Iterative Algorithms for a Joint Pricing and Inventory Control Problem with Nonlinear Demand Functions

by

Anupam Mazumdar

B.Eng. Electrical Engineering (2008)
National University of Singapore

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author ....
School of Engineering
August 5, 2009

Certified by ....
Georgia Perakis
Professor of Operations Research
Thesis Supervisor

Accepted by ....
Jaime Peraire
Professor of Aeronautics and Astronautics
Director, Computation for Design and Optimization

Iterative Algorithms for a Joint Pricing and Inventory Control Problem with Nonlinear Demand Functions

by

Anupam Mazumdar

Submitted to the School of Engineering
on August 5, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

Price management, production planning and inventory control are important determinants of a firm's profitability. The intense competition brought about by rapid innovation, lean manufacturing time and the internet revolution has compelled firms to adopt a dynamic strategy that involves complex interplay between pricing and production decisions. In this thesis we consider some of these problems and develop computationally efficient algorithms that aim to tackle and optimally solve these problems in a finite amount of time.

In the first half of the thesis we consider the joint pricing and inventory control problem in a deterministic and multiperiod setting utilizing the popular log linear demand model. We develop four algorithms that aim to solve the resulting profit maximization problem in a finite amount of time. The developed algorithms are then tested in a variety of settings ranging from small to large instances of trial data.

The second half of the thesis deals with setting prices effectively when the customer demand is assumed to follow the multinomial logit demand model, which is the most popular discrete choice demand model. The profit maximization problem (even in the absence of constraints) is non-convex and hard to solve. Despite this fact we develop algorithms that compute the optimal solution efficiently. We test the algorithms we develop in a wide variety of scenarios from small to large customer segment, with and without production/inventory constraints. The last part of the thesis develops solution methods for the joint pricing and inventory control problem when costs are linear and demand follows the multinomial logit model.

Thesis Supervisor: Georgia Perakis
Title: Professor of Operations Research

Acknowledgments

I will like to thank:

Prof. Georgia Perakis, for introducing me to the fascinating world of constructing and solving optimization models, her constant guidance, prompt help and most importantly her enthusiasm and keen interest in my work. Without her mentoring, valuable advice and constant feedback this dissertation could never have been completed.

All my teachers and course instructors for sharing their knowledge and interests with me.

My parents for their patience and constant encouragement.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 Contributions and Thesis Outline 14

2 Joint Pricing and Inventory Control Problem with Log Linear Demand **17**

2.1 Problem Overview 18

2.1.1 Algorithms for solving the Joint Pricing and Inventory Control Problem 18

2.1.2 Log Linear Demand Model 20

2.2 JPICP Formulation and Model Description 22

2.3 JPICP Algorithms under Log Linear Demand 24

2.4 Runtime Performance Comparison of the JPICP Algorithms 32

2.5 Conclusions 39

3 Pricing in Response to the Multinomial Logit Demand Function **41**

3.1 Problem Overview 42

3.1.1 Discrete Choice Models 42

3.1.2 Algorithms for Pricing in Response to DCMs 44

3.2 Description of the MNL based Model 45

3.3 Algorithm based on the KKT Optimality Conditions 47

3.4 Algorithm based on Log Reformulation of the MNL Demand Model . 60

3.5 Runtime Performance Comparison of Different MNL Pricing Algorithms 61

3.6 Conclusions 67

4 Joint Pricing and Inventory Control Problem under the Multinomial Demand **69**

4.1 JPICP in a MNL Setting 69

4.2 An Iterative Algorithm based on a Log Reformulation of the MNL model 71

4.3 Computational Performance of the Algorithm 74

4.4 Conclusions 75

A Definitions **77**

List of Figures

2-1 Log linear demand function for a single product 21
2-2 Run time comparison of different algorithms. (Product, Periods) denotes that a value of 10 on the x - axis indicates that the instance had ten products and ten time - periods 35
2-3 Run time comparison of different algorithms. In these instances the total number of products was fixed at 100 and only time periods were varied
36
2-4 Run time comparison of ST1 and ST4 for a randomly generated starting point 37
2-5 Run time comparison of ST1 and ST4 for a starting point far from the optimal solution 37
2-6 Run time comparison of ST1 and ST4 for a starting point close to the optimal solution 37

List of Tables

2.1 Comparison of optimal profit generated by different algorithms 35

2.2 Comparison of optimal L1 norms of optimal demand generated by different algorithms 35

2.3 Comparison of optimal profit generated by different algorithms 36

2.4 Comparison of optimal L1 norms of optimal demand generated by algorithms 36

2.5 Comparison of optimal profit generated by ST1 and ST4 38

2.6 Comparison of starting L1 norms of the initial feasible demand 38

3.1 Run-time comparison of algorithms T1 and T4 for unconstrained version of problem (R) with MS = 10^5. Time is measured in cpu seconds. 62

3.2 Run-time comparison of algorithms T1 and T4 for unconstrained version of problem (R) with MS = 1. Time is measured in cpu seconds. . 62

3.3 Run-time comparison of algorithms T1-T4 with MS = 1. Time is measured in cpu seconds. '–' indicates that the simulation took too long to run. 64

3.4 Optimal Revenue computed by algorithms T1-T4 with MS = 1. '–' indicates that the simulation took too long to run. 64

3.5 Run-time comparison of algorithms T1-T4 with MS = 10^2. Time is measured in cpu seconds. '–' indicates that the simulation took too long to run. '$*$' indicates that the simulation did not generate the optimal solution. 64

3.6 Optimal Revenue computed by algorithms T1-T4 with MS = 10^2. '$-$' indicates that the simulation took too long to run. '$*$' indicates that the simulation did not generate optimal results. 65
3.7 Run-time comparison of algorithms T1-T4 with MS = 10^5. Time is measured in cpu seconds. '$-$' indicates that the simulation took too long to run. '$*$' indicates that the simulation encountered problems. . 65
3.8 Optimal Revenue computed by algorithms T1-T4 with MS = 10^5. '$-$' indicates that the simulation took too long to run. '$*$' indicates that the simulation encountered problems. 66
3.9 Run-time comparison of algorithms T2, T3 and T4 with MS = 1, after removing products with small value of optimal θ. Time is measured in cpu seconds. 66
3.10 Run-time comparison of algorithms T2,T4 and T5 with MS = 10^5. Time is measured in cpu seconds. 67

4.1 Comparison of optimal profit and total time required by M1 and ST5. '-' indicates that the data instance took very long time to run 75

Chapter 1

Introduction

1.1 Motivation

Setting and managing prices effectively is one of the key determinants of a firm's profitability. Today when consumers can compare prices for an entire assortment of substitutable goods at the click of a mouse, setting and managing prices effectively has become even more important. Notable examples of industry sectors where this setting applies include the airline, hotel, car - rental and fashion industries. Pricing managers need to take into consideration a number of factors like how the price for a particular product/offering will compare with other similar products of the competitors, other products in the comany's own product line and where the possibility exists with used second hand variants of the same product [1]. The problem of determining prices effectively is further compounded by seasonal variations in demand, temporary promotions and short product life cycles (Eg. Cellphone handsets). All these factors are driving firms towards adopting a dynamic pricing strategy which can rapidly respond to changes in the competitive enviroment. Two key components of a dynamic pricing strategy are good demand models and the ability to solve these large scale models involving the entire assortment of products. In this thesis we are concerned with the latter i.e. given a revenue model (based on a specific demand function), how to solve it accurately and efficiently in minimal amount of time. In order to test our proposed algorithms we utilize the loglinear and multinomial logit demand models.

These two demand models have been utilized in a wide variety of industry settings (for details see Chapters 2 and 3).

Another important dimension of the above problem is production planning / inventory control. Production decisions must take into account the seasonal fluctuation in demand, the short duration demand surge on account of promotions and most importantly the effect of relative differences in prices between products on the demand for a specific product. Historically there has been a disconnect between price theory and inventory control although both problems are complementary. Frequently the inventory control systems assume a given fixed price structure and focus only on cost minimization [2]. However given the variability in demand, effect of relative differences in prices on product sales and the fact that often times the products being manufactured or stored share the same production or warehouse space (Eg. a car manufacturer manufacturing different car variants), there is a strong motivation to study both the aspects of the problem under a unified frameowork utilizing models that simultaneously address both aspects of the problem. In this thesis we address this problem for the cases when demand is assumed to follow the log linear demand model (Chapter 2) and the multinomial logit (MNL) demand model (Chapter 4). There is an extensive literature on pricing based on MNL demand model and solving the joint pricing and inventory control problem (JPICP) based on different demand models. We will briefly review the relevant literature at the beginning of each chapter.

1.2 Contributions and Thesis Outline

The thesis is divided into four chapters including the current one. Chapters 2 and 3 can be read independently. However as Chapter 4 builds on concepts developed in all the preceeding chapters, hence the reader is strongly advised to read the preceeding chapters before taking up Chapter 4. A definition list of mathematical terms / conditions is included in the appendix for the convenience of the reader. The terminology defined in the appendix have been underlined throughout the thesis. A brief discussion on the content of different chapters is provided next.

Chapter 2 considers algorithms to find the optimal solution to the joint pricing and inventory control problem in the context of a nonlinear and deterministic demand function. The problem and the demand function utilized (log-linear) are covered in the first two sections. This is followed by a discussion on the developed solution techniques. Computational results are included next, followed by conclusion that brings together the important points of the chapter and includes suggestions for future research work. The main contribution of this chapter is the development of accurate and computationally tractable algorithms that perform better than commercially available software in a wide variety of settings. In particular an algorithm based on a combination of quasi newton and projection methods was found to be computationally the most efficient for solving the joint pricing and inventory control problem when demand follows the log linear model and costs are linear.

Chapter 3 considers the problem of optimally pricing different products in a product line when customer demand is assumed to follow the nonconvex multinomial logit (MNL) demand model. In the first section we review the literature that discusses the wide applicability of discrete choice demand models. The subsequent sections describe algorithms for finding the optimal prices of different products in response to a MNL demand model. Computational results on the algorithm performance and accuracy are included in the penultimate section which is followed by a brief conclusion section. Contributions of Chapter 3 include the developement of a set of algorithms based on the Karush Kuhn Tucker conditions for solving the nonconvex profit maximization problem, run time comparison of the developed algorithms with other known algorithms and a brief discussion on the advantages and disadvantages of the developed algorithms. The algorithms based on the KKT optimality conditions were found to be computationally the most efficient for the cases with a large number of products and customers in the relevant market segment.

Finally Chapter 4 utilizes concepts from Chapters 2 and 3 in order to propose an algorithm for solving the joint pricing and inventory control problem when the demand model is assumed to be MNL. The first section introduces the two models. This is followed by algorithm development in the next section. Computational results are then presented followed by suggestions on how the existing model can be expanded and improved further. The main contribution of this chapter is the development of a computationally efficient and tractable algorithm for solving the JPICP based on the MNL demand model.

Chapter 2

Joint Pricing and Inventory Control Problem with Log Linear Demand

In this chapter we describe algorithms for solving the joint pricing and inventory control problem (JPICP) under the log - linear demand model. Section 2.1 provides a broad overview of the log - linear demand model, its important properties and of algorithms for solving the JPICP problem. We formulate the problem in Section 2.2. Section 2.3 provides a detailed description of three variants of the developed algorithm and discusses underlying assumptions necessary for the algorithms to converge to an optimal solution. A runtime performance comparison of these algorithms is then provided in Section 2.4, followed by a discussion on the advantages and disadvantages of the developed algorithms. As the developed algorithms build on and extend an earlier work by Rao [15], some of the proofs and general discussion on convergence criteria of iterative Newton based schemes have been omitted.

2.1 Problem Overview

2.1.1 Algorithms for solving the Joint Pricing and Inventory Control Problem

In this subsection we will review some of the work that has been done in addressing the JPICP problem. One of the first JPICP models was formulated by T.M. Whitin [2]. The proposed model was based on a linear demand and cost model for a single product in a single time - period. Known lot size models were modified to include 'price' as a variable. The new variable was introduced utilizing the demand function for the product under consideration. Whitin's work was improved upon by Pekelman [3] who considers a continuous time horizon and a time dependent demand curve for a single product with convex production costs. The proposed algorithm for solving this joint pricing and production problem utilizes control theory and is based on a state constraint in the form of a non negative constraint on the inventory level. Pekelman's work has been extended by a number of other researchers. In an extension that considers nonlinear but strictly convex production costs in a similar setting, Teng et al. [4] propose a forward branch and bound algorithm based on imposing upper and lower bounds on both state and control variables and utilizing Lagrangian form of maximum principle. Feichtinger and Harl [5] extend Pekelman's model to nonlinear demand models and allow for product shortages or backorders. Utilizing the generalized maximum principle and phase potrait analysis, they show that if one starts with initial excesss inventory both the price and production rate will initially remain at zero. This phase will be followed by a phase where price will start to increase but production rate will still remain at zero. Finally once the inventory falls below a set threshold, production will start while price will continue to rise further. In an infinte time horizon the authors show that both production rate and price converge to their long run optima. Gaimon [6] considers an extension of Pekelman's model where the time dependent demand function is still linear but production capacity can be acquired or increased over time leading to lower marginal costs and reduced inventory carrying cost. Jorgenssen *et al.* [7] utilize Pekelman's model in the setting of a monopolistic firm that plans out its production, pricing and inventory policy for a finite and fixed time horizon. They enhance the model with marketing insights, for example, that past period prices affect current period prices, as well as principles of industrial organization that dictate that with time firm's learn by doing, resulting in lower per unit production cost. The authors solve their proposed single product model using a path synthesizing procedure.

All the models considered so far while considering important and interesting aspects of the JPICP problem are all restricted to the case of a single product in a discrete or continuous time setting. We next consider some models that have been developed in a multiproduct multiperiod setting. One of the first works to explicitly consider both multiple products and shared production capacities in a discrete multiple time setting was done by Gilbert [8]. Assuming seasonable variation in demand and negligible set up costs, he exploited the resulting structure of the problem to split it into two parts - (1) an LP (linear program) to generate the optimal basis to minimize production costs given a a fixed demand vector (2) an NLP (nonlinear program) utilizing the dual variables of the LP to solve the pricing problem associated with maxmimizing the firm's net profit. This model utilizes a solution technique very similar to the one utilized in this thesis. Nevetheless, we propose and develop efficient ways to solve the NLP part of the problem. Furthermore seasonality of demand is not explicitly addressed in this thesis. In yet another relevant work Adida and Perakis [9] extend Pekelman's original single product model to a multiproduct model with capacity constraints. The continuous time model is based on a linear demand function, linear inventory costs and increasing strictly convex production costs. The solution technique for the model utilizes adjoint variables and dual multipliers iteratively to compute the optimal policy for the different decision variables. The model presented in this thesis is similar to the one utilized in this work, the key diffrences being (i) we choose demand functions that are nonlinear and sometimes also nonconvex, and (ii) we discretize time.

2.1.2 Log Linear Demand Model

A product's demand function depends on a whole multitude of factors such as price, advertising, brand name etc. Modelling all the relevant variables makes the demand function more acurate [19]. However, often accurate information about all the model parameters is not readily available. Most demand functions are designed to take into account only price related effects. Elasticity of demand relates change in quantity demanded with changes in price i.e. $\epsilon(p) = (\frac{p}{d}).(\frac{\partial d}{\partial p})$. The log - linear demand function is characterized by constant elasticity i.e $\epsilon(p)$ is a constant. See also for example [19] for a general discussion on the log linear demand model. For a single product with only one alternative (or substitute) this demand function is given by:

$$ln(D) = C_0 + C_1 p_o + C_2 p_c$$

Here C_0 denotes the demand for the product when price is zero, C_1 is the own price elasticity of the product, p_o is the price of the product, p_c is the price of the alternative product and C_2 is the cross price elasticity of the product with the alternative. The coefficient C_1 has to be negative for the law of demand to hold (i.e. as the price of product increases, the demand for the product goes down). The coefficient C_2 is positive or negative depending on whether the other product is a substitute or a complement. For a single product with no alternative ($C_2 = 0$) Figure 2-1 illustrates the log-linear demand function. In a multiproduct setting the log linear demand model as a function of prices is given by:

$$ln(d(\mathbf{P})_i) = a_i - B_i^T P, \quad \forall i = 1, 2, ..., N \tag{2.1}$$

In Equation (2.1) N is the number of all products under consideration, P and B_i denote the N x 1 price and price sensitivity (of product 'i' $\in N$ with respect to all other products) vectors respectively. a_i is a scalar constant which represents the demand for the product when the price equals zero. Utilizing the log - linear demand model the expression for the revenue, r($\mathbf{D}$), in a multiproduct setting is given by Equation

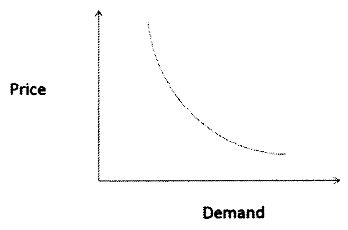

Figure 2-1: Log linear demand function for a single product

(2.3). In the equation for r(**D**), **D** is the vector of demand for different products (in a multi period setting D and P will be N x T matrices , where N is the total number of products and T is the total number of time - periods under consideration), **a** is the vector representing the demand for different products when their prices are set to zero and B^{-1} is the inverse of the N x N price sensitivity matrix. It is worth noticing that the revenue equation has been written in the demand space. This can be done as long the matrix B is strictly diagonally dominant which ensures that B^{-1} exists, which in turn ensures the existence of the inverse of the demand function given by Equation (2.2).

$$P(D) = B^{-1}\{a - ln(D)\} \tag{2.2}$$

$$R(D) = [B^{-1}\{a - ln(D)\}]^T.D \tag{2.3}$$

The log - linear demand function introduced above has several useful properties. Most notable among these is the fact that demand is always nonnegative which implies that the quantity arising from the demand function is always nonnegative and no additional constraint is needed to ensure that fact. Also by taking the log of the demand, one can recover the linear form which makes it easy to estimate the different coefficients utilizing linear regression.

2.2 JPICP Formulation and Model Description

In this section a formal model of the JPICP is introduced in terms of a generic demand function d($\mathbf{P}$). The important model parmeters and variables are as follows:
$d(p)_{it}$, p_{it}, u_{it} and I_{it} denote the demand, price, units produced and units stored respectively for product i in time period t
c_{it} and h_{it} denote the cost of producing and inventory holding cost of product i in time period t
C_t denotes the total production capacity in time period t
N denotes the number of all available products
T denotes the number of all the discrete time - periods under consideration

Utilizing the notation defined above, a generic model of the JPICP can be written as (Profit = Revenue - Cost):

$$\text{P1: } \max \sum_{i=1}^{N}\sum_{t=1}^{T}(p_{it}d(p)_{it} - c_{it}u_{it} - h_{it}I_{it}) \tag{2.4}$$

subject to:

$$d(p)_{it} = I_{i,t-1} - I_{it} + u_{it}, \quad \forall i = 1,2,....,N \quad \forall t = 1,2,....,T \tag{2.5}$$

$$\sum_{i=1}^{N} u_{it} \leq C_t, \quad \forall t = 1,2,...,T \tag{2.6}$$

$$I_i(0) = I_{i0}, \quad \forall i = 1,2,....,N \tag{2.7}$$

$$d(p)_{it} \geq 0, \quad \forall i = 1,2,...N \quad \forall t = 1,2,....,T \tag{2.8}$$

$$p_{it} \geq 0, \quad \forall i = 1,2,...N \quad \forall t = 1,2,....,T \tag{2.9}$$

$$u_{it} \geq 0, \quad \forall i = 1, 2, ...N \quad \forall t = 1, 2, ...T \tag{2.10}$$

$$I_{it} \geq 0, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, ..., T \tag{2.11}$$

Constraint (2.5) ensures that the entire demand for the product is met either by producing or utilizing the stored inventory of the product in the specific period. Constraint (2.6) ensures that the sum of all the goods produced in a specific period does not exceed the total joint production capacity corresponding to that period. Finally Constraint (2.7) initializes the initial supply of inventory so that Constraint (2.5) is well defined for time period *t=1*. The remaining constraints simply ensure the non-negativity of the model variables.
Before proceeding to the next section where algorithms for solving the JPICP efficiently in a log linear setting are discussed, it will be worthwhile to take note of the following important observations. These observations are discussed in depth in the next section.

1. If the demand function is **invertible** i.e. d(**P**) is a one to one function over its entire domain, then the problem (P1) can be transformed completely into the demand space.

2. Also in case of an invertible demand function, if 'somehow' the demand levels are set, then its possible to separate out the 'cost part' of the objective and minimize it by solving a separate linear program.

3. Furthermore the linear program for minimizing the cost part of the objective can be reformulated into a network optimization problem which in practice can be solved very efficiently.

2.3 JPICP Algorithms under Log Linear Demand

We begin this section with a brief description on how to reformulate the cost minimization part of the original formulation (P1) into a linear transportation problem (see [15] for more details). In a linear transportation problem, there are a set of demand (d_j) and supply nodes (s_i). The supply nodes have a fixed capacity and each of the demand nodes have a fixed demand that needs to be satisfied. The supply nodes are connected to the demand nodes by a set of arcs with cost c_{ij} i.e. c_{ij} represents the cost per unit of the goods transported from supply node i to demand node j. Assumming that the problem is feasible or in other words $\sum_i s_i = \sum_j d_j$, in general, the linear transportation problem can be written as:

$$min \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij}x_{ij} \tag{2.12}$$

subject to:

$$\sum_{i=1}^{M} x_{ij} = d_j, \quad \forall j = 1,2,...,N \tag{2.13}$$

$$\sum_{j=1}^{N} x_{ij} = s_i, \quad \forall i = 1,2,...,M \tag{2.14}$$

$$x_{i,j} \geq 0, \quad \forall i = 1,2,...,M \quad \forall j = 1,2,...,N \tag{2.15}$$

The cost part of the objective in the optimization problem (P1) can be reformulated into a linear transportation problem as follows (see also [15] for a more detailed discussion):

Let us denote the total supply capacity (this is the sum of the goods that can be produced as well as the available inventory) for each period t (= 1,2...,T) with C_s and assosiate this capacity with the supply node s (= 1,2,...,T) corresponding to that time period i.e. we will have one supply node for each of time period under consideration. Next we create N X T demand nodes, where d_{it} is the demand for product i in period t, N is the total number of products and T is the total number of time - periods under consideration. Finally we create a set of arcs (s, i, t) denoting flow of goods x_{it}^s from supply node s to demand node d_{it} with cost b_{it}^s. The point to note here is that in practice $s \leq t$, because $s > t$ will imply supplying goods produced in the future to meet demand in the past. The cost asociated with each of the supply arcs is given by $b_{it}^s = c_i^s + \sum_{s \leq u < t} h_i^u$. In the equation for the cost associated with the arcs, c_i^s denotes the cost of producing the product i in time - period s and $\sum_{s \leq u < t} h_i^u$ represents the sum of storing the particular product from the period s when it was produced to the t when it was finally sold/consumed. Further to account for the initial inventory (i.e. inventory at time $t = 0$), we may either set it to zero or add a set of additional N supply nodes with arcs to all the demand nodes d_{it} denoting the initial inventory level. The cost of the connecting arcs should be set to $b_{it}^s = \sum_{s \leq u < t} h_i^u$.

Based on the above definitions and reformulations the cost part in the optimization problem $P1$ can be rewritten as:

$$D1 : c(\mathbf{D}) = min_x \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{s=1}^{T} b_{it}^s x_{it}^s \tag{2.16}$$

subject to:

$$\sum_{i,t:s \leq t \leq T} x_{it}^s \leq C_s, \quad \forall s = 1, 2, ..., T \tag{2.17}$$

$$\sum_{s=1}^{t} x_{it}^s \geq d_{it}, \quad \forall i = 1, .., N \quad \forall t = 1, 2, ..., T \tag{2.18}$$

$$x_{it}^s \geq 0, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, .., T \quad \forall s = 1, 2, ..., T \tag{2.19}$$

We next outline the set of assumptions under which the algorithms introduced later in this section are guaranteed to converge to the global optimal solution (for proofs refer to [15] and Chapter 2 in [17]).

Assumption 2.1 The demand function D($\mathbf{P}$), where
$D(\mathbf{P}) = (d_{11}(\mathbf{P}), d_{12}(\mathbf{P}), ..., d_{NT}(\mathbf{P})$, is **invertible** or in other words there is a one to one mapping from the price space to the demand space ($\Re_+^{NT} \mapsto \Re_+^{NT}$).

Assumption 2.2 The matrix B is strictly diagonally dominant and has positive diagonal elements and non positive off diagonal elements.
This assumption ensures that the price sensitivity matrix B is invertible and positive definite. Furthermore it implies that the products are gross substitutes.

Assumption 2.3 The revenue function denoted by $r(\mathbf{D}) = P(\mathbf{D})^T\mathbf{D}$ is a concave function.

Assumption 2.4 $r(\mathbf{D})$ has continuous first and second derivatives.

Assumption 2.5 The function -r($\mathbf{D}$) is Lipschitz continuous with constant $L > 0$.

The above assumptions are required to ensure that the profit maximization problem has a concave objective over a convex feasible region. This in turn ensures that the iterative algorithms based on considering a local quadratic approximation of the objective at each iteration converge to the global optimal. We next outline a new JPICP formulation ($P2$) which is based on the demand space and makes use of the new production decision variables x_{it}^s introduced in the beginning of this section.

$$P2 : max_{\mathbf{D},x} \sum_{i=1}^{N}\sum_{t=1}^{T} d_{it}p(\mathbf{D})_{it} - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{s=1}^{T} b_{it}^s x_{it}^s \tag{2.20}$$

subject to:

$$\sum_{s=1}^{t} x_{it}^s \geq d_{it}, \quad \forall i = 1,2,..N \quad \forall t = 1,2,..,T \tag{2.21}$$

$$\sum_{i,g:s\leq g\leq t} x_{ig}^s \leq C_s, \quad \forall s = 1,2,..,T \quad \forall t = s,..,T \tag{2.22}$$

$$d_{it} \geq 0, \quad \forall i = 1,2..,N \quad \forall t = 1,2,...,T \tag{2.23}$$

$$p_{it}(\mathbf{D}) \geq 0, \quad \forall i = 1,2,...,N \quad \forall t = 1,2,...,T \tag{2.24}$$

We utilize the log linear demand model, which is one of the deterministic nonlinear demand models. This model will enable the optimization problem $(P2)$ to satisfy the required assumptions on the revenue function of the objective. The problem $(P2)$ can be rewritten as:

$$P3 : max_{\mathbf{D},x} \quad \sum_{t=1}^{T}\sum_{i=1}^{N} d_{it}(\sum_{j\in N} B_{ij}^{-1}(a_j - ln(d_{jt}))) - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{s=1}^{T} b_{it}^{s} x_{it}^{s} \tag{2.25}$$

subject to:

$$\sum_{s=1}^{t} x_{it}^{s} \geq d_{it}, \quad \forall i = 1, ..., N \quad \forall t = 1, 2..., T \tag{2.26}$$

$$\sum_{i,g:s\leq g\leq t} x_{ig}^{s} \leq C_s, \quad \forall s = 1, 2, .., T \quad \forall t = s, .., T \tag{2.27}$$

$$e^{a_i} \geq d_{it}, \quad \forall i = 1, 2, .., N \quad \forall t = 1, 2, .., T \tag{2.28}$$

In the above formulation the elements a_i of vector '**a**', representing the demand for different products when the price equals zero is assumed to be the same for all the different periods $t = 1, 2, .., T$. Notice that relation (2.28) implies that prices stay positive at all times. This follows directly from Equation (2.2) in conjunction with assumption 2.2 (this implies matrix B is a M matrix and hence that $B^{-1} \geq 0$ componentwise). In a log - linear demand model as demand is always positive, we can drop constraints corresponding to Equation (2.23). The formulation $(P3)$ can be solved directly utilizing a non - linear solver such as LOQO. We refer to the algorithm for solving formulation (P3) through LOQO as **ST1**.

Another iterative algorithm for solving formulation (P3) can be developed by replacing at each iteration, the part of the objective corresponding to r(D) with its local quadratic approximation around the current iterate $\overline{D}$. The algorithm (**ST2**) is oulined below:

Step 1:

Start with a feasible point $\overline{D}$ in the feasible region of the problem $(P3)$. Compute $\nabla r(\overline{D})$, diag $H(\overline{D})$ at this point

Step 2:

Construct a local quadratic approximation of r((D)) at $\overline{D}$. Use this quadratic approximation instead of the exact function r((D)) in the objective of $(P3)$.

Step 3:

Solve the optimization problem $(P3)$ using a QP solver such as CPLEX.

Step 4:

Check the difference between the optimal demand D and $\overline{D}$. If this difference is less than a fixed tolerence ($= 10^{-8}$), stop, otherwise replace $\overline{D}$ with the optimal demand found in Step 3, D, and recompute $\nabla r(\overline{D})$, diag$H(\overline{D})$ and goto step 2.

A quadratic local approximation of r($\mathbf{D}$) around a feasible point $\overline{D}$ is given by Equation (2.29).

$$r(\overline{D}) = r(\overline{D}) + \nabla r(\overline{D})^T(D - \overline{D}) + \frac{1}{2}(D - \overline{D})^T.diagH(\overline{D}).(D - \overline{D}) \tag{2.29}$$

In the above equation diag$H(\overline{D})$ denotes the diagonal part of the <u>Hessian matrix</u> of r($\mathbf{D}$). Utilizing a log linear demand model r($\mathbf{D}$) for a single period is given by equation 2.30.

$$r(\mathbf{D}) = B^{-1}(\mathbf{a} - ln(\mathbf{D})).\mathbf{D} = \sum_{i=1}^{N}(\sum_{j=1}^{N} B_{ij}^{-1}(a_j - ln(d_j)))d_i \tag{2.30}$$

The $m^{th}(= 1, 2.., N)$ term of $\nabla r(\overline{D})$ is given by equation 2.31.

$$[\nabla r(\overline{D})]_m = \sum_{j=1}^{N} B_{mj}^{-1}(a_j - ln(d_j)) - \sum_{j=1}^{N} b_{jm}\frac{d_j}{d_m} \tag{2.31}$$

Finally the H_{ij}^{th} term of the Hessian matrix is given by:

$$H[\mathbf{D}]_{ij} = -\frac{B_{ij}^{-1}}{d_j} - \frac{B_{ji}^{-1}}{d_i} \tag{2.32}$$

We next develop a third algorithm denoted by **ST3** for solving problem (P3). This algorithm is an extension of (**ST2**). In this algorithm apart from utilizing a local quadratic approximation for r(**D**), the cost part c(**D**) of the objective is replaced by its linear approximation. As the assumed cost function is piecewise convex (see [16]) over its entire domain, subgradients of the cost function will always exist. These subgradients can be computed by solving the dual of the optimization problem (D1) given by (D2). The reason for this follows from **strong duality** of linear optimization (refer to [16]). According to this theorem, the decision variables in problem (D2) represent the **marginal costs** associated with the constraints of optimization problem (D1). In other words the dual variables can be interpreted as the improvement in the objective of (D1) for each additional/less unit of right hand side of constraints in the optimization problem. The other point to note is that since the cost function is piecewise convex, more than one value for the subgradients can exist as well but for constructing a linear approximation of the cost function, it will suffice to consider only the dual variables z_{it} of the dual optimization problem (D2) (for an extended discussion see [15]).

$$D2 : max_{y,z} \sum_{i,t} \overline{d}_{it} z_{it} - \sum_{s} C_s y_s \tag{2.33}$$

subject to:

$$z_{it} - y_s \leq b_{it}^s, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, .., T \quad \forall s = 1, 2, ..., T \tag{2.34}$$

$$y_s \geq 0, \quad \forall s = 1, 2, ..., T \tag{2.35}$$

$$z_{it} \geq 0, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, .., T \tag{2.36}$$

The new problem formulation $(P4)$ is based on a local quadratic approximation of r(**D**) and a linear approximation for the cost c(**D**) is included next.

$$P4 : max_D \;\; r(\overline{D}) + \nabla r(\overline{D})^T(D - \overline{D}) + \frac{1}{2}(D - \overline{D})^T.diagH(\overline{D})(D - \overline{D}) - z^T(D - \overline{D}) \tag{2.37}$$

$$\sum_{i=1}^{N}\sum_{t=1}^{g} d_{it} \leq \sum_{s=1}^{g} C_s, \quad \forall g = 1, 2, ..., T \tag{2.38}$$

$$e^{a_i} \geq d_{it}, \quad \forall i = 1, 2, .., N \quad \forall t = 1, 2, .., T \tag{2.39}$$

The algorithm for solving the JPICP problem based on optimization models $(D2)$ and $(P4)$ is outlined next.

Step 1:

Start with a feasible point $\overline{D}$ in the feasible region of the problem $(P4)$. Compute $\nabla r(\overline{D})$, diag$H(\overline{D})$ at this point.

Step 2:

Solve the optimization problem (D2).

Step 3:

Construct a local quadratic model of r((D)) at $\overline{D}$ and a linear model of the cost function c(**D**) based on the optimal decision variables z_{it} of (D2).

Step 4:

Solve the optimization problem $(P4)$ using a QP solver such as CPLEX.

Step 5:

Check the difference between the optimal demand D and $\overline{D}$. If this difference is less than a fixed tolerence ($= 10^{-8}$), stop, otherwise replace $\overline{D}$ with the optimal demand D, recompute $\nabla r(\overline{D})$, diag$H(\overline{D})$ and goto step 2.

We conclude this section with a discussion of the last algorithm **ST4** which is a hybrid of **ST3** discussed above and the algorithm in [15]. The two algorithms differ only in how the objective in the problem ($P4$) is modelled. If $\sum_{i,t} |D_{it} - \overline{D}_{it}|$ is less than ϵ ($= 3.5$), then in step 4 of the algorithm discussed above, we solve the optimization problem ($P4$), otherwise optimization problem ($P5$) (which is the model utilized in [15]) is solved. The model ($P5$) is build around a fixed Hessian in the objective and is never updated. Based on our experience this makes it difficult for the model to converge to a solution when the tolerence criteria is set to a very low value such as 10^{-8} which is the default value utilized in commercially available optimization softwares such as LOQO.

$$P5 : min_D \sum_{t=1}^{T}(\sum_{i=1}^{N}(2H_{it}d_{it} + \frac{0.025L}{\alpha}(\overline{d}_{it} - d_{it})^2)) \tag{2.40}$$

subject to:

$$H_{it} = -\sum_{j=1}^{N}(B_{ij}^{-1}(-ln(\overline{d}_{jt})+a_j))+\sum_{m=1}^{N}(B_{mi}^{-1}\overline{d}_{mt}/\overline{d}_{it})+Z_{it} \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..T \tag{2.41}$$

$$\sum_{i=1}^{N}\sum_{t=1}^{g} d_{it} \leq \sum_{s=1}^{g} C_s, \quad \forall g = 1,2,...,T \tag{2.42}$$

$$e^{a_i} \geq d_{it}, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \tag{2.43}$$

The objective of ($P5$) has a few new variables which are defined below:

$\alpha = \frac{1}{\text{current iteration number}}$

L = Lipschitz continuity constant $= 2.\sqrt{\sum_{j=1}^{N}(\sum_{i=1}^{N} b_{ij}^2)}$

For a general discussion on the principles behind the different optimization models introduced in this chapter please refer to *Nonlinear Programming* by D.P. Bertsekas [17].

2.4 Runtime Performance Comparison of the JPICP Algorithms

In this section we will compare the computational performance and accuracy of the different algorithms we presented in Section 2.3. The models were implemented in AMPL and utilized either LOQO (for **ST1**) or CPLEX as their solver to solve the different optimization problems. AMPL is a high level mathematical programming language that allows for rapid prototyping of optimization models and allows the user to choose the most suitable solver for the optimization model being implemented from a wide variety of available solvers. CPLEX is a high performance optimizer. It is one of the fastest available solvers for solving linear programs, quadratic programs, quadratically constrained programs and mixed integer programs. LOQO is another optimizer that is designed to solve **any** smooth nonlinear optimization problem. It is based on interior point methods and in particular, it utilizes a primal-dual path following algorithm. The data files were generated utilizing MATLAB. The instances included in this section were run on a shared computer with two Intel Xeon 1500 MHz processor and combined cache of 512 KB (256 KB each).

All model parameters were randomly generated from a uniform distribution and then modified if required to ensure that the generated instances satisfy Assumptions 2.1 - 2.5 discussed in Section 2.3. The production and inventory holding cost/period were generated from a uniform distribution between 0 and 1. The '**a**' vector denoting the demand when the price of the products equals zero was generated utilizing a uniform distribution between 1 and 2. The inverse of the price sensitivity matrix was randomly generated from a uniform distribution and modified to ensure that it is invertible. The joint production capacities were generated based on the initial feasible demand vector $\overline{D}$ and then adjusted to ensure that the capacity constraint is tight in at least some of the time - periods under consideration.

The solvers (i.e. CPLEX and LOQO) were used with their default settings. The tolerance parameter for the iterative schemes (**ST2, ST3 and ST4**) was set to 10^{-8}. The time required by the different algorithms to generate the optimal solution is in terms of **cpu seconds**. This time is the sum of two components:

1. The CPU time utilized by the solver to solve the optimization problem. This can be monitored by utilizing the AMPL command **_total_solve_time**. Alternatively the user can keep track of the time required to solve the optimization problem each time by using **_solve_time**.

2. The CPU time utilized by the AMPL script itself to perform such tasks as computing the first and second derivatives and updating other iteration specific variables. The AMPL script time can be monitored utilizing **_ampl_time**.

Figure 2-2 shows the computational performance of different JPICP algorithms. For a small number of products and time periods all four algorithms are comparable in terms of total time required to solve the problem. However as the number of products and time periods are simultaneously increased, the algorithms begin to differ in their computational performance. **ST4** is clearly the fastest, followed by **ST3** and **ST2**. Table 2.1 compares the optimal profit generated by the diffrent solution algorithms. Table 2.2 compares the L1 norm of the optimal demand (i.e. $\sum_{t=1}^{T}\sum_{t=1}^{N} d_{it}$) generated by the different algorithms. The optimal profit and L1 norms of the optimal demand generated by all four algorithms is in close agreement reflecting the fact that all four methods proposed in Section 2.3 have a high degree of accuracy. Figure 2-3 and tables 2.1 and 2.4 present similar information for sample instances in which the number of products was fixed to 100 and only the number of time periods was varied from 10 to 60. Again **ST4** is the fastest algorithm and the optimal problem parameters (i.e.

L1 norm of demand and profit) are in close agreement.

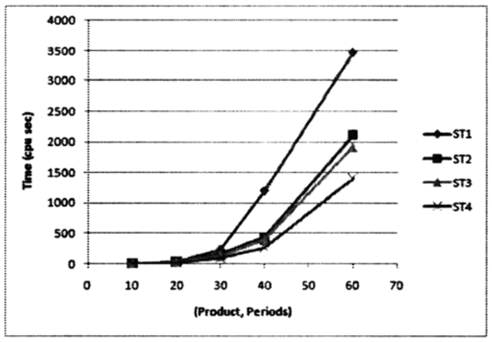

Figure 2-2: Run time comparison of different algorithms. (Product, Periods) denotes that a value of 10 on the x - axis indicates that the instance had ten products and ten time - periods

(Products,Periods)	Profit (ST1)	Profit (ST2)	Profit (ST3)	Profit (ST4)
10,10	7202.05	7202.05	7202.05	7202.05
20,20	53145.9	53145.9	53145.9	53144.4
30,30	174949	174949	174949	174946
40,40	436998	436998	436998	436943
60,60	1437430	1437430	1437430	1437430

Table 2.1: Comparison of optimal profit generated by different algorithms

(Products,Periods)	Dstart	Dopt (ST1)	Dopt (ST2)	Dopt(ST3)	Dopt(ST4)
10,10	470.9096	188.478	188.478	188.478	188.478
20,20	1734.1	708.424	708.424	708.424	709.236
30,30	3471.6	1437.73	1437.73	1437.69	1438.62
40,40	6482	2675.85	2675.85	2675.85	2682.8
60,60	14463	5979.53	5979.53	5979.53	5994.85

Table 2.2: Comparison of optimal L1 norms of optimal demand generated by different algorithms

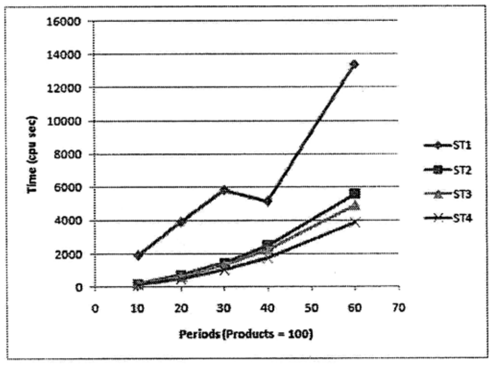

Figure 2-3: Run time comparison of different algorithms. In these instances the total number of products was fixed at 100 and only time periods were varied

(Products,Periods)	Profit (ST1)	Profit (ST2)	Profit (ST3)	Profit (ST4)
100,10	749526	749526	749526	749481
100,20	1429830	1429830	1429830	1429680
100,30	2075240	2075260	2075260	2075040
100,40	2795550	2795550	2795550	2795290
100,60	4258260	4258360	4258360	4257900

Table 2.3: Comparison of optimal profit generated by different algorithms

(Products,Periods)	Dstart	Dopt (ST1)	Dopt (ST2)	Dopt(ST3)	Dopt(ST4)
100,10	4367.6	1791.67	1791.67	1791.67	1794.02
100,20	8395.4	3455.47	3455.47	3455.47	3462.77
100,30	12137	5018.74	5018.74	5018.74	5030.57
100,40	16063	6645.66	6645.66	6645.66	6661.09
100,60	24919	10277.7	10277.7	10277.7	10305.6

Table 2.4: Comparison of optimal L1 norms of optimal demand generated by algorithms

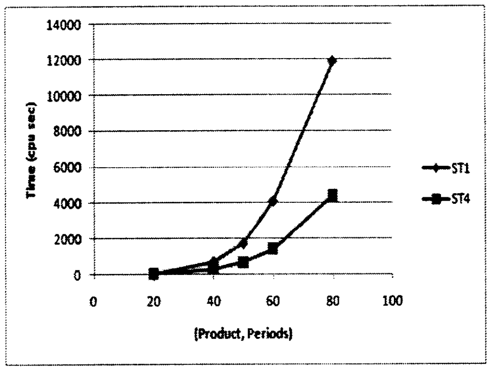

Figure 2-4: Run time comparison of ST1 and ST4 for a randomly generated starting point

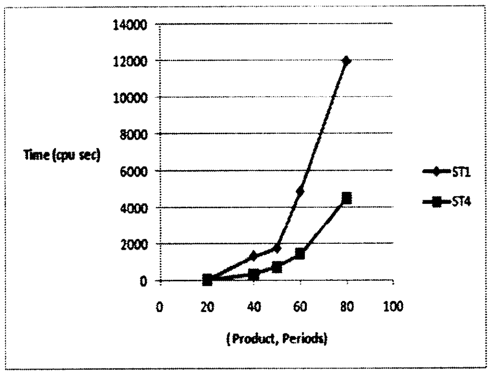

Figure 2-5: Run time comparison of ST1 and ST4 for a starting point far from the optimal solution

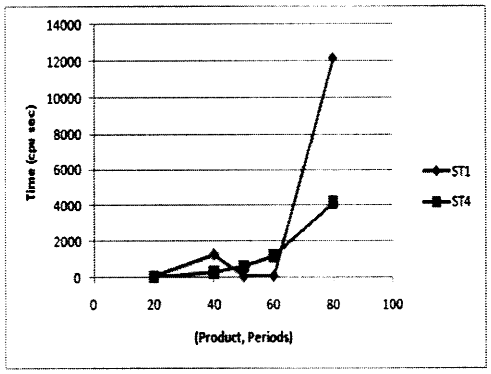

Figure 2-6: Run time comparison of ST1 and ST4 for a starting point close to the optimal solution

As **ST1** utilizes the LOQO solver which in turn utilizes interior point method (IPM) to locate the optimal solution, its performance can be expected to vary depending on

(Products,Periods)	Profit (ST1)	Profit (ST4)
20,20	57492.1	57465.6
40,40	463761	463727
50,50	873333	873273
60,60	1454250	1454120
80,80	3470420	3470370

Table 2.5: Comparison of optimal profit generated by ST1 and ST4

(Products,Periods)	D(Small)	D(Medium)	D(Large)	Dopt(ST1)	Dopt(ST2)
20,20	848.0227	1756.1	1885.8	718.303	722.574
40,40	3243.6	6692.9	7185.6	2750.86	2756
50,50	4944.8	10188	10967	4195.84	4204.11
60,60	6970.4	14320	15370	5920.45	5933.11
80,80	12327	25295	27148	10474.4	10482.7

Table 2.6: Comparison of starting L1 norms of the initial feasible demand

whether the initial feasible demand (or $\overline{D}_{init}$) is very close or far from the optimal demand. The other three algorithms utilize network and normal versions of the simplex algorithm implemented in the CPLEX solver, the performance of which, although to some extent does depend on the starting point but this dependance is relatively small compared to an IPM based procedure. Figures 2-4 - 2-6 compare the total time taken by **ST1** (which utilizes IPM) and **ST4** (which utilizes network and normal versions of the simplex method) to solve the same data instances but each time starting from a different starting point. From the figures it can be verified that the IPM based algorithm **ST1** is much more sensitive to the starting point than the simplex method based algorithm **ST4**. The algorithm **ST4** takes about the same amount of time in all three cases and for the relatively large size instance comprising of 80 products in 80 time - periods delivers superior performance in all the three scenarios. The starting points (classified as D(Small), D(Medium) and D(Large) depending on how close their L1 norm is to the L1 norm of the optimal demand Dopt) are included in Table 2.6.

2.5 Conclusions

In this chapter we introduced four algorithms for solving the JPICP problem based on the observation that the cost part of the problem can be effectively transformed into a linear transportation problem. All four algorithms were shown to be accurate and computationally tractable although they do differ substantially in their computational efficiency. For the data instances utilized in this work, **ST4** was computationally the most efficient of the four developed algorithms although **ST3** comes very close to **ST4** in terms of computational efficiency and is more accurate than **ST4** in terms of the optimal solution computed by the model. The algorithms developed in this chapter have a much higher degree of accuracy (10^{-8} vs. 0.05) than the algorithm developed in the earlier work by Rao [15]. Also **ST4** runs faster for comparable data instances when compared with the algorithm in [15], if the tolerence levels are set to be the same. We also demonstrated that the choice of solver can have implications on the computational efficiency of the models as some solvers are based on their in built algorithms and are more sensitive to the initial starting point of optimization problems than others. An interesting avenue for further work would be to test the four algorithms developed in this chapter on real data instances procured either from supermarkets or other similar establishments where the deterministic log linear demand function can be utilized to model the customer demand.

Chapter 3

Pricing in Response to the Multinomial Logit Demand Function

In this chapter we introduce algorithms for efficiently computing optimal prices when demand follows the Multinomial Logit (MNL) demand model. In Section 3.1 we give a broad overview of discrete choice models, their relative advantages over deterministic demand models and algorithms developed to locate the optimal solution in response to demand that follows the MNL model. This is followed by Section 3.2 which contains a description of the most popular and widely used discrete choice model i.e., the Multinomial Logit (MNL) model. In Section 3.3, we provide a description of an algorithm developed based on the Karush Kuhn Tucker (KKT) optimality conditions. In Section 3.4 we proceed to describe an alternative algorithm based on a logarithmic reformulation of the MNL based revenue model. Computational performance of the algorithms is documented in Section 3.5. Finally, Section 3.6 wraps up the chapter with a brief discussion on relative advantages and disadvantages of the developed procedures.

3.1 Problem Overview

3.1.1 Discrete Choice Models

Discrete choice models (DCM) are probabilistic models that describe the decisions made by individuals while choosing from a discrete set of alternatives. DCMs have several advantages over deterministic demand models [19]. They can be used to represent differences in preference among customers of a particular market segment. They are also suitable for situations where customers exhibit a time varying variety seeking behavior. Also being probabilistic models that model uncertainty in choice outcomes, with DCMs one does need to identify and separate all pertinent variables governing customer behavior. This kind of modelling of random customer choice behavior was introduced for the first time by Luce [20] and perfected by McFadden [21]. An overview of important families of DCMs and their properties was given by McFadden in his 2000 Nobel lecture [21].

In DCMs the expected benefit of a particular choice to the customer is given by utility functions and the customer chooses the alternative with the highest utility. More formally let there be 'n' alternatives in a choice set 'S', denoted by j = 1,2,...,n. A customer's utility for each of these products is given by:

$$U_j = u_j + \xi_j$$

In the above utility equation u_j represents the deterministic component of product j's utility which can be determined by survey data/econometric analysis (Eg. $u_j = \beta^T x$, where β is a vector of attribute weights and x is vector of observed attribute values) and ξ_j represents the mean zero random component. In DCMs the probability that a particular alternative is chosen (discrete choice probability) is given by:

$$P_j(S) = P(U_j \geq max\{U_i : i \in S\})$$

Depending on the hypothesis on the distribution of the error terms ξ_j, we get different DCMs. If the error terms are independent and identically distributed normal random variables, the resulting model is called a Probit Model. On the other hand if they have a Gumbel or a double exponential distribution the resulting model is called a Logit Model. Many other variants of DCMs based on a different assumption on the distribution of error terms exist as well. The assumption made on the error term distribution is critical as it directly relates to whether or not the resulting discrete choice probabilities have a closed form expression. In case of independent and identically distributed Gumbel random variables, it is possible to derive close form expressions for the discrete choice probabilities and this is precisely the reason for the popularity of the Multinomial Logit (MNL) demand model. Under the MNL model, the probability that an alternative 'j' is chosen is given by:

$$P_j(S) = \frac{e^{u_j/\mu}}{\sum_{i \in S} e^{u_i/\mu}}$$

The parameter 'μ' is a scale factor that is related with the Gumbel distribution. The MNL model on account of its analytical tractability has been used as the demand model of choice in a wide variety of settings. Guadagni and Little [22] employed it in a marketing context to study the brand choice behavior of coffee consumers. Ben-Akiva and Lerman [23] employed it in their study of demand forecasting for urban transportation systems. Berkovec [24] utilized it in his short run general equilibrium model of the automobile market based on demand and production rate of new automobiles and the scrappage rate of old automobiles. Train, McFadden and Ben-Akiva [25] used the MNL to study the demand for local telephone service based on residential calling patterns and service choices. Anderson and dePalma [26] employed it as a model of product differentiation in game theoretic setting.

However despite its widespread use as the most popular DCM, MNL suffers from one significant limitation. The assumption that the errors in the unobserved portion of the utility (i.e. ξ_j) are independent of each other gives rise to what is known as *Independence from Irrelevant Alternatives (IIA)* property of MNL. This property

implies that for two alternatives 'i', 'j' , the probabilities of choosing either one of them is independent of the choice sets containing these alternatives. Debreu [27, 19] highlighted this property of MNL with the help of the following example:
Suppose **S** (={Car, Bus}) represents the set of our transportation choices and we are equally likely to choose either of the alternative i.e. $P_{car}(S) = 0.5$ and $P_{bus}(S) = 0.5$. Now if we introduce a new set **T** (={Car, Red Bus, Blue Bus}), MNL predicts that $P_{car}(T) = 0.67$, $P_{redbus}(T) = 0.67$ and $P_{bluebus}(T) = 0.67$. In reality we can expect the probabilities to be $P_{car}(T) = 0.5$, $P_{redbus}(T) = 0.25$ and $P_{bluebus}(T) = 0.25$ as the color of the bus is most likely to be an irrelevant attribute of the choice we make to commute from one point to another.
IIA depending on the context can be a useful or a limiting modelling property. It restricts the use of the MNL model to choice sets that contain members that are 'equally dissimilar' in some sense. In general models where only the average behavior is under consideration violations of this assumption seem to have a less profound effect on model accuracy. The main reason for the existence of other DCMs is in part due to the IIA property of the MNL model as all the alternate models try to work around the IIA property of the MNL. Generalized Extreme Value (GEV) models, nested MNL and Mixed MNL (MMNL) are some of the MNL variants that have been specifically developed to work around the IIA property of the MNL model while still providing closed form analytical expressions for some of the parameters of interest. More information on the derivation of different kinds of DCMs and their relative advantages and disadvantages can be found in Kenneth Train's *Discrete Choice Methods with Simulation* [28]. Information specifically relevant to MNL in a multi - product setting can be found in *Theory and Practice of Revenue Mangement* by Talluri and Ryzin [19].

3.1.2 Algorithms for Pricing in Response to DCMs

In this subsection we will review some of the work that has been done in solving for optimal prices in response to the MNL demand model. MNL is the most popular and widely used discrete choice demand model. Despite its widespread popularity, revenue / profit functions resulting from a MNL based demand model are nonconvex and as a direct consequence of this the resulting optimization problems are hard to solve. Due to this, to the best of our knowledge, the algorithms that exist in the literature, cannot compute provably the globally optimal solution. In this subsection we will restrict ourselves to algorithms that find the globally optimal solution when demand follows the MNL model. One of the first algorithms for finding the global optimum when demand follows the MNL model was developed by Hanson and Martin [10]. They utilized a path following algorithm based on solving a relaxed version of the problem at each iteration. The algorithm was developed for the unconstrained model i.e., there is no restriction on the production / inventory storage capacity.

Gallego and Stefanescu [11] in their study on the effects of upgrades and upsells offered to customers on revenue streams of a firm have utilized a capacity constrained MNL model and proposed a column generation based heuristic to compute the globally optimal solution. However the computational performance of the heuristic they proposed is not known for large scale data instances. More recently Levi and Perakis [14] Keller *et al.* [13] have proposed a provally optimal algorithm for locating the global optimal solution of a MNL based capacity constrained profit function. The first lgorithm is based on the KKT optimality conditions while the latter is based on a convex reformulation of the problem. The formulation we consider in this thesis is similar to the model in [13], [14] and we will utilize this model to compare the computational efficiency of our proposed algorithms for capacity constrained MNL problem.

3.2 Description of the MNL based Model

In this section we describe the MNL based revenue model which we will use throughout this chapter. The profit maximization problem described here is similar to the one considered by Keller *et al.* [13] (see Section 3.4). We consider the case of a retailer offering a fixed assortment of substitutable products subject to capacity and inventory constraints. As the price of one product directly affects the demand for all products in the entire assortment, pricing decisions for the entire assortment of products need to be made jointly.

Consider a set **N** containing 'i' (= 1,2,...n) different products. We assume that the deterministic component of the utility function (i.e. u_i) is given by a linear in-attributes model represented by (3.1). In the equation b_i, p_i and c_i represent the price sensitivity coefficient, price and cost of product 'i' respectively.

$$u_i = x_i = b_i p_i \quad \forall i \in N \tag{3.1}$$

The demand for the different products in the assortment in the MNL setting is then given as in ([19]):

$$d_i(x) = \frac{MS.e^{-x_i}}{1 + \sum_{j \in N} e^{-x_j}} \quad \forall i \in N \tag{3.2}$$

In Equation (3.2), 'MS' represents the total number of customers in the market segment under consideration and $\frac{e^{-x_i}}{1+\sum_{j \in N} e^{-x_j}}$ represents the MNL choice probability that a particular product is chosen by the consumer. Furthermore, if we impose linear constraints on the demands to model production / capacity / inventory storage limits, the resulting optimization problem is represented by equations (3.3) - (3.5):

$$R : max_x \;\; MS. \sum_{i \in N} \frac{a_i x_i e^{-x_i}}{1 + \sum_{j \in N} e^{-x_j}} \tag{3.3}$$

subject to:

$$u_k - \sum_{i \in N} A_{ki} d_i(x) \geq 0, \quad \forall k \in M \tag{3.4}$$

$$x_i \geq 0, \quad \forall i \in N \tag{3.5}$$

In the optimization problem (R), **A** is a M x N matrix representing the linear capacity / inventory holding constraints, u_k is the maximum production /inventory holding capacity and a_i represent the inverse of the price sensitivity coefficients (or $a_i = 1/b_i$). Optimization problem (R) is in general non convex and as discussed in Section 3.1, is a hard to solve maximization problem. For the remainder of this chapter we will devise computationally tractable and efficient algorithms that are designed to locate the globally optimal solution for problem (R).

3.3 Algorithm based on the KKT Optimality Conditions

In this section we will develop an algorithm for solving the non convex optimization problem (R). This algorithm is an extension of an algorithm developed based on the KKT optimality conditions for the problem developed by Levi and Perakis [14]. The original algorithm was developed based on the following assumption:

Assumption 3.1

$$A_{ki} - \frac{u_k}{MS} \geq 0, \quad \forall k \in M, \forall i \in N$$

Assumption 3.1 is equivalent to saying that some of the constraints in problem (R) will be violated, if one allocates all demand to one product. One of the implications of the above assumption is that it ensures that the set represented by the constraints in Equation (3.4) is a convex set. Notice also that the feasible region of optimization problem (R) has a non empty interior. As a result, the Slater condition holds and the KKT optimality conditions are necessary conditions, i.e. the optimal solution satisfies these conditions. The KKT conditions (ignoring the non-negativity constraint on x_i) for the optimization problem (R) are given by Equations (3.6) - (3.8):

$$\frac{\partial R(x)}{\partial x_i} + \sum_{k \in M} \lambda_k . \frac{\partial f_k(x)}{\partial x_i} = 0 \qquad (3.6)$$

$$f_k(x)\lambda_k = 0, \quad \forall k \in M \tag{3.7}$$

$$f_k(x) \geq 0, \quad \forall k \in M \tag{3.8}$$

$$\lambda_k \geq 0, \quad \forall k \in M \tag{3.9}$$

Equations (3.6) - (3.9) utilize the following expressions:

$$C = R(x)/MS \tag{3.10}$$

$$\frac{\partial R(x)}{\partial x_i} = d_i(x)(a_i(1-x_i)+C), \quad \forall i \in N \tag{3.11}$$

$$f_k(x) = u_k - MS.\sum_{i\in N}(A_{ki} - (u_k/MS))e^{-x_i}, \quad \forall k \in M \tag{3.12}$$

$$\frac{\partial f_k(x)}{\partial x_i} = -\sum_{l\in N} A_{kl}\frac{\partial d_l(x)}{\partial x_i}, \quad \forall k \in M, \forall i \in N \tag{3.13}$$

$$\frac{\partial d_l(x)}{\partial x_i} = d_i(x)d_l(x)/MS, \quad \forall l \neq i \tag{3.14}$$

$$\frac{\partial f_l(x)}{\partial x_i} = d_i(x)^2/MS - d_i(x), \quad \forall l = i \tag{3.15}$$

If we substitute the expressions for $\frac{\partial R(x)}{\partial x_i}$ and $\frac{\partial f_k(x)}{\partial x_i}$ in Equation (3.6) and utilize the complementary slackness condition given by (3.7), a closed form expression for the globally optimal x_i^* is given in Equation (3.16) (for more details and a proof see [14]).

$$x_i^* = 1 + \frac{(C + \sum_{k\in M}\lambda_k(A_{ki} - (u_k/MS))}{a_i}, \quad \forall i \in N \tag{3.16}$$

If we replace x_i from Equation (3.16) to function f_k (see Equation (3.12)), we obtain Equation (3.17).

$$f_k^C(x) = u_k - MS.\sum_{i\in N}(A_{ki} - (u_k/MS))e^{-(1+\frac{(C+\sum_{k\in M}\lambda_k(A_{ki}-(u_k/MS)))}{a_i})}, \quad \forall k \in M \quad (3.17)$$

Conditions (3.7) - (3.9) form the nonlinear complementarity problem (NCP(f^C)) (see [12]). Solving the (NCP(f^C)) problem is equivalent to solving the optimization problem (O1(C)) as the Jacobian matrix of vector **f** is symmetric and positive semi-definite. As a result multipliers λ can also be computed by solving the optimization problem (O1(C)) which utilizes the assumed optimal value of C (for a proof see [14]).

$$O1(C) : min_\lambda \ \sum_{k=1}^{M} u_k\lambda_k + MS.\sum_{i=1}^{N} a_i e^{-(1+\frac{(C+\sum_{l\in M}\lambda_l(A_{li}-(u_l/MS)))}{a_i})} \quad (3.18)$$

subject to:

$$\lambda_k \geq 0, \forall k \in M \quad (3.19)$$

After solving minimization problem O1(C), we will have all the required parameters to evaluate Equation (3.16). This is an important observation to be kept in mind as we proceed next to derive a scheme to generate a sequence of Cs, that will lead us to the globally optimal solution for problem (R).

An iterative scheme to generate a sequence of ansatz/guesses for the optimal value of C can be devised by substituting in Equation (3.10) the expression for R(x). After rearranging and cancelling some terms, the result will be Equation (3.20), where the only unknown is C. This equation can be solved for the optimal value of C^*, by utilizing a binary search or a Newton based procedure for solving one equation with one unknown.

$$C - \sum_{i=1}^{N}(a_i + \sum_{k=1}^{M}\lambda_k(A_{ki} - (u_k/MS)))e^{-x_i} = 0 \quad (3.20)$$

The previous discussion gives rise to the following algorithm (**T1**) for finding the optimal value of objective ($R(x)$) (see [14] for more details):

Step 1:

Set upper bound U = $(1/e)\sum_{i=1}^{N} a_i$

Set lower bound L = 0

Set C = (U+L)/2

Set $\lambda_k = 0$, $\forall k \in M$

Step 2:

For the given C, solve the optimization problem (O1(C))

Step 3:

Compute $x_i = \frac{C+\sum_{k=1}^{M}\lambda_k(A_{ki}-(u_k/MS))}{a_i} + 1$, $\forall$ i $\in$ N

Step 4:

Compute F(C) = $\sum_{i=1}^{N}(a_i + \sum_{k=1}^{M}\lambda_k(A_{ki} - (u_k/MS)))e^{-x_i}$

Step 5:

Compute D = C - F(C)

For D = 0, goto step 8

Step 6:

For D > 0 , set U = C

For D < 0 , set L = C

Step 7:

Compute U - L. For (U-L) > 10^{-8}, goto step 2

Step 8:

Compute optimal Revenue, R(x) = MS.$\sum_{i=1}^{N} \frac{a_i x_i e^{-x_i}}{1+\sum_{j=1}^{N} e^{-x_j}}$

This algorithm utilizes a binary search to solve Equation (3.19). This is justified as, $D(C) = C - F(C)$, is a monotonically increasing function of C. Furthermore, we can develop an upper and lower bound for the optimal C^*. The upper bound utilized in the above algorithm was selected based on run time observations and is a valid upper bound for all data instances considered in this chapter.

Algorithm **T1** though tractable and guaranteed to converge to the global optimal of the problem is computationally expensive. The main reason for this is step 2, which requires solving a relatively hard optimization problem (O1(C)) at each iteration. This problem can be alleviated by utilizing appropiate upper and lower bounds on $D(C_{current}, \lambda)$. Specifically bounds on $D(C_{current}, \lambda)$ can potentially help to improve the computational efficiency of the algorithm by cutting down on the number of times the optimization problem $(O1)$ is solved. This will be the case if the bounds on $D(C_{current}, \lambda)$ are constructed utilizing multipliers $\overline{\lambda}$ that have been computed at some point in past iterations. In particular, let us assume that we have two bounds, D_u and D_l, such that $D_l(C_{current}, \overline{\lambda}) \leq D(C_{current}, \lambda) \leq D_u(C_{current}, \overline{\lambda})$, where multipliers $\overline{\lambda}$ solve optimization problem $(O1)$ at some past iteration. If $0 \leq D_l(C_{current}, \overline{\lambda}) \leq D(C_{current}, \lambda) \leq D_u(C_{current}, \overline{\lambda})$, then the upper bound U in the algorithm discussed above, can be updated to $C_{current}$, without computing the multipliers λ through solving the optimization problem (O1) at that particular step. Similarly if $D_l(C_{current}, \overline{\lambda}) \leq D(C_{current}, \lambda) \leq D_u(C_{current}, \overline{\lambda}) \leq 0$, then the lower bound L in the algorithm can be updated to $C_{current}$, without computing the exact solution to $(O1)$ at that particular step. Based on these observations we can construct a new algorithm (**T2**) to find the optimal solution for problem (R). However, before outlining the idea behind the exact new algorithm **T2**, it will be worthwile to construct the required upper and lower bounds $D_u(C, \overline{\lambda})$ and $D_l(C, \overline{\lambda})$.

Proposition 3.1: If $\overline{\lambda}$ are the optimal multipliers from minimization problem $(O1(\overline{C}))$ for $\overline{C}$ and $\lambda_{current}$ is the optimal solution to minimization problem $(O1(C_{current}))$ for $C_{current}$, then a lower bound for D is given by $D_l(C_{current}, \overline{\lambda}) = C_{current} - (1/MS).[\sum_{k\in M} u_k\overline{\lambda} + MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\overline{\lambda})}]$

Proof:

$$O1(C_{current}, \lambda^{current}) \le O1(C_{current}, \overline{\lambda}) \tag{3.21}$$

$$\sum_{k\in M} u_k \lambda_k^{current} + MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\lambda^{current})} \le \sum_{k\in M} u_k\overline{\lambda}_k + MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\overline{\lambda})} \tag{3.22}$$

This follows from the facts that (O1) is a minimization problem and $\lambda_{current}$ solves the optimization problem (O1($C_{current}$)). $\overline{\lambda}$ is a feasible vector. As $\lambda^{current}$ minimizes (O1($C^{current}$)), utilizing NCP($f^{current}$) implies

$$\lambda_k^{current} f_k^{C_{current}}(\lambda_{current}) = 0, \quad \forall k \in M \tag{3.23}$$

$$u_k\lambda_k^{current} = MS.\sum_{i\in N} \lambda_k^{current}(A_{ki} - (u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))} \tag{3.24}$$

Summing over all the constraints,

$$\sum_{k\in M} u_k\lambda_k^{current} = MS.\sum_{i\in N}\sum_{k\in M} \lambda_k^{current}(A_{ki} - (u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))} \tag{3.25}$$

Utilizing the above expression for $\sum_{k\in M} u_k\lambda_k^{current}$ in equation (3.22)

$$MS\sum_{i\in N}(\sum_{k\in M} \lambda_k^{current}(A_{ki} - (u_k/MS)) + a_i)e^{(-x_i(C_{current},\lambda^{current}))} \tag{3.26}$$

$$\le$$

$$\sum_{k\in M} u_k\overline{\lambda}_k + MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\overline{\lambda})}$$

$$\sum_{i \in N}(\sum_{k \in M} \lambda_k^{current}(A_{ki} - (u_k/MS)) + a_i)e^{(-x_i(C_{current}, \lambda^{current}))} \tag{3.27}$$

$$\leq$$

$$\frac{1}{MS}(\sum_{k \in M} u_k \overline{\lambda}_k + MS. \sum_{i \in N} a_i e^{-x_i(C_{current}, \overline{\lambda})})$$

Multiplying both sides of the above equation by -1 and adding $C_{current}$.

$$C_{current} - \sum_{i \in N}(\sum_{k \in M}(A_{ki} - (u_k/MS)) + a_i)e^{(-x_i(C_{current}, \lambda^{current}))} \tag{3.28}$$

$$\geq$$

$$C_{current} - \frac{1}{MS}(\sum_{k \in M} u_k \overline{\lambda}_k + MS. \sum_{i \in N} a_i e^{-x_i(C_{current}, \overline{\lambda})})$$

$$D(C_{current}, \lambda^{current}) \geq D_l(C_{current}, \overline{\lambda}) \qquad \bullet$$

Proposition 3.2: If $\overline{\lambda}$ are the optimal multipliers to minimization problem $(O1(\overline{C}))$ for $\overline{C}$, $\lambda_{current}$ is the optimal solution to minimization problem $(O1(C_{current}))$ for $C_{current}$ and bdd is the bound given by Equation (3.29), then an upper bound for D is given by $D_u(\overline{C}, \overline{\lambda}) = C_{currrent} - F(\overline{C}, \overline{\lambda}).min(1, \frac{1}{bdd})$

$$bdd = max_i \quad e^{\frac{(C_{current} - \overline{C})}{a_i}} \tag{3.29}$$

Proof:

$$O1(\overline{C}, \overline{\lambda}) \leq O1(\overline{C}, \lambda^{current}) \tag{3.30}$$

This follows from the fact that (O1) is a minimization problem. Substituting the expression for the objective function in problem (O1) and rearranging the terms we get,

$$\sum_{k \in M} u_k \overline{\lambda_k} + MS. \sum_{i \in N} a_i e^{-x_i(\overline{C}, \overline{\lambda})} \leq \sum_{k \in M} u_k \lambda_k^{current} + MS. \sum_{i \in N} a_i e^{-x_i(\overline{C}, \lambda^{current})} \tag{3.31}$$

$$\sum_{k\in M} u_k\overline{\lambda_k}+MS.\sum_{i\in N} a_i e^{-x_i(\overline{C},\overline{\lambda})} \leq \sum_{k\in M} u_k\lambda_k^{current}+MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\lambda^{current})} e^{\frac{(C_{current}-\overline{C})}{a_i}} \tag{3.32}$$

$$\sum_{k\in M} u_k\overline{\lambda_k} + MS.\sum_{i\in N} a_i e^{-x_i(\overline{C},\overline{\lambda})} \leq \sum_{k\in M} u_k\lambda_k^{current} + MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\lambda^{current})}.bdd \tag{3.33}$$

As $\overline{\lambda}$ minimizes O1($\overline{C}$), utilizing NCP($f^{\overline{C}}$), we have,

$$\overline{\lambda_k} f_k^{\overline{C}}(\overline{\lambda}) = 0, \quad \forall k \in M \tag{3.34}$$

$$u_k\overline{\lambda_k} = MS.\sum_{i\in N} \overline{\lambda}_k(A_{ki} - (u_k/MS))e^{(-x_i(\overline{C},\overline{\lambda}))} \tag{3.35}$$

Summing over all the constraints,

$$\sum_{k\in M} u_k\overline{\lambda}_k = MS.\sum_{i\in N}\sum_{k\in M} \overline{\lambda}_k(A_{ki} - (u_k/MS))e^{(-x_i(\overline{C},\overline{\lambda}))} \tag{3.36}$$

Similarly,

$$\lambda_k^{current} f_k^{C_{cuurent}}(\lambda_{current}) = 0, \quad \forall k \in M \tag{3.37}$$

$$u_k\lambda_k^{current} = MS.\sum_{i\in N} \lambda_k^{current}(A_{ki} - (u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))} \tag{3.38}$$

Summing over all the constraints,

$$\sum_{k\in M} u_k\lambda_k^{current} = MS.\sum_{i\in N}\sum_{k\in M} \lambda_k^{current}(A_{ki} - (u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))} \tag{3.39}$$

Utilizing the above expressions following from NCP($f^{\overline{C}}$), NCP($f^{C_{current}}$) in Equation (3.33), we have,

$$MS.\sum_{i\in N}\sum_{k\in M}\overline{\lambda}_k(A_{ki}-(u_k/MS))e^{(-x_i(\overline{C},\overline{\lambda}))}+MS.\sum_{i\in N}a_ie^{-x_i(\overline{C},\overline{\lambda})}$$
$$\leq$$
$$MS.\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}+MS.\sum_{i\in N}a_ie^{-x_i(C_{current},\lambda^{current})}.bdd \tag{3.40}$$

Cancelling common terms and after some rearrangement, Equation (3.40) can be written as:

$$\begin{aligned}&\sum_{i\in N}(\sum_{k\in M}\overline{\lambda}_k(A_{ki}-(u_k/MS))e^{(-x_i(\overline{C},\overline{\lambda}))}+a_i)e^{-x_i(\overline{C},\overline{\lambda})}\\ &\qquad\leq\\ &\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}\\ &+bdd.[\sum_{i\in N}(a_i+\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))})]\\ &-(\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}).bdd\end{aligned} \tag{3.41}$$

$$F(\overline{C},\overline{\lambda})\leq\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}(1-bdd)+bdd.F(C_{current},\lambda^{current}) \tag{3.42}$$

Now we need to consider two cases:

(1) $C_{current}\geq\overline{C}$

In this case $bdd\geq 1$, therefore we can drop the negative term $\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}(1-bdd)$ from the RHS of Equation (3.42), to get

$$(1/bdd).F(\overline{C},\overline{\lambda})\leq F(C_{current},\lambda^{current}) \tag{3.43}$$

(2) $C_{current} < \overline{C}$

In this case $0 < bdd < 1$, so we can omit bdd from the term $MS.\sum_{i\in N} a_i e^{-x_i(C_{current},\lambda^{current})}.bdd$ in the RHS of Equation (3.40) to get,

$$MS.\sum_{i\in N}\sum_{k\in M}\overline{\lambda}_k(A_{ki}-(u_k/MS))e^{(-x_i(\overline{C},\overline{\lambda}))}+MS.\sum_{i\in N}a_i e^{-x_i(\overline{C},\overline{\lambda})}$$
$$\leq$$
$$MS.\sum_{i\in N}\sum_{k\in M}\lambda_k^{current}(A_{ki}-(u_k/MS))e^{(-x_i(C_{current},\lambda^{current}))}+MS.\sum_{i\in N}a_i e^{-x_i(C_{current},\lambda^{current})} \tag{3.44}$$

This is same as,

$$F(\overline{C},\overline{\lambda}) \leq F(C_{current},\lambda^{current}) \tag{3.45}$$

Cases (1) and (2) can be combined into one equation by:

$$min(1,\frac{1}{bdd}).F(\overline{C},\overline{\lambda}) \leq F(C_{current},\lambda^{current}) \tag{3.46}$$

Multiplying both sides of equation (3.46) by -1 and adding $C_{current}$ to both sides, we get

$$C_{current} - min(1,\frac{1}{bdd}).F(\overline{C},\overline{\lambda}) \geq C_{current} - F(C_{current},\lambda^{current}) \tag{3.47}$$

or

$$D_u(\overline{C},\overline{\lambda}) \geq D(C_{current},\lambda^{current}) \qquad \bullet$$

Using the expressions for D_u and D_l derived above, we next proceed to outline the algorithm **T2** that can be utilized to locate the optimal solution to optimization problem (R).

Step 1:

Set upper bound U = (1/e)$\sum_{i=1}^{N} a_i$

Set lower bound L = 0

Set $\lambda_k = 0$, $\forall k \in M$

Set $\overline{C} = (U + L)/2$

Step 2:

Set C = (U+L)/2

Compute $x_i = \frac{C+\sum_{k=1}^{M} \lambda_k (A_{ki} - (u_k/MS))}{a_i} + 1$, $\forall$ i $\in$ N

Step 3:

Compute D_l, utilizing the currently stored value of λ and x_i, computed in step 2

Step 4:

For $D_l > 0$, set U = C, goto step 7

For $D_l < 0$, goto step 5

Step 5:

Recompute x_i based on $\overline{C}$ and the currently stored value of λ. Compute D_u

For $D_u < 0$, set L=C, goto step 7

Step 6:

Recompute λ by solving (O1(C)). Set $\overline{C}$ to C

Compute x_i

Compute D

For D > 0, set U = C

For D < 0, set L = C

Step 7:

Compute U - L. For (U-L) > 10^{-8}, goto step 2

Step 8:

Compute the optimal Revenue, R(x) = MS.$\sum_{i=1}^{N} \frac{a_i x_i e^{-x_i}}{1+\sum_{j=1}^{N} e^{-x_j}}$

Theorem 3.1:

Under Assumption 3.1, the iterates of algorithm **T2** converge to the globally optimal solution of problem (R).

Proof:
The proof follows easily from Propositions 3.1 and 3.2 and from the fact that the original algorithm **T1** is guaranteed to locate the globally optimal solution of problem (R).

In the actual implementation of the above algorithm, in step 6 the λs were only computed if D_u - $\epsilon > 0$ and (U-L) $> \kappa$. This was done because D_u is a tight bound for the actual D, so in cases when $D_u \geq \epsilon \geq 0 \geq D_l$, computationally it was observed that it is very likely that even the actual D is positive so we can safely update the upper bound U, without computing the actual λs. This step also utilizes the fact that if the difference between $(U - L) \leq \kappa$, than the current vector λ is '*on average*' close enough to the vector λ at the optimal solution C^*, so we can again skip the computationally expensive process of solving the optimization problem $(O1)$. These parameters (i.e. κ, ϵ) may need to be tuned depending on the data instances under consideration.

We conclude this section by discussing another algorithm **T3** for solving optimization problem (R). This algorithm was derived from the fact that the optimal solution of the unconstrained problem (i.e. version of (R) without the contraints) is itself a natural upper bound on the revenue of the capacity constrained problem (R). So if we were to solve the unconstrained version of (R), we could use the optimal objective value (optimal profit C) as an upper bound U for the constrained version of (R). As solving the unconstrained version is much faster (as there no λs to compute at each iteration), we start solving the constrained version of the problem through solving the unconstrained version of (R). We continue to do so till its time to update the lower bound on the revenue. At this point as the lower bound of the unconstrained problem tells us nothing about the lower bound of the constrained problem, we need to compute the λs. From this point on in the algorithm we only need to compute the

λs at each iteration if $D \leq D_l$. However computationally it was observed that the set of λs is computed only at the point when the lower bound is updated for the first time. This *on average* is a good approximation of the actual set of λs that hold at the optimal solution provided the difference between the upper and lower bounds $(U - L)$ is less than ρ. If not the set of λs may need to be recomputed again when the lower bound is updated $(D \leq D_l)$. This discussion is somewhat justified by Proposition 3.1 (i.e. since $D \geq D_l$). The observations summarized above in general hold good for the data instances utilized for this thesis. For the trial data the value of ρ was set to 1. In general the parameter ρ may need to be tuned for the algorithm **T3** to generate accurate results. The complete algorithm **T3** is outlined below:

Step 1:

Set upper bound U = $(1/e)\sum_{i=1}^{N} a_i$

Set lower bound L = 0

Set C = (U+L)/2

Set $\rho = 1$

Set $\lambda_k = 0, \forall k \in M$

Step 2:

Compute $x_i = \frac{C+\sum_{k=1}^{M}\lambda_k(A_{ki}-(u_k/MS))}{a_i} + 1$, $\forall$ i $\in$ N

Step 3:

Compute F(C) = $\sum_{i=1}^{N}(a_i + \sum_{k=1}^{M}\lambda_k(A_{ki} - (u_k/MS)))e^{-x_i}$

Step 4:

Compute D = C - F(C)

For D = 0, goto step 7

Step 5:

For $D \geq D_l > 0$, set U = C

For D < 0 , solve the optimization problem (O1) if (U-L) > ρ and set L = C

where O1 = min $\sum_{i=1}^{M} u_i\lambda_i + MS.\sum_{i=1}^{N} a_i e^{-x_i}$

subject to: $\lambda_i \geq 0$, $\forall$ i $\in$ M

Step 6:

Compute U - L. For (U-L) > 10^{-8}, goto step 2

Step 7:

Compute optimal Revenue, R(x) = MS.$\sum_{i=1}^{N} \frac{a_i x_i e^{-x_i}}{1+\sum_{j=1}^{N} e^{-x_j}}$

3.4 Algorithm based on Log Reformulation of the MNL Demand Model

In this section we outline a reformulation of the problem (**T4**) developed by Keller *et al.* [13] to solve optimization problem (R). Solving this reformulation (**T4**) is based on a convex reformulation of the problem and is guaranteed to converge to the global optimal. To explain this reformulation, we need to redefine some of the model parameters in terms of new variables. The new parameters and variable substitutions are defined next in Equations (3.48) - (3.50).

$$\theta_o = \frac{1}{1+\sum_{j\in N} e^{-x_j}} \tag{3.48}$$

$$\theta_i = (d_i(x)/MS) = e^{-x_i}\theta_0, \quad \forall i \in N \tag{3.49}$$

$$x_i = -log(\frac{\theta_i}{\theta_o}), \quad \forall i \in N \tag{3.50}$$

Utilizing the new decision variables θ_i and θ_o defined above, the optimization problem R can be re - written as:

$$R(\theta) : max \quad -MS.\sum_{i\in N} a_i\theta_i log(\frac{\theta_i}{\theta_o}), \quad \forall i \in N \tag{3.51}$$

subject to:

$$u_k - MS.\sum_{i\in N} A_{ki}\theta_i(x) \geq 0, \quad \forall k \in M \tag{3.52}$$

$$\sum_{i=0}^{N} \theta_i = 1 \tag{3.53}$$

$$\theta_i \geq 0, \quad \forall i \in N \tag{3.54}$$

$$\theta_o \geq 0 \tag{3.55}$$

Equation (3.53) represents an additional constraint that ensures that the probabilities sum to one as θ_i in this formulation denotes the choice probability that a particular alternative is chosen.

3.5 Runtime Performance Comparison of Different MNL Pricing Algorithms

In this section we will compare the computational performance and accuracy of the different solution algrithms presented in Section 3.3. The models were implemented in AMPL and utilized LOQO in order to solve the different optimization problems. The data instances included in this section were run on a shared computer with two Intel Xeon 1500 MHz processor and combined cache of 512 KB (256 KB each). All data instances were randomly generated. The **A** matrix required in the models was generated randomly from a uniform distribution between (0,1] in such a way that Assumption 3.1 held for all trial data instances. The price sensitivity vector **b** was randomly generated from a uniform distribution between [0.5,2.5]. The capacity vector u was generated from a normal distribution with mean N (where N is the total number of products) and standard deviation 10, which was further scaled up (for MS $> 10^5$) or down (for MS $< 10^5$). The starting point of θs for algorithm **T4** were initialized to $\frac{1}{n+1}$ while the upper bound for C in the KKT based algorithms was initialized to $\frac{1}{e} \cdot \sum_{i \in N}(1/b_i)$.

In Tables 3.1 and 3.2 we compare the optimal results and computation times for algorithms **T1** and **T4** for **unconstrained data instances** of problem (R). A one to one comparison of the optimal values and computation times of the two algorithms reveals that both algorithms are accurate and efficient for the unconstrained instances of the problem. Algorithm **T4** can be seen to be typically twice as fast as algorithm **T1** for the trial data utilized for these experiments.

Products	Revenue (T1)	Time (T1)	Revenue (T4)	Time (T4)
25	104641	0.044002	104641	0.04
100	241867	0.152008	241867	0.068001
400	340056	0.48803	340056	0.236014
900	456213	1.1	456213	0.424024
1600	528972	2.08413	528972	0.976055

Table 3.1: Run-time comparison of algorithms T1 and T4 for unconstrained version of problem (R) with MS $= 10^5$. Time is measured in cpu seconds.

Products	Revenue (T1)	Time (T1)	Revenue (T4)	Time (T4)
25	1.0924	0.068004	1.0924	0.04
100	2.37603	0.148009	2.37603	0.068001
400	3.69198	0.49603	3.69198	0.236014
900	4.38097	1.17607	4.38097	0.424024
1600	5.23188	2.14013	5.23188	0.976055

Table 3.2: Run-time comparison of algorithms T1 and T4 for unconstrained version of problem (R) with MS $= 1$. Time is measured in cpu seconds.

We next compare the optimal results and computation times for algorithms **T1** - **T4** for **constrained version** of the optimization problem (R). Tables 3.3, 3.5 and 3.7 compare the computational performance while Tables 3.4, 3.6 and 3.8 compare the optimal revenues generated by the four algorithms (**T1** - **T4**). For the data instances for which all four algorithms terminated within a reasonable amount of time, the maximum difference between the optimal revenues generated by the different algorithms is 4.75% (for the 25 product data instance with MS = 10^5) and on average the optimal solutions are within 0.2% of each other. For data instances with a large number of products, algorithm **T3** is computationally the most efficient (on average 1.5 times faster than **T4**). For small to medium sized data instances the performance varies with the solution method **T4** being more efficient for some while the solution method **T2** is more efficient for others. An important observation that can be made from the results tabulated in Tables 3.1,3.2, 3.4, 3.6, 3.8, 3.3, 3.5 and 3.7 is that algorithm **T4** encounters problems when the θs start getting smaller. This situation will most likely occur whenever the number of products is large or the customer segment to be served is large when compared to the production/inventory capacity. The most likely reason for this is the fact that the objective in this method utilizes the log function which is not a well behaved function for small values. So for the unconstrained instances of the problem (Tables 3.1 and 3.2) when ordinarily none of the θs can be expected to get very small, solution method **T4** is computationally most efficient for all values of MS. For the constrained instances of the problem, there is a gradual degradation in performance of algorithm **T4** as one moves from data instances with $MS = 1$ to $MS = 10^5$ and from small size product instances to large size product instances.

However small θs may not be the only reason for the relatively poor performance of algorithm **T4** for data instances with large number of products or large value of MS. This is evident from the results included in Table 3.9. The data instances used to generate the results in the table are the same as those utilized to generate Table 3.3 with the only exception being the products whose optimal θs were of the smallest order of magnitude (Eg., for the data instance with 400 products these were of order 10^{-7}) were removed from the corresponding data files. The results included in the table show that the relative performance of the KKT based algorithms and the log algorithm remains unchanged i.e. for 'small' data instances algorithm **T4** still performs better while for larger data instances the KKT based algorithms perform better.

Products	T1	T2	T3	T4
25	0.168005	0.248014	0.160009	0.024001
100	0.66404	1.83611	1.27608	0.300016
400	–	34.294102	22.9454	16.677054
900	–	187.040006	127.212	197.74427
1600	–	537.662008	428.271	1939.4889
2025	–	984.08101	691.58301	4889.9756

Table 3.3: Run-time comparison of algorithms T1-T4 with MS = 1. Time is measured in cpu seconds. '–' indicates that the simulation took too long to run.

Products	T1	T2	T3	T4
25	1.20826	1.20827	1.20827	1.20827
100	2.303063	2.30365	2.30365	2.30365
400	–	3.76279	3.76279	3.76279
900	–	4.66478	4.66478	4.66478
1600	–	5.13357	5.13357	5.13357
2025	–	5.38896	5.38896	5.38896

Table 3.4: Optimal Revenue computed by algorithms T1-T4 with MS = 1. '–' indicates that the simulation took too long to run.

Products	T1	T2	T3	T4
25	0.908055	0.164009	0.196011	0.096007
100	21.16928	1.708101	2.144136	1.420084
400	1023.3282	71.8848	68.0123	158.97008
900	–	422.275	624.987	2218.91834*
1225	–	1292.5	1322.518	2944.8947
1600	–	12575.478	2691.21	17496.4336*

Table 3.5: Run-time comparison of algorithms T1-T4 with MS = 10^2. Time is measured in cpu seconds. '–' indicates that the simulation took too long to run. '$*$' indicates that the simulation did not generate the optimal solution.

Products	T1	T2	T3	T4
25	2.19094	2.19904	2.19034	2.19094
100	16.2074	16.2795	16.1889	16.2074
400	71.9604	48.327	71.8227	71.9604
900	–	162.2	162.184	148.137*
1225	–	221.561	221.976	221.981
1600	–	293.9	294.101	275.599*

Table 3.6: Optimal Revenue computed by algorithms T1-T4 with MS = 10^2. '$-$' indicates that the simulation took too long to run. '$*$' indicates that the simulation did not generate optimal results.

Products	T1	T2	T3	T4
25	0.752046	0.19201	0.220011	0.224013
100	17.51706	3.114201	3.56422	0.068001
400	–	131.5282	137.5124	177.71507
900	–	1556.619	1560.447	3051.95235
1600	–	16141.454	9836.577	$*$
1764	–	32477.5	11286.774	$*$

Table 3.7: Run-time comparison of algorithms T1-T4 with MS = 10^5. Time is measured in cpu seconds. '$-$' indicates that the simulation took too long to run. '$*$' indicates that the simulation encountered problems.

We now include a brief discussion on the differences in computational performance of solution methods **T2** & **T3** and further improvements that can be made to the KKT based algorithms in order to improve their computational performance. Both algorithms are quite similar with method **T3** being a slightly less refined version of **T2** (T2 requires additional steps). However an important run time observation is that **optimization problem ($O1$) is much harder to solve for small values of C as compared to large values** (eg., solving the problem (O1) at C = 0.1 will take longer than solving it at C = 5). Solution method **T2** being more refined delays solving problem ($O1$) until very late in the algorithm and as a result, requires more time to generate the optimal solution to ($O1$) as compared to **T3** which in general ends up solving problem ($O1$) at a slightly higher value of C. This explains the fact that even though both solution methods are very similar, for very large data instances solution method **T3** performs marginally better than **T2**.

Products	T1	T2	T3	T4
25	44.5122	42.3945	43.7656	44.5122
100	2115.26	2176.43	2110.52	2115.26
400	–	9937.16	9906.58	9922.5
900	–	22927.2	22860.9	22899.5
1600	–	42313.7	42249.6	*
1764	–	46502.5	46410.5	*

Table 3.8: Optimal Revenue computed by algorithms T1-T4 with MS $= 10^5$. '–' indicates that the simulation took too long to run. '*' indicates that the simulation encountered problems.

Products	T2	T3	T4
16	0.088	0.108	0.04
81	1.72	1.372	0.208
361	22.185	17.6691	13.84
841	271.305	362.286	352.106

Table 3.9: Run-time comparison of algorithms T2, T3 and T4 with MS $= 1$, after removing products with small value of optimal θ. Time is measured in cpu seconds.

One natural way to improve the computational efficiency of the KKT based algorithms is to come up with better upper bounds for the optimal profit of optimization problem (R). To explore this idea we use the unconstrained optimal profit (r^*) of the data instances as an upper bound instead of the default upper bound $(= \frac{1}{e}.\sum i \in N(1/b_i))$. The results are tabulated in Table 3.10. **T5a** is the same as algorithm **T2** except that we start from the unconstrained optimal solution of the problem as an upper bound. In **T5b** the time required to find the unconstrained optimal solution is included within the total computation time whereas in **T5a** this time is excluded. In **T5c** we start from a random point ω which satisfies the following inequality, $r^* \geq \omega \geq \frac{1}{e}.\sum i \in N(1/b_i)$.

As expected for most instances algorithm **T5a** takes the least amount of time while **T2** takes the longest with some exceptions. The exceptions arise mainly because of the run time observation that has already been made i.e., the smaller the value of C, the harder it is to solve the problem $(O1)$. Generally starting at a lower value of the upper bound, one can expect to cut down on the time the algorithm takes to arrive at the point where the solution to problem $(O1)$ needs to be computed. For the problem instances when algorithm **T5a** takes longer to run than the other algorithms, the value of C at which the solution to problem $(O1)$ is computed is lower, hence **T4a** requires a lot longer to solve problem $(O1)$ as compared to other algorithms (see Table 3.10). To further clarify this point assume that we start initially from '1.5' as upper bound for one method (say **T5a**) and '2' as the upper bound for some other method (say **T2**). As the lower bound for both the methods is same and fixed at $L = 0$, the value of C will be '0.75' for the first method and '1' for the second method. Further let's assume that based on the respective algorithms we need to compute the solution to optimization problem $(O1)$ at this value of C. We can expect the method utilizing $C = 0.75$ to take much longer than the method utilizing $C = 1$, resulting in higher total run time for the first method as compared to the second method.

Based on the discussion in the preceeding paragraph for speeding up the KKT based solution procedures, in our opinion it appears that while finding tighter upper bound may help, what might help more these algorithms to become significantly more efficient, is an easier and faster way to solve optimization problem $(O1)$.

Products	T2	T5a	T5b	T5c
25	0.19201	0.096005	0.17201	0.168009
100	3.114201	2.304143	3.08819	3.1722
400	131.5282	320.19638	320.5204	130.2282
900	1556.619	1256.25371	1257.7218	1526.431
1600	16141.454	16409.674	16654.822	20286.686

Table 3.10: Run-time comparison of algorithms T2,T4 and T5 with MS = 10^5. Time is measured in cpu seconds.

3.6 Conclusions

In this chapter we introduced four algorithms for solving the nonconvex profit maximization problem based on the Multinomial Logit Demand (MNL) model under capacity constraints. All four algorithms converge although they do differ in their computational efficiency. The algorithms based on the KKT conditions were shown to be the fastest for data instances with a large number of products and a large number of customers in the segment. The algorithm based on a log reformulation was found to be the fastest for the unconstrained version of problem (R). An interesting avenue for further work would be to test the four algorithms developed in this chapter on real data instances procured either from supermarkets or other similar establishments where the MNL can be utilized to model the customer demand. Another avenue for further work can be to devise more efficient ways to solve optimization problem $(O1)$ which will help to improve the computational efficiency of the algorithms based on the KKT conditions. Also as newton based algorithms for solving system of equations are faster than binary search based algorithms, devising a newton based scheme for solving Equation (3.19) can potentially speed up the algorithms based on the KKT optimality conditions for both the constrained and unconstrained versions of problem (R).

Chapter 4

Joint Pricing and Inventory Control Problem under the Multinomial Demand

This chapter describes the joint pricing and inventory control problem (JPICP) with the multinomial logit (MNL) demand model. The first section describes the JPICP model in the MNL setting. This is followed by algorithm description based on a convex reformulation of the demand model proposed by Keller *et al.* [13] and introduced in Section 3.4 of the previous chapter. Runtime performance of the developed algorithm is discussed in Section 4.3. Finally Section 4.4 brings together the important ideas of this chapter and addresses both the advantages and disadvantages of the proposed algorithm.

4.1 JPICP in a MNL Setting

In this section we introduce two models of the JPICP based on the MNL demand model. In both models the cost part of the objective is piecewise convex and can be represented utilizing the a linear transportation problem as in Chapter 2. The first model (**M1**) of the problem is based on a nonconvex objective function. Notice that for this model the feasible region is a convex set. The variables and parameters have the same meaning as those described in Chapters 2 and 3 (where no inventory constraints were present). This model can be solved directly by utilizing the nonlinear LOQO solver.

$$M1 : max_{\overline{x},x} \quad MS.\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{a_{it}\overline{x}_{it}e^{(-\overline{x}_{it})}}{1+\sum_{j=1}^{N}\sum_{k=1}^{T}e^{-\overline{x}_{jk}}} - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{s=1}^{T}b_{it}^{s}x_{it}^{s} \tag{4.1}$$

$$\sum_{s=1}^{t}x_{it}^{s} \geq MS\frac{a_{it}\overline{x}_{it}e^{(-\overline{x}_{it})}}{1+\sum_{j=1}^{N}\sum_{k=1}^{T}e^{-\overline{x}_{jk}}}, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \tag{4.2}$$

$$\sum_{i,g:g\geq s} x_{ig}^{s} \leq C_s \quad \forall s = 1,2,..,T \tag{4.3}$$

$$\overline{x}_{it} \geq 0, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \tag{4.4}$$

$$x_{it}^{s} \geq 0, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \quad \forall s = 1,2,..,T \tag{4.5}$$

The new decision variables $\overline{x}_{it}$ are defined by the equality, $\overline{x}_{it} = b_{it}p_{it}$,where b_{it} and p_{it} have the same meaning associated with them as in Chapter 3. The constraints described in relation (4.2) ensure that all demand is met while those represented by inequality (4.3) ensure that the joint production capacity for each period is not violated.

The second model of problem (**M2**) based on the convex reformulation of the MNL demand model (see [13]) is presented next.

$$M2 : max_{\theta,\theta_o,x} \quad -MS.\sum_{i=1}^{N}\sum_{t=1}^{T}a_i\theta_{it}log(\frac{\theta_{it}}{\theta_o}) - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{s=1}^{T}b_{it}^{s}x_{it}^{s} \tag{4.6}$$

$$\sum_{i,g:g\geq s} x_{ig}^{s} \leq C_s \quad \forall s = 1,2,..,T \tag{4.7}$$

$$\sum_{s=1}^{t} x_{it}^{s} \geq MS.\theta_{it}, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \tag{4.8}$$

$$\sum_{i=1}^{N}\sum_{t=1}^{T} \theta_{it} + \theta_o = 1 \tag{4.9}$$

$$\theta_{it} \geq 0, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \tag{4.10}$$

$$\theta_o \geq 0 \tag{4.11}$$

$$x_{it}^{s} \geq 0, \quad \forall i = 1,2,..,N \quad \forall t = 1,2,..,T \quad \forall s = 1,2,..,T \tag{4.12}$$

4.2 An Iterative Algorithm based on a Log Reformulation of the MNL model

As the objective in model (M2) is concave (for a proof see [13]), this model can be utilized for an iterative algorithm similar to **ST3** discussed in Chapter 2. At each iteration, as in algorithm **ST3** (see Chapter 2), we consider a local quadratic approximation of the revenue part and a linear approximation of the cost part, in the objective function at the current iterate. The gradient of the revenue function is given by Equation (4.13). The diagonal approximation of the Hessian matrix is given by Equation (4.14).

$$\nabla r(\theta)_{it} = -MS.a_{it}(1+ln(\theta_{it}/\theta_o))-MS.\sum_{j=1}^{N}\sum_{k=1}^{T} a_{jk}\frac{\theta jk}{\theta_o}, \quad \forall i = 1,2..,N \quad \forall t = 1,2,..,T \tag{4.13}$$

$$diagH_{it} = -MS.a_{it}(1+\theta_{it}/\theta_o)(1/\theta_{it}) - (MS.a_{it}/\theta_o) - MS.\sum_{j=1}^{N}\sum_{k=1}^{T} a_{jk}(\theta_{jk}/(\theta_o^2)) \tag{4.14}$$

The dual problem utilized for computing the gradient of the cost function in the objective of model (M2) is given by optimization problem (D3).

$$D3: max_{y,z} \sum_{i,t} \overline{\theta}_{it} z_{it} - \sum_{s} C_s y_s \tag{4.15}$$

subject to:

$$z_{it} - y_s \leq b_{it}^s, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, .., T \quad \forall s = 1, 2, ..., T \tag{4.16}$$

$$y_s \geq 0, \quad \forall s = 1, 2, ..., T \tag{4.17}$$

$$z_{it} \geq 0, \quad \forall i = 1, 2, ..., N \quad \forall t = 1, 2, .., T \tag{4.18}$$

The reformulated optimization model approximated at a point $\overline{\theta}_{it}$ is given by the optimization problem (P5). In order for this iterative scheme to converge to the optimal solution all assumptions discussed in Section 2.3 must hold.

$$P5: max_{\theta} \ \ r(\overline{\theta}) + \nabla r(\overline{\theta})^T(\theta - \overline{\theta}) + \frac{1}{2}(\theta - \overline{\theta})^T.diagH(\overline{\theta})(\theta - \overline{\theta}) - z^T(\theta - \overline{\theta}) \tag{4.19}$$

$$MS.\sum_{i=1}^{N}\sum_{t=1}^{g} \theta_{it} \leq \sum_{s=1}^{g} C_s, \quad \forall g = 1, 2, ..., T \tag{4.20}$$

$$\sum_{i=1}^{N}\sum_{t=1}^{T} \theta_o = 1, \tag{4.21}$$

$$\theta_{it} \geq 0, \quad \forall i = 1, 2, .., N \quad \forall t = 1, 2, .., T \tag{4.22}$$

$$\theta_o \geq 0 \tag{4.23}$$

Algorithm **ST5** for utilizing the above optimization models (D3 and P5) to solve the MNL based JPICP is outlined next.

Step 1:

Start with a feasible point $\overline{\theta}$ in the feasible region of the problem ($P5$). Compute $\nabla r(\overline{\theta})$, diag$H(\overline{\theta})$ at this point.

Step 2:

Solve the optimization problem (D3).

Step 3:

Consider a quadratic approximation of r(θ) and a linear approximation of the cost function at the current iterate values based on optimal decision variables z_{it} of (D3).

Step 4:

Solve the optimization problem ($P5$) using a QP solver such as CPLEX.

Step 5:

Check the difference between the optimal decision variables θ and $\overline{\theta}$. If this difference is less than a fixed tolerence (= 0.01), stop, otherwise replace $\overline{\theta}$ with the currently optimal variables θ, recompute $\nabla r(\overline{\theta})$, diag$H(\overline{\theta})$ and go to step 2.

4.3 Computational Performance of the Algorithm

In this section we will compare the computational performance and accuracy of the models **M1** and **ST5** presented in the last section. The instances included in this section were run on a shared computer with two Intel Xeon 1500 MHz processor and combined cache of 512 KB (256 KB each). Model (M1) was solved utilizing the LOQO solver and CPLEX was used to solve the optimization problems in **ST5**. The total time was calculated using the same procedure as the instances in the preceeding two chapters. The tolerance criteria utilized for **ST5** was 0.01.

The data instances and model parameters were generated using MATLAB. The production and inventory holding cost/period were generated from a uniform distribution between 0 and 1. The market segment (MS) variable was set to 10^5. The joint production capacity was generated from a normal distribution having mean N x T (where N is the total number of products and T is the total number of time periods) and standard deviation of 10.

Table 4.1 contains information on the computational efficiency and accuracy of **ST5** and **M1**. The optimal results are not too far off (within 4% of each other) implying both algorithms worked well for the data instances utilized to generate the table. However **ST5** is more efficient than **M1**. This is in agreement with the computational behavior of the LOQO based MNL revenue model studied in Chapter 3. However one thing to be kept in mind when comparing the results is that the tolerance criterion utilized for **ST5** is much higher than 10^{-8} (i.e. 10^{-2}) utilized for all the other algorithms presented in this thesis. In particular the update rule for $\overline{\theta}$ in step 5 needs to be improved. For the current version of the algorithm this rule is given $\overline{\theta}_{new} = \overline{\theta}_{old} + \alpha(\theta - \overline{\theta})$ with $\alpha = 1$. However for this particular model iterative (newton based) procedures with $\alpha = 1$ do not always converge to the optimal solution. Another problem with the current version of **ST5** is oscillation and some checks based on the relative improvement of the objective and change in the optimal decision variables have been utilized to make sure that the algorithm terminates after a finite number of steps.

(Products,Periods)	Profit (M1)	Total Time (cpu sec)	Profit (ST5)	Total Time (cpu sec)
5,5	19718.1	0.428025	18905.6	0.060002
10,10	218702	50.835205	218329	0.156009
15,15	602666	575.200026	602509	0.576035
20,20	44177500	5682.82	44133300	0.948058
25,25	-	-	309410000	7.27245

Table 4.1: Comparison of optimal profit and total time required by M1 and ST5. '-' indicates that the data instance took very long time to run

4.4 Conclusions

In this chapter we proposed a computationally efficient and tractable algorithm **ST5** for solving the JPICP based on a MNL demand model. The proposed algorithm is build around observations that the cost part of the objective can be minimized separately by solving a linear transportation problem and the actual nonconvex MNL based revenue function can be replaced by its concave equivalent. The proposed algorithm though efficient appears to have some convergence problems. An interesting avenue for further work will be to come up with appropiate step size rules for this algorithm that will resolve the convergence problem of the proposed scheme and to then test it comprehensively with different data instances similar to the ones used to test algorithms in Chapter 3.

Appendix A

Definitions

Concave Function: A function for which all points satisfy the inequality, $f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y), \forall x, y \in \Re^n$ and $0 \leq \lambda \leq 1$, is defined as a concave function. For a function to be concave the negative of its Hessian matrix has to be at least positive semi-definite. For a strictly concave function the negative of its Hessian matrix has to be strictly positive definite.

Cost: The total amount of capital required in manufacturing and storing a particular product.

Hessian Matrix: It is the matrix of partial second derivatives of a function f. A sample $HessianMatrix$ for a function of two variables, f(x,y), is given by :
$H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y}, & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$.

Lipschitz Continuous: It is condition on smoothness of a continuous function (f:$\Re^m \rightarrow \Re^n$) which is given by the following relationship:
$L||x - x^*|| \geq ||f(x) - f(x^*)||, \forall x, \quad \forall L \geq 0.$
x^* is the optimal solution of the problem under consideration, x is the current iterate value and L is the Lipschitz continuity constant.

Local Quadratic Approximation: It is the Taylor approximation of a function, $f(x)$, at a point $\overline{x}$ truncated after the first three terms.

M−matrix B: A matrix with positive diagonals, non positive off-diagonals, which is positive semi-definite.

Piecewise Convex Function: A function which is defined by
$f(x) = max_i \ \ m_i x + b_i$ is called a piecewise convex function (see also [16]).

Price Sensitivity: A coefficient that reflects how the price of a particular product is affect when the price of another comparable product changes.

Positive Definite Matrix: A matrix whose all eigenvalues are strictly greater than zero, is called a positive definite matrix.

Positive Semi-Definite Matrix: A matrix whose all eigenvalues are greater than or equal to zero, is called a positive semi-definite matrix.

Profit: Cost deducted from the revenue yields profit.

Revenue: Revenue is the product of total number of units sold and price/unit.

Strictly Diagonally Dominant Matrix: If the sum of the absolute values of all the off diagonal elements in all rows of a square matrix are strictly less than the absolute value of the corresponding diagonal terms then the matrix under consideration is called a strictly diagonally dominant matrix. Note a strict diagonal dominant matrix is also positive definite but the inverse is not always true.

Subgradient: For a piecewise convex function, $f(x)$, p is a subgradient of the function at $\overline{x}$, if $\forall x$ in its domain, the following relationship is satisfied:
$f(x) - f(\overline{x}) \geq g^T(x - \overline{x})$.

Substitutes: Products that can be used readily in place of another product. Eg. Similar toothpastes marketed by different brands.

Bibliography

[1] Kent B. Monroe. *Pricing making Profitable Decisions.* McGraw-Hill/Irwin, 2003.

[2] T.M. Whitin. Inventory Control and Price Theory. *Management Science*, 2(1):61-68, 1955.

[3] D. Pekelman. Simultaneous price-production decisions. *Operations Research*, 22(4):788794, 1974.

[4] Gerald L. Thompson, Suresh P. Sethi and Jinn-Tsair Teng. Strong planning and forecast horizons for a model with simultaneous price and production decisions. *European Journal of Operational Research*, 16(3):378-388, 1984.

[5] G. Feichtinger and R.F. Hartl. Optimal pricing and production in an inventory model. *European Journal of Operational Research*, 19(1):45-56, 1985.

[6] C. Gaimon. Simultaneous and dynamic price, production, inventory and capacity decisions. *European Journal of Operational Research*, 35(3):426-441, 1988.

[7] Stefen Jurgensen, Peter M. Kort and Georges Zaccour. Production, inventory, and pricing under cost and demand learning effects. *European Journal of Operational Research*, 117(2):382-395, 1999.

[8] Stephen M. Gilbert. Coordination of pricing and multi-period production for constant priced goods. *European Journal of Operations Research*, 114:330-337, 1999.

[9] Elodie Adida and Georgia Perakis. A nonlinear continuous time optimal control model of dynamic pricing and inventory control with no backorders. *Naval Research Logistics*, 54(7):1602-1616, 2007.

[10] Ward Hanson and Kipp Martin. Optimizing Multinomial Logit Profit Functions. *Management Science*, 42(7):992-1003, 1996.

[11] Guillermo Gallego and Catalina Stefanescu. Upgrades, Upsells and Pricing in Revenue Management. *Working Paper.*

[12] A. Nagurney. *Network Economics.* Kluwer Academic Publ., 1999.

[13] Philipp Keller, Restef Levi and Georgia Perakis. Pricing in response to Multinomial Logit demand. *Working Paper.*

[14] Retsef Levi and Georgia Perakis. KKT based algorithms for optimizing Multinomial Logit profit function. *Working Paper.*

[15] Tingting Rao. LP-based Subgradient Algorithm for Joint Pricing and Inventory Control Problems. *MIT S.M. Thesis*, 2008.

[16] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[17] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2003.

[18] Line Searches and Newton's Method. *15.093 Optimization Methods Lecture 19.* MIT, Fall 2008.

[19] K. T. Talluri and G. V. Ryzin. *The Theory and Practice of Revenue Management.* Springer, 2004.

[20] Duncan R. Luce. *Individual Choice Behavior.* Wiley, 1959.

[21] Daniel L. McFadden. Economic Choices. *Nobel Memorial Prize Lecture*, 2000.

[22] Peter M. Guadagni and John D. C. Little. A Logit Model of Brand Choice Calibrated on Scanner Data. *Marketing Science*, 27(1):29-48, 1983.

[23] Moshe Ben-Akiva and Steven Lerman. *Discrete Choice Analysis Theory and Application to Travel Demand.* MIT Press, 1985.

[24] James Berkovec. New Car Sales and Used Car Stocks: A Model of the Automobile Market. *The RAND Journal of Economics*, 16(2):195-214, 1985.

[25] K. Train, D. McFadden, and M. Ben-Akiva. The Demand for Local Telephone Service: A Fully Discrete Model of Residential Calling Patterns and Service Choices. *RAND Journal of Economics*, 18(1):109-123, 1987.

[26] S.P. Anderson and A. de Palma. The Logit as a Model of Product Differentiation. *Oxford Economic Papers*, 44:51-67, 1992.

[27] G. Debreu. A Review of Individual Choice Behavior: A Theoretical Analysis. *American Economic Review*, 50(December):186-188, 1960.

[28] Kenneth Train. *Discrete Choice Methods with Simulation.* Cambridge University Press, 2003.