Giselher Pankratz (Eds.), Intelligente Systeme zur Entscheidungsunterstützung. Multikonferenz Wirtschaftsinformatik, München, SCS Publishing House e.V, San Diego et al., pp. 19–31.
18. Meyr, H., (2008): Customer Segmentation, Allocation Planning and Order Promising in Make-To-Stock Production. OR Spectrum, 31(1), pp. 229-256.
19. Makorin, A. (2010): Modeling Language GNU MathProg: Language Reference for GLPK Version 4.45 (DRAFT, December 2010), Free Software Foundation, Inc Boston, MA, USA.
20. Nourelfath, M. (2010): Service Level Robustness in Stochastic Production Planning under Random Machine Breakdowns, Technical Report, CIRRELT.
21. Pibernik, R. (2005): Advanced Available-To-Promise: Classification, Selected Methods and Requirements for Operations and Inventory Management, International Journal of Production Economics Vol. 93-94, pp. 239-252.
22. Pibernik, R. and Yadav, P., (2009): Inventory Reservation and Real-Time Order Promising in A Make-To-Stock System. OR Spectrum, 31, pp. 281-307.
23. Raaymakers, W. H. M., Bertrand, J. W. M., Fransoo, J. C. (2000a): The Performance of Workload Rules for Order Acceptance in Batch Chemical Manufacturing, Journal of Intelligent Manufacturing 11: pp. 217–228.

24. Raaymakers, W. H. M., Bertrand, J. W. M., Fransoo, J. C. (2000b): Using Aggregate Estimation Models for Order Acceptance in A Decentralized Production Control Structure for Batch Chemical Manufacturing, IIE Transactions 32, pp. 989-998.

25. Raaymakers, W. H. M. and Fransoo, J. C. (2000c): Identification of Aggregate Resource and Job Set Characteristics for Predicting Job Set Makespan in Batch Process Industries. International Journal of Production Economics, Vol. 68, No.2, pp. 137-149.

26. Raaymakers, W. H. M. and Hoogeveen, J. A. (2000): Scheduling Multi-Purpose Batch Process Industries with No-Wait Restrictions by Simulated Annealing, European Journal of Operational Research, Vol. 126, No.1, pp. 131-151.

27. Saadatyar, S. (2012): Medium Term Production Planning and Campaign Scheduling for Sawmill, M.A.Sc Thesis, Department of Industrial Engineering, Dalhousie University, Canada.

28. Sohrabi, P. (2012), A Three-stage Control Mechanism for the Lumber Production Process of a Sawmill Based on a Powers-of-two Modelling Approach. M.A.Sc Thesis, Department of Industrial Engineering, Dalhousie University, Canada.

29. Venkatadri, U., Srinivasan, A., Montreuil, B., and Saraswat, A. (2006): Optimization-based Decision Support for Order Promising in Supply Chain Networks. International Journal of Production Economics, Vol. 103, No. 1, pp. 117-130.

30. Venkatadri, U., Wang, S., and Saraswat, A. (2008): Promising Orders in Supply Chain Networks. International Journal of Industrial and Systems Engineering Vol. 3, No. 2, pp. 211-228.

31. Wang, J., Yang, J.-Q. and Lee, H. (1994): Multicriteria Order Acceptance Decision Support in Over-Demanded Job Shops: A Neural Network Approach, Mathematical and Computer Modeling, Vol. 19, No. 5, pp. 1-19.

32. Zanjani, K. M., Nourelfath, M., and Ait-Kadi, D. (2009a): A Stochastic Programming Approach for Production Planning with Uncertainty in The Quality of Raw Materials: A Case in Sawmills, Technical Report, CIRRELT.

33. Zanjani, K. M., Nourelfath, M. and Ait-Kadi, D. (2009b): A Multi-Stage Stochastic Programming Approach for Production Planning with Uncertainty in The Quality of Raw Materials and Demand, Technical Report, CIRRELT.

34. Zanjani, K. M., Ait-Kadi, D., and Nourelfath, M. (2010): Robust Production Planning in A Manufacturing Environment with Random Yield: A Case in Sawmill Production Planning, European Journal of Operational Research, 201(3), pp. 882-891.

35. Zhao, Z., Ball, M. O., and Kotake, M. (2005): Optimization-based Available-To-Promise with Multi-Stage Resource Availability. Annals of Operations Research 135, pp. 65-85.

APPENDIX A: GENERATION OF SHADOW PRICE

The model coded in GLPK produces *.mod file and converted into *.lp file to solve by using Gurobi 5.5 solver. As the MIP solution does not produce shadow price, the model is optimize first (m.optimize) and fixed the integer variable values (m.fixed) and solved the model again (mfix.optimize) as a linear programming model. The product shadow prices are collected from the inventory constraints.

```
m = read("C:/Documents and Settings/Dalhousie/Desktop/Research_Sharif/Production
run for 13 weeks_120Campaign/blv.lp")
m.optimize()
mfix = m.fixed()
mfix.optimize()
con=mfix.getConstrs()
shadowPrice=[]
for i in range(1066,2106):
    x= con[i].getAttr('Pi')
    nm= con[i].getAttr('ConstrName')
    data=(i,nm,x)
    shadowPrice.append(data)
    shadowPrice.append(data)
    shadowPrice.append(data)

import csv
b = open('C:/Documents and Settings/Dalhousie/Desktop/Research_Sharif/Production
run for 13 weeks_120Campaign/shadowPrice.csv', 'wb')
a = csv.writer(b)
a.writerows(shadowPrice)
b.close()
```

APPENDIX B: GLPK CODE

The model is coded in GLPK environment. The model objective is to maximize the total revenue. The objective function, equation 4.1, is coded in such a way that every variable upper and lower bound and the variable coefficient is accessiable through Gurobi solver 5.5. Every constraints name (constraints 4.1-4.23) is defined so that any modification is needed for a specific constraint can be accessiable by name. The initial parameter values are read from *.csv file

```
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 spieces */

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 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 Fccc dimen 3;
set vccc1 dimen 3;
set vccc2 dimen 3;
set vccc3 dimen 3;
set ppp dimen 3;
set ppp1 dimen 3;
set ppp2 dimen 3;
set ppp3 dimen 3;
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 iMax{i in I, s in S, t in T}, >= 0;
#/* the maximum allowable inventory of product i in period t */

#table tab_invmax IN "CSV" "Data.csv" :
#  mx <- [Product, Species, Period], iMax ~ InvMax;

param Fcost{i in I, s in S, t in T}, >= 0;
/* Selling price for product i in period t at level L*/

table tab_costmx IN "CSV" "demand1.csv" :
  Fccc <- [Product, Species, Period], Fcost ~ fprice;

param Vcost1{i in I, s in S, t in T}, >= 0;
/* Selling price for product i in period t at level L*/

table tab_costmx IN "CSV" "demand1.csv" :
  vccc1 <- [Product, Species, Period], Vcost1 ~ vprice1;

param Vcost2{i in I, s in S, t in T}, >= 0;
/* Selling price for product i in period t at level L*/
```

```
table tab_costmx IN "CSV" "demand1.csv" :
  vccc2 <- [Product, Species, Period], Vcost2 ~ vprice2;

param Vcost3{i in I, s in S, t in T}, >= 0;
/* Selling price for product i in period t at level L*/

table tab_costmx IN "CSV" "demand1.csv" :
  vccc3 <- [Product, Species, Period], Vcost3 ~ vprice3;

param fd{i in I, s in S, t in T}, >= 0;
/* product i upper bound for level L at period t */

table tab_Lmax IN "CSV" "demand1.csv" :
  ppp <- [Product, Species, Period], fd ~ fdemand;

param vd1{i in I, s in S, t in T}, >= 0;
/* product i upper bound for level L at period t */

table tab_Lmax IN "CSV" "demand1.csv" :
  ppp1 <- [Product, Species, Period], vd1 ~ vdemand1;

param vd2{i in I, s in S, t in T}, >= 0;
/* product i upper bound for level L at period t */

table tab_Lmax IN "CSV" "demand1.csv" :
  ppp2 <- [Product, Species, Period], vd2 ~ vdemand2;

param vd3{i in I, s in S, t in T}, >= 0;
/* product i upper bound for level L at period t */

table tab_Lmax IN "CSV" "demand1.csv" :
  ppp3 <- [Product, Species, Period], vd3 ~ vdemand3;

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}, >= 0;
/* amount of product i sold in market level L at period t */

var S1{i in I, s in S, t in T}, >= 0;
/* amount of product i sold in market level L at period t */

var S2{i in I, s in S, t in T}, >= 0;
/* amount of product i sold in market level L at period t */

var S3{i in I, s in S, t in T}, >= 0;
/* amount of product i sold in market level L at period t */

var S4{i in I, s in S, t in T}, >= 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 TI}, >= 0;
/* Lost Invenotry amount */

#var extraInv{i in I, s in S, t in T}, >=0;
#/* Extra Inventory 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 < 41}: Inv[i,s,t]-lostInv[i,s,t]= Inv[i,s,t-1]-lostInv[i,s,t-
1]+ P[i,s,t]- Sale[i,s,t]-toChips[i,s,t];

s.t. inventoryChips{s in S, t in T}: Inv[41,s,t]-lostInv[41,s,t] = Inv[41,s,t-1]-
lostInv[41,s,t-1]+P[41,s,t]- Sale[41,s,t] +sum{i in I: i<41}(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;

/*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<41}(Inv[i,s,t])<= 4*ProdRate;

/* 8. Starting stock */

s.t. startlostInv{i in I, s in S}: lostInv[i,s,0]=0;#inlost[i,s];

s.t. starttoChips{i in I, s in S}:toChips[i,s,0] = 0;

/*9- Market maximum price */
s.t. ContractDemand{i in I, s in S, t in T}: S1[i,s,t]=fd[i,s,t];# added new constrain

s.t. SpotDemandLead0{i in I, s in S, t in T}: S2[i,s,t]<=vd1[i,s,t]; # added new constrain

s.t. SpotDemandLead2{i in I, s in S, t in T}: S3[i,s,t]<=vd2[i,s,t]; # added new constrain

s.t. SpotDemandLead3{i in I, s in S, t in T}: S4[i,s,t]<=vd3[i,s,t]; # added new constrain

s.t. totalSale{i in I, s in S, t in T:t<3}: Sale[i,s,t]=S1[i,s,t]+S2[i,s,t];# added new constrain

s.t. totalSale1{i in I, s in S, t in T:t=3}: Sale[i,s,t]=S1[i,s,t]+S2[i,s,t]+S3[i,s,t-2];

s.t. totalSale2{i in I, s in S, t in T:t>3}: Sale[i,s,t]=S1[i,s,t]+S2[i,s,t]+S3[i,s,t-2]+S4[i,s,t-
3];# added new constrain

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

s.t. endInv{i in I, s in S,t in T:t>12}: Inv[i,s,t] >= 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]) <= 46;

s.t. maxcls{t in T}: sum{c in C} (Z[c,t]) <= 8;

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

maximize obj: sum{i in I,s in S, t in T:t<3} (Fcost[i,s,t]*S1[i,s,t]+Vcost1[i,s,t]*S2[i,s,t]-
h[i,s,t] * Inv[i,s,t] - PC * lostInv[i,s,t])+sum{i in I,s in S, t in T:t=3}
(Fcost[i,s,t]*S1[i,s,t]+Vcost1[i,s,t]*S2[i,s,t]+Vcost2[i,s,t]*S3[i,s,t-2]- h[i,s,t] * Inv[i,s,t] -
PC * lostInv[i,s,t])+sum{i in I,s in S, t in T:t>3}
(Fcost[i,s,t]*S1[i,s,t]+Vcost1[i,s,t]*S2[i,s,t]+Vcost2[i,s,t]*S3[i,s,t-
2]+Vcost3[i,s,t]*S4[i,s,t-3] - 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;

data;

param m := 41;
param n := 366;
param pr := 52;
param cNo := 8;
param ProdRate := 128205 ;
/* weekly production Rate */
/*param Factor := 0.0242646;
 ft^3 to m^3 /1.167 (m^3/tone) */
end;
```

APPENDIX C: SOLUTION TECHNIQUE

The model is read in two variable names m and m1. All modifications are made on ‘m’ and m1 is used to reset the values of variables upper and lower bound, objective co-efficient, constraint sense and right hand side value. The solution technique is described using the steps.

Steps	Operations	Explanation

1	m = read("C:/Documents and Settings/Orderpromising.lp")
	m1 = read("C:/Documents and Settings /Orderpromising.lp")
		Read model in two variables name

2	def var_values(m,var3,var2,period)	To eliminate variables (14-52 period)
3	def cons_RhsValues(m,con3,con2,con1,period)	To relax constrains (14-52 period)
4	m.optimize()	For solving model
5	solution_update(m,var3,var2,solution_period[i])	For updating implementable solution
6	activareNextPeriodVariable(m,m1,var3,var2,period[0])	Activate period 14 variables and objective coefficient comparing with m1
7	activareNextPeriodConstrain(m,m1,con3,con2,con1,period[0])	Activate period 14 constraints comparing with m1
8	relaxedPreviousPeriodEndInventoryConstrain(m,con,conperiod[0])	Period 13, now end inventory is on period 14
9	setVariabkeDemand(m,demand,newDemandPeriod)	For updating demand- period 2-14
10	setVariabkePrice(m,price,newDemandPeriod)	For updating price- period 2-14
11	m.update()	For updating model

12	m.optimize()	For solving problem
13	Repeat from 5	

SCENARIO 1

```
#*******************************************************************
#*******************************************************************

def var_values(m,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==period[j]: #Take care on varPeriod[]
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
         for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==period[j]: #Take care on varPeriod[]
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
   return m.update()

#*******************************************************************
#*******************************************************************

def cons_RhsValues(m,con3,con2,con1,period):  #To relax constrains
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',50000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-5000000000000)

   return m.update()


#*******************************************************************
#********************************************************************

def relaxedPreviousPeriodEndInventoryConstrain(m,con3,period):  #To relax end
inventory constrains
   cons_list=m.getConstrs()
   #print '---------------',con3,period
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      #print '-------------------',name[0],conPeriod1[2]
      if name[0]==con3 and conPeriod1[2]==period: #Take care on conPeriod[]
         #print cons_list[i].getAttr('ConstrName'),conPeriod1[2]
         cons_sense=cons_list[i].getAttr('Sense')
         if cons_sense=='<':
            cons_list[i].setAttr('RHS',5000000000000)
         elif cons_sense=='>':
            cons_list[i].setAttr('RHS',-5000000000000)
         else:
            cons_list[i].setAttr('Sense','>')
            cons_list[i].setAttr('RHS',-50000000000)
   return m.update()



#*****************************************************************
#******************************************************************


def solution_update(m,var3,var2,solution_period): # To update solution
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for k in range(len(var3)):
         if name[0]==var3[k] and varPeriod[2]==solution_period: #Take care on
varPeriod[]
            var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==solution_period: #Take care on
varPeriod[]
            var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
   return m.update()

#*****************************************************************
#*****************************************************************

def activareNextPeriodVariable(m,m1,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   var_list1=m1.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for k in range(len(var3)):
         if name[0]==var3[k] and varPeriod[2]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print var_list[i].getAttr('varName')
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print r
   return m.update()

#*******************************************************************
#*******************************************************************

def activareNextPeriodConstrain(m,m1,con3,con2,con1,period):  #To add next period
constrain
   cons_list=m.getConstrs()
   cons_list1=m1.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
      for k in range(len(con3)):
         if name[0]==con3[k] and conPeriod1[2]==period: #Take care on conPeriod[]
            cons_sense=cons_list1[i].getAttr('Sense')
            cons_RHS=cons_list1[i].getAttr('RHS')
            cons_list[i].setAttr('Sense',cons_sense)
            cons_list[i].setAttr('RHS',cons_RHS)
            #print cons_list[i].getAttr('RHS')
      for k in range(len(con2)):
         if name[0]==con2[k] and conPeriod1[1]==period: #Take care on conPeriod[]
            cons_sense=cons_list1[i].getAttr('Sense')
            cons_RHS=cons_list1[i].getAttr('RHS')
            cons_list[i].setAttr('Sense',cons_sense)
            cons_list[i].setAttr('RHS',cons_RHS)
      for k in range(len(con1)):
         if name[0]==con1[k] and conPeriod[1]==period: #Take care on conPeriod[]
            cons_sense=cons_list1[i].getAttr('Sense')
            cons_RHS=cons_list1[i].getAttr('RHS')
            cons_list[i].setAttr('Sense',cons_sense)
            cons_list[i].setAttr('RHS',cons_RHS)
            #print cons_list[i].getAttr('RHS')
   return m.update()


#*****************************************************************
#*****************************************************************
def setVariabkeDemand(m,demand,newDemandPeriod):  #To add next period constrain
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      for j in range(len(newDemandPeriod)):
         if name[0]==demand[0] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            #print cons_list[i].getAttr('ConstrName')
            productnumber=name[1].split(',')
            d1=0.25*demandamount[int(productnumber[0])]
            cons_list[i].setAttr('RHS',d1)
            p1=name,d1
            demands1.append(p1)
            #print name,d1
         if name[0]==demand[1] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            #print cons_list[i].getAttr('ConstrName')
            productnumber=name[1].split(',')
            d2=random.uniform(0.20,0.30)*demandamount[int(productnumber[0])]
            cons_list[i].setAttr('RHS',d2)
            p2=name,d2
            demands2.append(p2)
         if name[0]==demand[2] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            productnumber=name[1].split(',')
            d3=random.uniform(0.30,0.35)*demandamount[int(productnumber[0])]
            #print cons_list[i].getAttr('ConstrName')
            cons_list[i].setAttr('RHS',d3)
            p3=name,d3
            demands3.append(p3)
         if name[0]==demand[3] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            productnumber=name[1].split(',')
            d4=random.uniform(0.30,0.35)*demandamount[int(productnumber[0])]
            #print cons_list[i].getAttr('ConstrName')
            cons_list[i].setAttr('RHS',d4)
            p4=name,d4
            demands4.append(p4)
    return m.update()

#*******************************************************************
#*******************************************************************
def setVariabkePrice(m,price,newDemandPeriod): #To eleminate variable
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for j in range(len(newDemandPeriod)):
            if name[0]==price[0] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                productnumber=name[1].split(',')
                price1=lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price1)
                #print name,lumberprice[int(productnumber[0])]
                p11=name,price1
                fixeddemandlogprice.append(p11)
            if name[0]==price[1] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price2=random.uniform(1.1,1.5)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price2)
                p22=name,price2
                variabledemandlogprice1.append(p22)
            if name[0]==price[2] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price3=random.uniform(1.0,1.4)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price3)
                p33=name,price3
                variabledemandlogprice2.append(p33)
            if name[0]==price[3] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price4=random.uniform(0.9,1.3)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price4)
                p44=name,price4
                variabledemandlogprice3.append(p44)
    return m.update()
#**********************************************************************
#**********************************************************************

#Main program
from collections import deque
import random
m = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
m1 = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
period=deque(['14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','
29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47
)','48)','49)','50)','51)','52)'])
conperiod=deque(['13)','14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','2
7)','28)','29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)'
,'46)','47)','48)','49)','50)','51)'])
solution_period=['1)','2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','
18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36
)','37)','38)','39)']
totalPeriod=deque(['2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','1
8)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36)'
,'37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47)','48)','49)','50)','51)','52)'])
#Variables*********************************************
var3=['Inv','lostInv','toChips','P','Sale','S1','S2','S3','S4']
var2=['TP','yP','Z']
var_values(m,var3,var2,period)
#*******************************************************
#Constrains*********************************************************
con3=['production','inventory','saleUB','InvLB','setupclass','startInv','endInv','ContractDe
mand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead3','totalSale2']
con2=['inventoryChips','setupUP','setupLB','TotClassSetup','AtLeastOneInClass','mintim
eClass']
con1=['timeProportion','AtLeastOnePerPeriod','maxmimumInv','maxcam','maxcls']
cons_RhsValues(m,con3,con2,con1,period)
#**********************************************************************
demand=['ContractDemand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead
3']
price=['S1','S2','S3','S4']

global
demandamount,lumberprice,demands1,demands2,demands3,demands4,fixeddemandlogp
rice,variabledemandlogprice1,variabledemandlogprice2,variabledemandlogprice3
demandamount=[0,3914,3715,2721,2461,2936,10470,2461,3473,1806,2978,1465,2756,2
675,1925,1563,4793,6226,3925,4196,1840,7532,6967,3985,3761,2645,5514,3070,
3269,2029,2094,2413,2827,2094,1435,415,3576,3075,1759,1261,183,1000]
```

```
lumberprice=[0,4.422433,4.519666,4.616899,4.714132,4.811365,4.58502,4.682198,4.77
9375,4.876553,4.97373,5.361465,5.458531,5.555598,5.652664,5.74973,5.48813,5.58508
5,5.68204,5.778995,5.87595,5.74146,5.838193,5.934925,6.031658,6.12839,5.99479,6.09
13,6.18781,6.284321,6.380831,6.24812,6.344408,6.440696,6.536983,6.633271,6.50145,
6.597516,6.693581,6.789646,6.885711,4.1456]
demands1=[]
demands2=[]
demands3=[]
demands4=[]
fixeddemandlogprice=[]
variabledemandlogprice1=[]
variabledemandlogprice2=[]
variabledemandlogprice3=[]
# Solution
m.setParam("MIPGap",0.04)
m.optimize()
variables=[]
totalsales=[]
totalChips=[]
variables1=[]
vars1=m.getVars()
for i in range(len(vars1)):
   x=vars1[i].getAttr('X')
   nm=vars1[i].getAttr('VarName')
   p=(i,nm,x)
   variables.append(p)
   if x>0 :
      p111=(i,nm,x)
      variables1.append(p111)
   name10=vars1[i].VarName.split('(')
   if name10[0]=='Sale':
      s=(i,nm,x)
      totalsales.append(s)
   if name10[0]=='toChips':
      s=(i,nm,x)
      totalChips.append(s)


import csv
b = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising.csv'
, 'wb')
a = csv.writer(b)
a.writerows(variables)
b.close()
```

```
b1 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand.csv', 'wb')
a1 = csv.writer(b1)
a1.writerows(demands1)
b1.close()

b2 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_Sal
es.csv', 'wb')
a2 = csv.writer(b2)
a2.writerows(totalsales)
b2.close()


b3 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_to
Chips.csv', 'wb')
a3 = csv.writer(b3)
a3.writerows(totalChips)
b3.close()

b4 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_fix
eddemandlogprices.csv', 'wb')
a4 = csv.writer(b4)
a4.writerows(fixeddemandlogprice)
b4.close()

b5 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice1.csv', 'wb')
a5 = csv.writer(b5)
a5.writerows(variabledemandlogprice1)
b5.close()

b6 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice2.csv', 'wb')
a6 = csv.writer(b6)
a6.writerows(variabledemandlogprice2)
b6.close()

b7 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice3.csv', 'wb')
```

```
a7 = csv.writer(b7)
a7.writerows(variabledemandlogprice3)
b7.close()

b8 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising1.cs
v', 'wb')
a8 = csv.writer(b8)
a8.writerows(variables1)
b8.close()

totalS3=[]
totalS4=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='S3':
        s3=(i,nm,x)
        totalS3.append(s3)
    if name10[0]=='S4':
        s4=(i,nm,x)
        totalS4.append(s4)

b9 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S3.csv', 'wb')
a9 = csv.writer(b9)
a9.writerows(totalS3)
b9.close()

b10 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S4.csv', 'wb')
a10 = csv.writer(b10)
a10.writerows(totalS4)
b10.close()

totalTP=[]
totalyP=[]
totalZ=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='TP':
        if x>0 :
            tp=(i,nm,x)
            totalTP.append(tp)
    if name10[0]=='yP':
        if x>0 :
            yp=(i,nm,x)
            totalyP.append(yp)
    if name10[0]=='Z':
        if x>0 :
            z=(i,nm,x)
            totalZ.append(z)

b11 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/TP.csv', 'wb')
a11 = csv.writer(b11)
a11.writerows(totalTP)
b11.close()

b12 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/yP.csv', 'wb')
a12 = csv.writer(b12)
a12.writerows(totalyP)
b12.close()

b13 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Z.csv', 'wb')
a13 = csv.writer(b13)
a13.writerows(totalZ)
b13.close()

b14 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand1.csv', 'wb')
a14 = csv.writer(b14)
a14.writerows(demands1)
b14.close()

b15 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand2.csv', 'wb')
a15 = csv.writer(b15)
a15.writerows(demands2)
b15.close()
```

```
b16 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand3.csv', 'wb')
a16 = csv.writer(b16)
a16.writerows(demands3)
b16.close()

b17 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand4.csv', 'wb')
a17 = csv.writer(b17)
a17.writerows(demands4)
b17.close()
```

SCENARIO 2

```
#*******************************************************************
#*******************************************************************

def var_values(m,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==period[j]:
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
         for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==period[j]:
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
   return m.update()

#*******************************************************************
#*******************************************************************
def cons_RhsValues(m,con3,con2,con1,period):  #To relax constrains
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period[j]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period[j]:
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period[j]:
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',50000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-5000000000000)

   return m.update()


#*******************************************************************
#********************************************************************

def relaxedPreviousPeriodEndInventoryConstrain(m,con3,period):  #To relax end
inventory constrains
   cons_list=m.getConstrs()
```

```
    #print '---------------',con3,period
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        #print '------------------',name[0],conPeriod1[2]
        if name[0]==con3 and conPeriod1[2]==period:
            #print cons_list[i].getAttr('ConstrName'),conPeriod1[2]
            cons_sense=cons_list[i].getAttr('Sense')
            if cons_sense=='<':
                cons_list[i].setAttr('RHS',5000000000000)
            elif cons_sense=='>':
                cons_list[i].setAttr('RHS',-5000000000000)
            else:
                cons_list[i].setAttr('Sense','>')
                cons_list[i].setAttr('RHS',-50000000000)
    return m.update()

#******************************************************************
#*******************************************************************

def solution_update(m,var3,var2,solution_period): # To update solution
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==solution_period: #Take care on
varPeriod[]
                var_value=var_list[i].getAttr('X')
                #print var_list[i].getAttr('varName'),var_value
                var_list[i].setAttr('LB',var_value)
                var_list[i].setAttr('UB',var_value)
        for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==solution_period: #Take care on
varPeriod[]
                var_value=var_list[i].getAttr('X')
                #print var_list[i].getAttr('varName'),var_value
                var_list[i].setAttr('LB',var_value)
                var_list[i].setAttr('UB',var_value)
    return m.update()

#******************************************************************
#******************************************************************

def activareNextPeriodVariable(m,m1,var3,var2,period): #To eleminate variable
    var_list=m.getVars()
    var_list1=m1.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==period: #Take care on varPeriod[]
                p=var_list1[i].getAttr('LB')
                q=var_list1[i].getAttr('UB')
                r=var_list1[i].getAttr('Obj')
                var_list[i].setAttr('LB',p)
                var_list[i].setAttr('UB',q)
                var_list[i].setAttr('Obj',r)
                #print var_list[i].getAttr('varName')
        for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==period: #Take care on varPeriod[]
                p=var_list1[i].getAttr('LB')
                q=var_list1[i].getAttr('UB')
                r=var_list1[i].getAttr('Obj')
                var_list[i].setAttr('LB',p)
                var_list[i].setAttr('UB',q)
                var_list[i].setAttr('Obj',r)
                #print r
    return m.update()

#*********************************************************************
#*********************************************************************

def activareNextPeriodConstrain(m,m1,con3,con2,con1,period):
    cons_list=m.getConstrs()
    cons_list1=m1.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
        for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
        for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
            cons_list[i].setAttr('RHS',cons_RHS)
      for k in range(len(con1)):
         if name[0]==con1[k] and conPeriod[1]==period: #Take care on conPeriod[]
            cons_sense=cons_list1[i].getAttr('Sense')
            cons_RHS=cons_list1[i].getAttr('RHS')
            cons_list[i].setAttr('Sense',cons_sense)
            cons_list[i].setAttr('RHS',cons_RHS)
            #print cons_list[i].getAttr('RHS')
   return m.update()


#*******************************************************************
#*******************************************************************
def setVariabkeDemand(m,demand,newDemandPeriod):  #To add next period constrain
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      for j in range(len(newDemandPeriod)):
         if name[0]==demand[0] and conPeriod1[2]==newDemandPeriod[j]:
            #print cons_list[i].getAttr('ConstrName')
            productnumber=name[1].split(',')
            d1=0.20*demandamount[int(productnumber[0])]
            cons_list[i].setAttr('RHS',d1)
            p1=name,d1
            demands1.append(p1)
            #print name,d1
         if name[0]==demand[1] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            #print cons_list[i].getAttr('ConstrName')
            productnumber=name[1].split(',')
            d2=random.uniform(0.15,0.35)*demandamount[int(productnumber[0])]
            cons_list[i].setAttr('RHS',d2)
            p2=name,d2
            demands2.append(p2)
         if name[0]==demand[2] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            productnumber=name[1].split(',')
            d3=random.uniform(0.15,0.35)*demandamount[int(productnumber[0])]
            #print cons_list[i].getAttr('ConstrName')
            cons_list[i].setAttr('RHS',d3)
            p3=name,d3
            demands3.append(p3)
         if name[0]==demand[3] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
            productnumber=name[1].split(',')
            d4=random.uniform(0.15,0.35)*demandamount[int(productnumber[0])]
            #print cons_list[i].getAttr('ConstrName')
            cons_list[i].setAttr('RHS',d4)
            p4=name,d4
            demands4.append(p4)
   return m.update()

#*****************************************************************
#*****************************************************************
def setVariabkePrice(m,price,newDemandPeriod): #To eleminate variable
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for j in range(len(newDemandPeriod)):
         if name[0]==price[0] and varPeriod[2]==newDemandPeriod[j]:
            productnumber=name[1].split(',')
            price1=lumberprice[int(productnumber[0])]
            var_list[i].setAttr('Obj',price1)
            #print name,lumberprice[int(productnumber[0])]
            p11=name,price1
            fixeddemandlogprice.append(p11)
         if name[0]==price[1] and varPeriod[2]==newDemandPeriod[j]:
            #print var_list[i].getAttr('varName')
            productnumber=name[1].split(',')
            price2=random.uniform(1.2,1.4)*lumberprice[int(productnumber[0])]
            var_list[i].setAttr('Obj',price2)
            p22=name,price2
            variabledemandlogprice1.append(p22)
         if name[0]==price[2] and varPeriod[2]==newDemandPeriod[j]:
            #print var_list[i].getAttr('varName')
            productnumber=name[1].split(',')
            price3=random.uniform(1.1,1.3)*lumberprice[int(productnumber[0])]
            var_list[i].setAttr('Obj',price3)
            p33=name,price3
            variabledemandlogprice2.append(p33)
         if name[0]==price[3] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
            #print var_list[i].getAttr('varName')
            productnumber=name[1].split(',')
            price4=random.uniform(1.0,1.2)*lumberprice[int(productnumber[0])]
            var_list[i].setAttr('Obj',price4)
            p44=name,price4
            variabledemandlogprice3.append(p44)
   return m.update()
#*****************************************************************
#*******************************************************************

#Main program
from collections import deque
import random
m = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
m1 = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
period=deque(['14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','
29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47
)','48)','49)','50)','51)','52)'])
conperiod=deque(['13)','14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','2
7)','28)','29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)'
,'46)','47)','48)','49)','50)','51)'])
solution_period=['1)','2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','
18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36
)','37)','38)','39)']
totalPeriod=deque(['2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','1
8)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36)'
,'37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47)','48)','49)','50)','51)','52)'])
#Variables*********************************************
var3=['Inv','lostInv','toChips','P','Sale','S1','S2','S3','S4']
var2=['TP','yP','Z']
var_values(m,var3,var2,period)
#****************************************************
#Constrains*******************************************************
con3=['production','inventory','saleUB','InvLB','setupclass','startInv','endInv','ContractDe
mand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead3','totalSale2']
con2=['inventoryChips','setupUP','setupLB','TotClassSetup','AtLeastOneInClass','mintim
eClass']
con1=['timeProportion','AtLeastOnePerPeriod','maxmimumInv','maxcam','maxcls']
cons_RhsValues(m,con3,con2,con1,period)
#*******************************************************************
demand=['ContractDemand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead
3']
price=['S1','S2','S3','S4']

global
demandamount,lumberprice,demands1,demands2,demands3,demands4,fixeddemandlogp
rice,variabledemandlogprice1,variabledemandlogprice2,variabledemandlogprice3
demandamount=[0,3914,3715,2721,2461,2936,10470,2461,3473,1806,2978,1465,2756,2
675,1925,1563,4793,6226,3925,4196,1840,7532,6967,3985,3761,2645,5514,3070,
3269,2029,2094,2413,2827,2094,1435,415,3576,3075,1759,1261,183,1000]
lumberprice=[0,4.422433,4.519666,4.616899,4.714132,4.811365,4.58502,4.682198,4.77
9375,4.876553,4.97373,5.361465,5.458531,5.555598,5.652664,5.74973,5.48813,5.58508
```

```
5,5.68204,5.778995,5.87595,5.74146,5.838193,5.934925,6.031658,6.12839,5.99479,6.09
13,6.18781,6.284321,6.380831,6.24812,6.344408,6.440696,6.536983,6.633271,6.50145,
6.597516,6.693581,6.789646,6.885711,3.1456]
demands1=[]
demands2=[]
demands3=[]
demands4=[]
fixeddemandlogprice=[]
variabledemandlogprice1=[]
variabledemandlogprice2=[]
variabledemandlogprice3=[]

for i in range (len(solution_period)):
   m.setParam("MIPGap",0.04)
   #m.setParam("timeLimit",150)
   m.optimize()
   solution_update(m,var3,var2,solution_period[i])
   print '****************Variables value binding period',solution_period[i]
   print '****************Add constrain and variable period',period[0]
   print '****************Relaxed end inventory constrain ',conperiod[0]

   activareNextPeriodVariable(m,m1,var3,var2,period[0])
   activareNextPeriodConstrain(m,m1,con3,con2,con1,period[0])
   con='endInv'
   relaxedPreviousPeriodEndInventoryConstrain(m,con,conperiod[0])
   newDemandPeriod=[]
   for j in range (13):
      newDemandPeriod.append(totalPeriod[j])
   print newDemandPeriod
   setVariabkeDemand(m,demand,newDemandPeriod)
   setVariabkePrice(m,price,newDemandPeriod)
   m.update()
   period.popleft()
   conperiod.popleft()
   totalPeriod.popleft()


# Solution Output
m.optimize()
variables=[]
totalsales=[]
totalChips=[]
variables1=[]
vars1=m.getVars()
for i in range(len(vars1)):
   x=vars1[i].getAttr('X')
```

```
    nm=vars1[i].getAttr('VarName')
    p=(i,nm,x)
    variables.append(p)
    if x>0 :
        p111=(i,nm,x)
        variables1.append(p111)
    name10=vars1[i].VarName.split('(')
    if name10[0]=='Sale':
        s=(i,nm,x)
        totalsales.append(s)
    if name10[0]=='toChips':
        s=(i,nm,x)
        totalChips.append(s)

import csv
b = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising.csv'
, 'wb')
a = csv.writer(b)
a.writerows(variables)
b.close()

b1 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand.csv', 'wb')
a1 = csv.writer(b1)
a1.writerows(demands1)
b1.close()

b2 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_Sal
es.csv', 'wb')
a2 = csv.writer(b2)
a2.writerows(totalsales)
b2.close()


b3 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_to
Chips.csv', 'wb')
a3 = csv.writer(b3)
a3.writerows(totalChips)
b3.close()
```

```
b4 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_fix
eddemandlogprices.csv', 'wb')
a4 = csv.writer(b4)
a4.writerows(fixeddemandlogprice)
b4.close()

b5 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice1.csv', 'wb')
a5 = csv.writer(b5)
a5.writerows(variabledemandlogprice1)
b5.close()

b6 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice2.csv', 'wb')
a6 = csv.writer(b6)
a6.writerows(variabledemandlogprice2)
b6.close()

b7 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice3.csv', 'wb')
a7 = csv.writer(b7)
a7.writerows(variabledemandlogprice3)
b7.close()

b8 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising1.cs
v', 'wb')
a8 = csv.writer(b8)
a8.writerows(variables1)
b8.close()

totalS3=[]
totalS4=[]
vars1=m.getVars()
for i in range(len(vars1)):
   x=vars1[i].getAttr('X')
   nm=vars1[i].getAttr('VarName')
   name10=vars1[i].VarName.split('(')
   if name10[0]=='S3':
      s3=(i,nm,x)
      totalS3.append(s3)
   if name10[0]=='S4':
          s4=(i,nm,x)
          totalS4.append(s4)

b9 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S3.csv', 'wb')
a9 = csv.writer(b9)
a9.writerows(totalS3)
b9.close()

b10 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S4.csv', 'wb')
a10 = csv.writer(b10)
a10.writerows(totalS4)
b10.close()

totalTP=[]
totalyP=[]
totalZ=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='TP':
        if x>0 :
            tp=(i,nm,x)
            totalTP.append(tp)
    if name10[0]=='yP':
        if x>0 :
            yp=(i,nm,x)
            totalyP.append(yp)
    if name10[0]=='Z':
        if x>0 :
            z=(i,nm,x)
            totalZ.append(z)

b11 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/TP.csv', 'wb')
a11 = csv.writer(b11)
a11.writerows(totalTP)
b11.close()

b12 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/yP.csv', 'wb')
a12 = csv.writer(b12)
a12.writerows(totalyP)
```

```
b12.close()

b13 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Z.csv', 'wb')
a13 = csv.writer(b13)
a13.writerows(totalZ)
b13.close()

b14 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand1.csv', 'wb')
a14 = csv.writer(b14)
a14.writerows(demands1)
b14.close()

b15 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand2.csv', 'wb')
a15 = csv.writer(b15)
a15.writerows(demands2)
b15.close()

b16 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand3.csv', 'wb')
a16 = csv.writer(b16)
a16.writerows(demands3)
b16.close()

b17 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand4.csv', 'wb')
a17 = csv.writer(b17)
a17.writerows(demands4)
b17.close()
```

SCENARIO 3

```
#*****************************************************************
#*****************************************************************

def var_values(m,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
```

```
  for i in range(len(var_list)):
    name=var_list[i].VarName.split('(')
    varPeriod=var_list[i].VarName.split(',') # Take care split('(')
    for j in range(len(period)):
      for k in range(len(var3)):
        if name[0]==var3[k] and varPeriod[2]==period[j]: #Take care on varPeriod[]
          #print var_list[i].getAttr('varName')
          var_list[i].setAttr('LB',0)
          var_list[i].setAttr('UB',0)
          var_list[i].setAttr('Obj',0)
      for k in range(len(var2)):
        if name[0]==var2[k] and varPeriod[1]==period[j]: #Take care on varPeriod[]
          #print var_list[i].getAttr('varName')
          var_list[i].setAttr('LB',0)
          var_list[i].setAttr('UB',0)
          var_list[i].setAttr('Obj',0)
  return m.update()

#******************************************************************
#******************************************************************

def cons_RhsValues(m,con3,con2,con1,period):  #To relax constrains
  cons_list=m.getConstrs()
  for i in range(len(cons_list)):
    name=cons_list[i].ConstrName.split('(')
    conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
    conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
    for j in range(len(period)):
      for k in range(len(con3)):
        if name[0]==con3[k] and conPeriod1[2]==period[j]: #Take care on conPeriod[]
          #print cons_list[i].getAttr('ConstrName')
          cons_sense=cons_list[i].getAttr('Sense')
          if cons_sense=='<':
            cons_list[i].setAttr('RHS',5000000000000)
          elif cons_sense=='>':
            cons_list[i].setAttr('RHS',-5000000000000)
          else:
            cons_list[i].setAttr('Sense','>')
            cons_list[i].setAttr('RHS',-50000000000)
      for k in range(len(con2)):
        if name[0]==con2[k] and conPeriod1[1]==period[j]: #Take care on conPeriod[]
          #print cons_list[i].getAttr('ConstrName')
          cons_sense=cons_list[i].getAttr('Sense')
          if cons_sense=='<':
            cons_list[i].setAttr('RHS',5000000000000)
          elif cons_sense=='>':
```

```
                    cons_list[i].setAttr('RHS',-5000000000000)
                else:
                    cons_list[i].setAttr('Sense','>')
                    cons_list[i].setAttr('RHS',-50000000000)
            for k in range(len(con1)):
                if name[0]==con1[k] and conPeriod[1]==period[j]: #Take care on conPeriod[]
                    #print cons_list[i].getAttr('ConstrName')
                    cons_sense=cons_list[i].getAttr('Sense')
                    if cons_sense=='<':
                        cons_list[i].setAttr('RHS',50000000000000)
                    elif cons_sense=='>':
                        cons_list[i].setAttr('RHS',-5000000000000)
                    else:
                        cons_list[i].setAttr('Sense','>')
                        cons_list[i].setAttr('RHS',-5000000000000)

    return m.update()


#*********************************************************************
#**********************************************************************

def relaxedPreviousPeriodEndInventoryConstrain(m,con3,period):  #To relax end
inventory constrains
    cons_list=m.getConstrs()
    #print '---------------',con3,period
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        #print '------------------',name[0],conPeriod1[2]
        if name[0]==con3 and conPeriod1[2]==period: #Take care on conPeriod[]
            #print cons_list[i].getAttr('ConstrName'),conPeriod1[2]
            cons_sense=cons_list[i].getAttr('Sense')
            if cons_sense=='<':
                cons_list[i].setAttr('RHS',5000000000000)
            elif cons_sense=='>':
                cons_list[i].setAttr('RHS',-5000000000000)
            else:
                cons_list[i].setAttr('Sense','>')
                cons_list[i].setAttr('RHS',-50000000000)
    return m.update()



#*********************************************************************
#**********************************************************************
```

```
def solution_update(m,var3,var2,solution_period): # To update solution
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for k in range(len(var3)):
         if name[0]==var3[k] and varPeriod[2]==solution_period: #Take care on
varPeriod[]
            var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==solution_period: #Take care on
varPeriod[]
            var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
   return m.update()

#*****************************************************************
#*****************************************************************

def activareNextPeriodVariable(m,m1,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   var_list1=m1.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for k in range(len(var3)):
         if name[0]==var3[k] and varPeriod[2]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print var_list[i].getAttr('varName')
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print r
    return m.update()

#*****************************************************************
#*****************************************************************

def activareNextPeriodConstrain(m,m1,con3,con2,con1,period):  #To add next period
constrain
    cons_list=m.getConstrs()
    cons_list1=m1.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
        for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
        for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
        for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
    return m.update()


#*****************************************************************
#*****************************************************************
def setVariabkeDemand(m,demand,newDemandPeriod):  #To add next period constrain
    cons_list=m.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        for j in range(len(newDemandPeriod)):
           if name[0]==demand[0] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
              #print cons_list[i].getAttr('ConstrName')
              productnumber=name[1].split(',')
              d1=0.30*demandamount[int(productnumber[0])]
              cons_list[i].setAttr('RHS',d1)
              p1=name,d1
              demands1.append(p1)
              #print name,d1
           if name[0]==demand[1] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
              #print cons_list[i].getAttr('ConstrName')
              productnumber=name[1].split(',')
              d2=random.uniform(0.25,0.35)*demandamount[int(productnumber[0])]
              cons_list[i].setAttr('RHS',d2)
              p2=name,d2
              demands2.append(p2)
           if name[0]==demand[2] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
              productnumber=name[1].split(',')
              d3=random.uniform(0.25,0.35)*demandamount[int(productnumber[0])]
              #print cons_list[i].getAttr('ConstrName')
              cons_list[i].setAttr('RHS',d3)
              p3=name,d3
              demands3.append(p3)
           if name[0]==demand[3] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
              productnumber=name[1].split(',')
              d4=random.uniform(0.25,0.35)*demandamount[int(productnumber[0])]
              #print cons_list[i].getAttr('ConstrName')
              cons_list[i].setAttr('RHS',d4)
              p4=name,d4
              demands4.append(p4)
   return m.update()

#*******************************************************************
#*******************************************************************
def setVariabkePrice(m,price,newDemandPeriod): #To eleminate variable
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for j in range(len(newDemandPeriod)):
           if name[0]==price[0] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
              productnumber=name[1].split(',')
              price1=lumberprice[int(productnumber[0])]
              var_list[i].setAttr('Obj',price1)
              #print name,lumberprice[int(productnumber[0])]
              p11=name,price1
              fixeddemandlogprice.append(p11)
           if name[0]==price[1] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
              #print var_list[i].getAttr('varName')
              productnumber=name[1].split(',')
              price2=random.uniform(1.1,1.5)*lumberprice[int(productnumber[0])]
              var_list[i].setAttr('Obj',price2)
              p22=name,price2
              variabledemandlogprice1.append(p22)
           if name[0]==price[2] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
              #print var_list[i].getAttr('varName')
              productnumber=name[1].split(',')
              price3=random.uniform(1.0,1.4)*lumberprice[int(productnumber[0])]
              var_list[i].setAttr('Obj',price3)
              p33=name,price3
              variabledemandlogprice2.append(p33)
           if name[0]==price[3] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
              #print var_list[i].getAttr('varName')
              productnumber=name[1].split(',')
              price4=random.uniform(0.9,1.3)*lumberprice[int(productnumber[0])]
              var_list[i].setAttr('Obj',price4)
              p44=name,price4
              variabledemandlogprice3.append(p44)
   return m.update()
#*******************************************************************
#*******************************************************************

#Main program
from collections import deque
import random
m = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
m1 = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
period=deque(['14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','
29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47
)','48)','49)','50)','51)','52)'])
```

```
conperiod=deque(['13)','14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','2
7)','28)','29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)'
,'46)','47)','48)','49)','50)','51)'])
solution_period=['1)','2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','
18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36
)','37)','38)','39)']
totalPeriod=deque(['2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','1
8)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36)'
,'37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47)','48)','49)','50)','51)','52)'])
#Variables********************************************
var3=['Inv','lostInv','toChips','P','Sale','S1','S2','S3','S4']
var2=['TP','yP','Z']
var_values(m,var3,var2,period)
#*****************************************************
#Constrains*****************************************************
con3=['production','inventory','saleUB','InvLB','setupclass','startInv','endInv','ContractDe
mand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead3','totalSale2']
con2=['inventoryChips','setupUP','setupLB','TotClassSetup','AtLeastOneInClass','mintim
eClass']
con1=['timeProportion','AtLeastOnePerPeriod','maxmimumInv','maxcam','maxcls']
cons_RhsValues(m,con3,con2,con1,period)
#*****************************************************************
demand=['ContractDemand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead
3']
price=['S1','S2','S3','S4']

global
demandamount,lumberprice,demands1,demands2,demands3,demands4,fixeddemandlogp
rice,variabledemandlogprice1,variabledemandlogprice2,variabledemandlogprice3
demandamount=[0,3914,3715,2721,2461,2936,10470,2461,3473,1806,2978,1465,2756,2
675,1925,1563,4793,6226,3925,4196,1840,7532,6967,3985,3761,2645,5514,3070,
3269,2029,2094,2413,2827,2094,1435,415,3576,3075,1759,1261,183,1000]
lumberprice=[0,4.422433,4.519666,4.616899,4.714132,4.811365,4.58502,4.682198,4.77
9375,4.876553,4.97373,5.361465,5.458531,5.555598,5.652664,5.74973,5.48813,5.58508
5,5.68204,5.778995,5.87595,5.74146,5.838193,5.934925,6.031658,6.12839,5.99479,6.09
13,6.18781,6.284321,6.380831,6.24812,6.344408,6.440696,6.536983,6.633271,6.50145,
6.597516,6.693581,6.789646,6.885711,4.1456]
demands1=[]
demands2=[]
demands3=[]
demands4=[]
fixeddemandlogprice=[]
variabledemandlogprice1=[]
variabledemandlogprice2=[]
variabledemandlogprice3=[]
```

```
for i in range (len(solution_period)):
    m.setParam("MIPGap",0.04)
    #m.setParam("timeLimit",150)
    m.optimize()
    solution_update(m,var3,var2,solution_period[i])
    print '****************Variables value binding period',solution_period[i]
    print '****************Add constrain and variable period',period[0]
    print '****************Relaxed end inventory constrain ',conperiod[0]

    activareNextPeriodVariable(m,m1,var3,var2,period[0])
    activareNextPeriodConstrain(m,m1,con3,con2,con1,period[0])
    con='endInv'
    relaxedPreviousPeriodEndInventoryConstrain(m,con,conperiod[0])
    newDemandPeriod=[]
    for j in range (13):
        newDemandPeriod.append(totalPeriod[j])
    print newDemandPeriod
    setVariabkeDemand(m,demand,newDemandPeriod)
    setVariabkePrice(m,price,newDemandPeriod)
    m.update()
    period.popleft()
    conperiod.popleft()
    totalPeriod.popleft()


# Solution Output
m.optimize()
variables=[]
totalsales=[]
totalChips=[]
variables1=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    p=(i,nm,x)
    variables.append(p)
    if x>0 :
        p111=(i,nm,x)
        variables1.append(p111)
    name10=vars1[i].VarName.split('(')
    if name10[0]=='Sale':
        s=(i,nm,x)
        totalsales.append(s)
    if name10[0]=='toChips':
        s=(i,nm,x)
        totalChips.append(s)



import csv
b = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising.csv'
, 'wb')
a = csv.writer(b)
a.writerows(variables)
b.close()

b1 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand.csv', 'wb')
a1 = csv.writer(b1)
a1.writerows(demands1)
b1.close()

b2 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_Sal
es.csv', 'wb')
a2 = csv.writer(b2)
a2.writerows(totalsales)
b2.close()


b3 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_to
Chips.csv', 'wb')
a3 = csv.writer(b3)
a3.writerows(totalChips)
b3.close()

b4 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_fix
eddemandlogprices.csv', 'wb')
a4 = csv.writer(b4)
a4.writerows(fixeddemandlogprice)
b4.close()

b5 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice1.csv', 'wb')
a5 = csv.writer(b5)
```

```
a5.writerows(variabledemandlogprice1)
b5.close()

b6 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice2.csv', 'wb')
a6 = csv.writer(b6)
a6.writerows(variabledemandlogprice2)
b6.close()

b7 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice3.csv', 'wb')
a7 = csv.writer(b7)
a7.writerows(variabledemandlogprice3)
b7.close()

b8 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising1.cs
v', 'wb')
a8 = csv.writer(b8)
a8.writerows(variables1)
b8.close()

totalS3=[]
totalS4=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='S3':
        s3=(i,nm,x)
        totalS3.append(s3)
    if name10[0]=='S4':
        s4=(i,nm,x)
        totalS4.append(s4)

b9 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S3.csv', 'wb')
a9 = csv.writer(b9)
a9.writerows(totalS3)
b9.close()

b10 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S4.csv', 'wb')
```

```
a10 = csv.writer(b10)
a10.writerows(totalS4)
b10.close()

totalTP=[]
totalyP=[]
totalZ=[]
vars1=m.getVars()
for i in range(len(vars1)):
   x=vars1[i].getAttr('X')
   nm=vars1[i].getAttr('VarName')
   name10=vars1[i].VarName.split('(')
   if name10[0]=='TP':
      if x>0 :
         tp=(i,nm,x)
         totalTP.append(tp)
   if name10[0]=='yP':
      if x>0 :
         yp=(i,nm,x)
         totalyP.append(yp)
   if name10[0]=='Z':
      if x>0 :
         z=(i,nm,x)
         totalZ.append(z)

b11 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/TP.csv', 'wb')
a11 = csv.writer(b11)
a11.writerows(totalTP)
b11.close()

b12 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/yP.csv', 'wb')
a12 = csv.writer(b12)
a12.writerows(totalyP)
b12.close()

b13 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Z.csv', 'wb')
a13 = csv.writer(b13)
a13.writerows(totalZ)
b13.close()
```

```
b14 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand1.csv', 'wb')
a14 = csv.writer(b14)
a14.writerows(demands1)
b14.close()

b15 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand2.csv', 'wb')
a15 = csv.writer(b15)
a15.writerows(demands2)
b15.close()

b16 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand3.csv', 'wb')
a16 = csv.writer(b16)
a16.writerows(demands3)
b16.close()

b17 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand4.csv', 'wb')
a17 = csv.writer(b17)
a17.writerows(demands4)
b17.close()
```

SCENARIO 4

```
#*********************************************************************
#*********************************************************************

def var_values(m,var3,var2,period): #To eleminate variable
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for j in range(len(period)):
            for k in range(len(var3)):
                if name[0]==var3[k] and varPeriod[2]==period[j]: #Take care on varPeriod[]
                    #print var_list[i].getAttr('varName')
                    var_list[i].setAttr('LB',0)
                    var_list[i].setAttr('UB',0)
                    var_list[i].setAttr('Obj',0)
            for k in range(len(var2)):
```

```
            if name[0]==var2[k] and varPeriod[1]==period[j]: #Take care on varPeriod[]
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
   return m.update()

#******************************************************************
#******************************************************************

def cons_RhsValues(m,con3,con2,con1,period):  #To relax constrains
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
      conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
      conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',5000000000000)
               elif cons_sense=='>':
                  cons_list[i].setAttr('RHS',-5000000000000)
               else:
                  cons_list[i].setAttr('Sense','>')
                  cons_list[i].setAttr('RHS',-50000000000)
         for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period[j]: #Take care on conPeriod[]
               #print cons_list[i].getAttr('ConstrName')
               cons_sense=cons_list[i].getAttr('Sense')
               if cons_sense=='<':
                  cons_list[i].setAttr('RHS',50000000000000)
               elif cons_sense=='>':
                cons_list[i].setAttr('RHS',-5000000000000)
            else:
                cons_list[i].setAttr('Sense','>')
                cons_list[i].setAttr('RHS',-5000000000000)

    return m.update()


#******************************************************************
#******************************************************************

def relaxedPreviousPeriodEndInventoryConstrain(m,con3,period):  #To relax end
inventory constrains
    cons_list=m.getConstrs()
    #print '---------------',con3,period
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        #print '-------------------',name[0],conPeriod1[2]
        if name[0]==con3 and conPeriod1[2]==period: #Take care on conPeriod[]
            #print cons_list[i].getAttr('ConstrName'),conPeriod1[2]
            cons_sense=cons_list[i].getAttr('Sense')
            if cons_sense=='<':
                cons_list[i].setAttr('RHS',5000000000000)
            elif cons_sense=='>':
                cons_list[i].setAttr('RHS',-5000000000000)
            else:
                cons_list[i].setAttr('Sense','>')
                cons_list[i].setAttr('RHS',-50000000000)
    return m.update()



#******************************************************************
#******************************************************************


def solution_update(m,var3,var2,solution_period): # To update solution
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==solution_period: #Take care on
varPeriod[]
                var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==solution_period: #Take care on
varPeriod[]
            var_value=var_list[i].getAttr('X')
            #print var_list[i].getAttr('varName'),var_value
            var_list[i].setAttr('LB',var_value)
            var_list[i].setAttr('UB',var_value)
   return m.update()

#*******************************************************************
#*******************************************************************

def activareNextPeriodVariable(m,m1,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   var_list1=m1.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for k in range(len(var3)):
         if name[0]==var3[k] and varPeriod[2]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print var_list[i].getAttr('varName')
      for k in range(len(var2)):
         if name[0]==var2[k] and varPeriod[1]==period: #Take care on varPeriod[]
            p=var_list1[i].getAttr('LB')
            q=var_list1[i].getAttr('UB')
            r=var_list1[i].getAttr('Obj')
            var_list[i].setAttr('LB',p)
            var_list[i].setAttr('UB',q)
            var_list[i].setAttr('Obj',r)
            #print r
   return m.update()

#*******************************************************************
#*******************************************************************

def activareNextPeriodConstrain(m,m1,con3,con2,con1,period):  #To add next period
constrain
```

```
    cons_list=m.getConstrs()
    cons_list1=m1.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
        for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
        for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
        for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
    return m.update()


#*******************************************************************
#*******************************************************************
def setVariabkeDemand(m,demand,newDemandPeriod):  #To add next period constrain
    cons_list=m.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        for j in range(len(newDemandPeriod)):
            if name[0]==demand[0] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                #print cons_list[i].getAttr('ConstrName')
                productnumber=name[1].split(',')
                d1=0.25*demandamount[int(productnumber[0])]
                cons_list[i].setAttr('RHS',d1)
                p1=name,d1
                demands1.append(p1)
                #print name,d1
            if name[0]==demand[1] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                #print cons_list[i].getAttr('ConstrName')
                productnumber=name[1].split(',')
                d2=random.uniform(0.20,0.30)*demandamount[int(productnumber[0])]
                cons_list[i].setAttr('RHS',d2)
                p2=name,d2
                demands2.append(p2)
            if name[0]==demand[2] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                productnumber=name[1].split(',')
                d3=random.uniform(0.30,0.35)*demandamount[int(productnumber[0])]
                #print cons_list[i].getAttr('ConstrName')
                cons_list[i].setAttr('RHS',d3)
                p3=name,d3
                demands3.append(p3)
            if name[0]==demand[3] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                productnumber=name[1].split(',')
                d4=random.uniform(0.30,0.35)*demandamount[int(productnumber[0])]
                #print cons_list[i].getAttr('ConstrName')
                cons_list[i].setAttr('RHS',d4)
                p4=name,d4
                demands4.append(p4)
    return m.update()

#*******************************************************************
#*******************************************************************
def setVariabkePrice(m,price,newDemandPeriod): #To eleminate variable
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for j in range(len(newDemandPeriod)):
            if name[0]==price[0] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                productnumber=name[1].split(',')
                price1=lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price1)
                #print name,lumberprice[int(productnumber[0])]
                p11=name,price1
                fixeddemandlogprice.append(p11)
            if name[0]==price[1] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price2=random.uniform(1.1,1.5)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price2)
                p22=name,price2
                variabledemandlogprice1.append(p22)
            if name[0]==price[2] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price3=random.uniform(1.0,1.4)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price3)
                p33=name,price3
                variabledemandlogprice2.append(p33)
            if name[0]==price[3] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price4=random.uniform(0.9,1.3)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price4)
                p44=name,price4
                variabledemandlogprice3.append(p44)
    return m.update()
#******************************************************************
#******************************************************************

#Main program
from collections import deque
import random
m = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
m1 = read("C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/Orderpromising.lp")
period=deque(['14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','
29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47
)','48)','49)','50)','51)','52)'])
conperiod=deque(['13)','14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','2
7)','28)','29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)'
,'46)','47)','48)','49)','50)','51)'])
solution_period=['1)','2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','
18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36
)','37)','38)','39)']
totalPeriod=deque(['2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','1
8)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36)'
,'37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47)','48)','49)','50)','51)','52)'])
#Variables********************************************
var3=['Inv','lostInv','toChips','P','Sale','S1','S2','S3','S4']
var2=['TP','yP','Z']
```

```
var_values(m,var3,var2,period)
#*****************************************************
#Constrains********************************************************
con3=['production','inventory','saleUB','InvLB','setupclass','startInv','endInv','ContractDe
mand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead3','totalSale2']
con2=['inventoryChips','setupUP','setupLB','TotClassSetup','AtLeastOneInClass','mintim
eClass']
con1=['timeProportion','AtLeastOnePerPeriod','maxmimumInv','maxcam','maxcls']
cons_RhsValues(m,con3,con2,con1,period)
#******************************************************************
demand=['ContractDemand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead
3']
price=['S1','S2','S3','S4']

global
demandamount,lumberprice,demands1,demands2,demands3,demands4,fixeddemandlogp
rice,variabledemandlogprice1,variabledemandlogprice2,variabledemandlogprice3
demandamount=[0,3914,3715,2721,2461,2936,10470,2461,3473,1806,2978,1465,2756,2
675,1925,1563,4793,6226,3925,4196,1840,7532,6967,3985,3761,2645,5514,3070,
3269,2029,2094,2413,2827,2094,1435,415,3576,3075,1759,1261,183,1000]
lumberprice=[0,4.422433,4.519666,4.616899,4.714132,4.811365,4.58502,4.682198,4.77
9375,4.876553,4.97373,5.361465,5.458531,5.555598,5.652664,5.74973,5.48813,5.58508
5,5.68204,5.778995,5.87595,5.74146,5.838193,5.934925,6.031658,6.12839,5.99479,6.09
13,6.18781,6.284321,6.380831,6.24812,6.344408,6.440696,6.536983,6.633271,6.50145,
6.597516,6.693581,6.789646,6.885711,4.1456]
demands1=[]
demands2=[]
demands3=[]
demands4=[]
fixeddemandlogprice=[]
variabledemandlogprice1=[]
variabledemandlogprice2=[]
variabledemandlogprice3=[]

for i in range (len(solution_period)):
   m.setParam("MIPGap",0.04)
   #m.setParam("timeLimit",150)
   m.optimize()
   solution_update(m,var3,var2,solution_period[i])
   print '****************Variables value binding period',solution_period[i]
   print '****************Add constrain and variable period',period[0]
   print '****************Relaxed end inventory constrain ',conperiod[0]

   activareNextPeriodVariable(m,m1,var3,var2,period[0])
   activareNextPeriodConstrain(m,m1,con3,con2,con1,period[0])
   con='endInv'
```

```
    relaxedPreviousPeriodEndInventoryConstrain(m,con,conperiod[0])
    newDemandPeriod=[]
    for j in range (13):
        newDemandPeriod.append(totalPeriod[j])
    print newDemandPeriod
    setVariabkeDemand(m,demand,newDemandPeriod)
    setVariabkePrice(m,price,newDemandPeriod)
    m.update()
    period.popleft()
    conperiod.popleft()
    totalPeriod.popleft()


# Solution Output
m.optimize()
variables=[]
totalsales=[]
totalChips=[]
variables1=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    p=(i,nm,x)
    variables.append(p)
    if x>0 :
        p111=(i,nm,x)
        variables1.append(p111)
    name10=vars1[i].VarName.split('(')
    if name10[0]=='Sale':
        s=(i,nm,x)
        totalsales.append(s)
    if name10[0]=='toChips':
        s=(i,nm,x)
        totalChips.append(s)


import csv
b = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising.csv'
, 'wb')
a = csv.writer(b)
a.writerows(variables)
b.close()
```

```
b1 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand.csv', 'wb')
a1 = csv.writer(b1)
a1.writerows(demands1)
b1.close()

b2 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_Sal
es.csv', 'wb')
a2 = csv.writer(b2)
a2.writerows(totalsales)
b2.close()


b3 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_to
Chips.csv', 'wb')
a3 = csv.writer(b3)
a3.writerows(totalChips)
b3.close()

b4 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_fix
eddemandlogprices.csv', 'wb')
a4 = csv.writer(b4)
a4.writerows(fixeddemandlogprice)
b4.close()

b5 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice1.csv', 'wb')
a5 = csv.writer(b5)
a5.writerows(variabledemandlogprice1)
b5.close()

b6 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice2.csv', 'wb')
a6 = csv.writer(b6)
a6.writerows(variabledemandlogprice2)
b6.close()

b7 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_var
iabledemandlogprice3.csv', 'wb')
```

```
a7 = csv.writer(b7)
a7.writerows(variabledemandlogprice3)
b7.close()

b8 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising1.cs
v', 'wb')
a8 = csv.writer(b8)
a8.writerows(variables1)
b8.close()

totalS3=[]
totalS4=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='S3':
        s3=(i,nm,x)
        totalS3.append(s3)
    if name10[0]=='S4':
        s4=(i,nm,x)
        totalS4.append(s4)

b9 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S3.csv', 'wb')
a9 = csv.writer(b9)
a9.writerows(totalS3)
b9.close()

b10 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/S4.csv', 'wb')
a10 = csv.writer(b10)
a10.writerows(totalS4)
b10.close()

totalTP=[]
totalyP=[]
totalZ=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='TP':
        if x>0 :
            tp=(i,nm,x)
            totalTP.append(tp)
    if name10[0]=='yP':
        if x>0 :
            yp=(i,nm,x)
            totalyP.append(yp)
    if name10[0]=='Z':
        if x>0 :
            z=(i,nm,x)
            totalZ.append(z)

b11 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/TP.csv', 'wb')
a11 = csv.writer(b11)
a11.writerows(totalTP)
b11.close()

b12 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/yP.csv', 'wb')
a12 = csv.writer(b12)
a12.writerows(totalyP)
b12.close()

b13 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Z.csv', 'wb')
a13 = csv.writer(b13)
a13.writerows(totalZ)
b13.close()

b14 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand1.csv', 'wb')
a14 = csv.writer(b14)
a14.writerows(demands1)
b14.close()

b15 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand2.csv', 'wb')
a15 = csv.writer(b15)
a15.writerows(demands2)
b15.close()
```

```
b16 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand3.csv', 'wb')
a16 = csv.writer(b16)
a16.writerows(demands3)
b16.close()

b17 = open('C:/Documents and
Settings/Dalhousie/Desktop/Research/orderpromising52weeks/result/Orderpromising_de
mand4.csv', 'wb')
a17 = csv.writer(b17)
a17.writerows(demands4)
b17.close()
```

SCENARIO 5

```
#******************************************************************
#******************************************************************

def var_values(m,var3,var2,period): #To eleminate variable
   var_list=m.getVars()
   for i in range(len(var_list)):
      name=var_list[i].VarName.split('(')
      varPeriod=var_list[i].VarName.split(',') # Take care split('(')
      for j in range(len(period)):
         for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==period[j]: #Take care on varPeriod[]
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
         for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==period[j]: #Take care on varPeriod[]
               #print var_list[i].getAttr('varName')
               var_list[i].setAttr('LB',0)
               var_list[i].setAttr('UB',0)
               var_list[i].setAttr('Obj',0)
   return m.update()

#******************************************************************
#******************************************************************

def cons_RhsValues(m,con3,con2,con1,period):  #To relax constrains
   cons_list=m.getConstrs()
   for i in range(len(cons_list)):
      name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
        for j in range(len(period)):
            for k in range(len(con3)):
                if name[0]==con3[k] and conPeriod1[2]==period[j]: #Take care on conPeriod[]
                    #print cons_list[i].getAttr('ConstrName')
                    cons_sense=cons_list[i].getAttr('Sense')
                    if cons_sense=='<':
                        cons_list[i].setAttr('RHS',5000000000000)
                    elif cons_sense=='>':
                        cons_list[i].setAttr('RHS',-5000000000000)
                    else:
                        cons_list[i].setAttr('Sense','>')
                        cons_list[i].setAttr('RHS',-50000000000)
            for k in range(len(con2)):
                if name[0]==con2[k] and conPeriod1[1]==period[j]: #Take care on conPeriod[]
                    #print cons_list[i].getAttr('ConstrName')
                    cons_sense=cons_list[i].getAttr('Sense')
                    if cons_sense=='<':
                        cons_list[i].setAttr('RHS',5000000000000)
                    elif cons_sense=='>':
                        cons_list[i].setAttr('RHS',-5000000000000)
                    else:
                        cons_list[i].setAttr('Sense','>')
                        cons_list[i].setAttr('RHS',-50000000000)
            for k in range(len(con1)):
                if name[0]==con1[k] and conPeriod[1]==period[j]: #Take care on conPeriod[]
                    #print cons_list[i].getAttr('ConstrName')
                    cons_sense=cons_list[i].getAttr('Sense')
                    if cons_sense=='<':
                        cons_list[i].setAttr('RHS',50000000000000)
                    elif cons_sense=='>':
                        cons_list[i].setAttr('RHS',-5000000000000)
                    else:
                        cons_list[i].setAttr('Sense','>')
                        cons_list[i].setAttr('RHS',-5000000000000)

    return m.update()


#*******************************************************************
#********************************************************************

def relaxedPreviousPeriodEndInventoryConstrain(m,con3,period):  #To relax end
inventory constrains
    cons_list=m.getConstrs()
    #print '---------------',con3,period
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        #print '-------------------',name[0],conPeriod1[2]
        if name[0]==con3 and conPeriod1[2]==period: #Take care on conPeriod[]
            #print cons_list[i].getAttr('ConstrName'),conPeriod1[2]
            cons_sense=cons_list[i].getAttr('Sense')
            if cons_sense=='<':
                cons_list[i].setAttr('RHS',5000000000000)
            elif cons_sense=='>':
                cons_list[i].setAttr('RHS',-5000000000000)
            else:
                cons_list[i].setAttr('Sense','>')
                cons_list[i].setAttr('RHS',-50000000000)
    return m.update()



#********************************************************************
#**********************************************************************


def solution_update(m,var3,var2,solution_period): # To update solution
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==solution_period: #Take care on
varPeriod[]
                var_value=var_list[i].getAttr('X')
                #print var_list[i].getAttr('varName'),var_value
                var_list[i].setAttr('LB',var_value)
                var_list[i].setAttr('UB',var_value)
        for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==solution_period: #Take care on
varPeriod[]
                var_value=var_list[i].getAttr('X')
                #print var_list[i].getAttr('varName'),var_value
                var_list[i].setAttr('LB',var_value)
                var_list[i].setAttr('UB',var_value)
    return m.update()

#********************************************************************
#********************************************************************
```

```
def activareNextPeriodVariable(m,m1,var3,var2,period): #To eleminate variable
    var_list=m.getVars()
    var_list1=m1.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for k in range(len(var3)):
            if name[0]==var3[k] and varPeriod[2]==period: #Take care on varPeriod[]
                p=var_list1[i].getAttr('LB')
                q=var_list1[i].getAttr('UB')
                r=var_list1[i].getAttr('Obj')
                var_list[i].setAttr('LB',p)
                var_list[i].setAttr('UB',q)
                var_list[i].setAttr('Obj',r)
                #print var_list[i].getAttr('varName')
        for k in range(len(var2)):
            if name[0]==var2[k] and varPeriod[1]==period: #Take care on varPeriod[]
                p=var_list1[i].getAttr('LB')
                q=var_list1[i].getAttr('UB')
                r=var_list1[i].getAttr('Obj')
                var_list[i].setAttr('LB',p)
                var_list[i].setAttr('UB',q)
                var_list[i].setAttr('Obj',r)
                #print r
    return m.update()

#*******************************************************************
#*******************************************************************

def activareNextPeriodConstrain(m,m1,con3,con2,con1,period):  #To add next period
constrain
    cons_list=m.getConstrs()
    cons_list1=m1.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        conPeriod=cons_list[i].ConstrName.split('(') # Take care split('(')
        for k in range(len(con3)):
            if name[0]==con3[k] and conPeriod1[2]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
        for k in range(len(con2)):
            if name[0]==con2[k] and conPeriod1[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
        for k in range(len(con1)):
            if name[0]==con1[k] and conPeriod[1]==period: #Take care on conPeriod[]
                cons_sense=cons_list1[i].getAttr('Sense')
                cons_RHS=cons_list1[i].getAttr('RHS')
                cons_list[i].setAttr('Sense',cons_sense)
                cons_list[i].setAttr('RHS',cons_RHS)
                #print cons_list[i].getAttr('RHS')
    return m.update()


#****************************************************************
#****************************************************************
def setVariabkeDemand(m,demand,newDemandPeriod):  #To add next period constrain
    cons_list=m.getConstrs()
    for i in range(len(cons_list)):
        name=cons_list[i].ConstrName.split('(')
        conPeriod1=cons_list[i].ConstrName.split(',')# Take care split(',')
        for j in range(len(newDemandPeriod)):
            if name[0]==demand[0] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                #print cons_list[i].getAttr('ConstrName')
                productnumber=name[1].split(',')
                d1=0.25*demandamount[int(productnumber[0])]
                cons_list[i].setAttr('RHS',d1)
                p1=name,d1
                demands1.append(p1)
                #print name,d1
            if name[0]==demand[1] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                #print cons_list[i].getAttr('ConstrName')
                productnumber=name[1].split(',')
                d2=random.uniform(0.30,0.35)*demandamount[int(productnumber[0])]
                cons_list[i].setAttr('RHS',d2)
                p2=name,d2
                demands2.append(p2)
            if name[0]==demand[2] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                productnumber=name[1].split(',')
                d3=random.uniform(0.35,0.40)*demandamount[int(productnumber[0])]
                #print cons_list[i].getAttr('ConstrName')
                cons_list[i].setAttr('RHS',d3)
                p3=name,d3
                demands3.append(p3)
            if name[0]==demand[3] and conPeriod1[2]==newDemandPeriod[j]: #Take care
on conPeriod[]
                productnumber=name[1].split(',')
                d4=random.uniform(0.35,0.40)*demandamount[int(productnumber[0])]
                #print cons_list[i].getAttr('ConstrName')
                cons_list[i].setAttr('RHS',d4)
                p4=name,d4
                demands4.append(p4)
    return m.update()

#******************************************************************
#******************************************************************
def setVariabkePrice(m,price,newDemandPeriod): #To eleminate variable
    var_list=m.getVars()
    for i in range(len(var_list)):
        name=var_list[i].VarName.split('(')
        varPeriod=var_list[i].VarName.split(',') # Take care split('(')
        for j in range(len(newDemandPeriod)):
            if name[0]==price[0] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                productnumber=name[1].split(',')
                price1=lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price1)
                #print name,lumberprice[int(productnumber[0])]
                p11=name,price1
                fixeddemandlogprice.append(p11)
            if name[0]==price[1] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price2=random.uniform(1.1,1.5)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price2)
                p22=name,price2
                variabledemandlogprice1.append(p22)
            if name[0]==price[2] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
                #print var_list[i].getAttr('varName')
                productnumber=name[1].split(',')
                price3=random.uniform(1.0,1.4)*lumberprice[int(productnumber[0])]
                var_list[i].setAttr('Obj',price3)
                p33=name,price3
                variabledemandlogprice2.append(p33)
            if name[0]==price[3] and varPeriod[2]==newDemandPeriod[j]: #Take care on
varPeriod[]
```

```
            #print var_list[i].getAttr('varName')
            productnumber=name[1].split(',')
            price4=random.uniform(0.9,1.3)*lumberprice[int(productnumber[0])]
            var_list[i].setAttr('Obj',price4)
            p44=name,price4
            variabledemandlogprice3.append(p44)
   return m.update()
#******************************************************************
#******************************************************************

#Main program
from collections import deque
import random
m = read("C:/Users/Sharif/Desktop/orderpromising52weeks/Orderpromising.lp")
m1 = read("C:/Users/Sharif/Desktop/orderpromising52weeks/Orderpromising.lp")
period=deque(['14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','
29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47
)','48)','49)','50)','51)','52)'])
conperiod=deque(['13)','14)','15)','16)','17)','18)','19)','20)','21)','22)','23)','24)','25)','26)','2
7)','28)','29)','30)','31)','32)','33)','34)','35)','36)','37)','38)','39)','40)','41)','42)','43)','44)','45)'
,'46)','47)','48)','49)','50)','51)'])
solution_period=['1)','2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','
18)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36
)','37)','38)','39)']
totalPeriod=deque(['2)','3)','4)','5)','6)','7)','8)','9)','10)','11)','12)','13)','14)','15)','16)','17)','1
8)','19)','20)','21)','22)','23)','24)','25)','26)','27)','28)','29)','30)','31)','32)','33)','34)','35)','36)'
,'37)','38)','39)','40)','41)','42)','43)','44)','45)','46)','47)','48)','49)','50)','51)','52)'])
#Variables********************************************
var3=['Inv','lostInv','toChips','P','Sale','S1','S2','S3','S4']
var2=['TP','yP','Z']
var_values(m,var3,var2,period)
#****************************************************
#Constrains*******************************************************
con3=['production','inventory','saleUB','InvLB','setupclass','startInv','endInv','ContractDe
mand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead3','totalSale2']
con2=['inventoryChips','setupUP','setupLB','TotClassSetup','AtLeastOneInClass','mintim
eClass']
con1=['timeProportion','AtLeastOnePerPeriod','maxmimumInv','maxcam','maxcls']
cons_RhsValues(m,con3,con2,con1,period)
#******************************************************************
demand=['ContractDemand','SpotDemandLead0','SpotDemandLead2','SpotDemandLead
3']
price=['S1','S2','S3','S4']
```

```
global
demandamount,lumberprice,demands1,demands2,demands3,demands4,fixeddemandlogp
rice,variabledemandlogprice1,variabledemandlogprice2,variabledemandlogprice3
demandamount=[0,3914,3715,2721,2461,2936,10470,2461,3473,1806,2978,1465,2756,2
675,1925,1563,4793,6226,3925,4196,1840,7532,6967,3985,3761,2645,5514,3070,
3269,2029,2094,2413,2827,2094,1435,415,3576,3075,1759,1261,183,1000]
lumberprice=[0,4.422433,4.519666,4.616899,4.714132,4.811365,4.58502,4.682198,4.77
9375,4.876553,4.97373,5.361465,5.458531,5.555598,5.652664,5.74973,5.48813,5.58508
5,5.68204,5.778995,5.87595,5.74146,5.838193,5.934925,6.031658,6.12839,5.99479,6.09
13,6.18781,6.284321,6.380831,6.24812,6.344408,6.440696,6.536983,6.633271,6.50145,
6.597516,6.693581,6.789646,6.885711,4.1456]
demands1=[]
demands2=[]
demands3=[]
demands4=[]
fixeddemandlogprice=[]
variabledemandlogprice1=[]
variabledemandlogprice2=[]
variabledemandlogprice3=[]

for i in range (len(solution_period)):
   m.setParam("MIPGap",0.04)
   #m.setParam("timeLimit",150)
   m.optimize()
   solution_update(m,var3,var2,solution_period[i])
   print '****************Variables value binding period',solution_period[i]
   print '****************Add constrain and variable period',period[0]
   print '****************Relaxed end inventory constrain ',conperiod[0]

   activareNextPeriodVariable(m,m1,var3,var2,period[0])
   activareNextPeriodConstrain(m,m1,con3,con2,con1,period[0])
   con='endInv'
   relaxedPreviousPeriodEndInventoryConstrain(m,con,conperiod[0])
   newDemandPeriod=[]
   for j in range (13):
      newDemandPeriod.append(totalPeriod[j])
   print newDemandPeriod
   setVariabkeDemand(m,demand,newDemandPeriod)
   setVariabkePrice(m,price,newDemandPeriod)
   m.update()
   period.popleft()
   conperiod.popleft()
   totalPeriod.popleft()


# Solution Output
```

```
m.optimize()
variables=[]
totalsales=[]
totalChips=[]
variables1=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    p=(i,nm,x)
    variables.append(p)
    if x>0 :
        p111=(i,nm,x)
        variables1.append(p111)
    name10=vars1[i].VarName.split('(')
    if name10[0]=='Sale':
        s=(i,nm,x)
        totalsales.append(s)
    if name10[0]=='toChips':
        s=(i,nm,x)
        totalChips.append(s)




import csv
b = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising.csv',
'wb')
a = csv.writer(b)
a.writerows(variables)
b.close()

b1 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_demand.c
sv', 'wb')
a1 = csv.writer(b1)
a1.writerows(demands1)
b1.close()

b2 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_Sales.csv'
, 'wb')
a2 = csv.writer(b2)
a2.writerows(totalsales)
b2.close()
```

```
b3 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_toChips.c
sv', 'wb')
a3 = csv.writer(b3)
a3.writerows(totalChips)
b3.close()

b4 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_fixeddem
andlogprices.csv', 'wb')
a4 = csv.writer(b4)
a4.writerows(fixeddemandlogprice)
b4.close()

b5 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_variabled
emandlogprice1.csv', 'wb')
a5 = csv.writer(b5)
a5.writerows(variabledemandlogprice1)
b5.close()

b6 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_variabled
emandlogprice2.csv', 'wb')
a6 = csv.writer(b6)
a6.writerows(variabledemandlogprice2)
b6.close()

b7 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_variabled
emandlogprice3.csv', 'wb')
a7 = csv.writer(b7)
a7.writerows(variabledemandlogprice3)
b7.close()

b8 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising1.csv',
'wb')
a8 = csv.writer(b8)
a8.writerows(variables1)
b8.close()

totalS3=[]
totalS4=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='S3':
        s3=(i,nm,x)
        totalS3.append(s3)
    if name10[0]=='S4':
        s4=(i,nm,x)
        totalS4.append(s4)

b9 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/S3.csv', 'wb')
a9 = csv.writer(b9)
a9.writerows(totalS3)
b9.close()

b10 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/S4.csv', 'wb')
a10 = csv.writer(b10)
a10.writerows(totalS4)
b10.close()

totalTP=[]
totalyP=[]
totalZ=[]
vars1=m.getVars()
for i in range(len(vars1)):
    x=vars1[i].getAttr('X')
    nm=vars1[i].getAttr('VarName')
    name10=vars1[i].VarName.split('(')
    if name10[0]=='TP':
        if x>0 :
            tp=(i,nm,x)
            totalTP.append(tp)
    if name10[0]=='yP':
        if x>0 :
            yp=(i,nm,x)
            totalyP.append(yp)
    if name10[0]=='Z':
        if x>0 :
            z=(i,nm,x)
            totalZ.append(z)

b11 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/TP.csv', 'wb')
a11 = csv.writer(b11)
a11.writerows(totalTP)
b11.close()
```

```
b12 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/yP.csv', 'wb')
a12 = csv.writer(b12)
a12.writerows(totalyP)
b12.close()

b13 = open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Z.csv', 'wb')
a13 = csv.writer(b13)
a13.writerows(totalZ)
b13.close()

b14 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_demand1.
csv', 'wb')
a14 = csv.writer(b14)
a14.writerows(demands1)
b14.close()

b15 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_demand2.
csv', 'wb')
a15 = csv.writer(b15)
a15.writerows(demands2)
b15.close()

b16 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_demand3.
csv', 'wb')
a16 = csv.writer(b16)
a16.writerows(demands3)
b16.close()

b17 =
open('C:/Users/Sharif/Desktop/orderpromising52weeks/result/Orderpromising_demand4.
csv', 'wb')
a17 = csv.writer(b17)
a17.writerows(demands4)
b17.close()
```