Maximum Utility Product Pricing Models and Algorithms Based on Reservation Prices*

R. Shioda L. Tunçel T. G. J. Myklebust

April 15, 2007

Abstract

We consider a revenue management model for pricing a product line with several customer segments under the assumption that customers' product choices are determined entirely by their reservation prices. We highlight key mathematical properties of the maximum utility model and formulate it as a mixed-integer programming problem, design heuristics and valid cuts. We further present extensions of the models to deal with various practical issues arising in applications. Our computational experiments with real data from the tourism sector as well as with the randomly generated data show the effectiveness of our approach.

C&O Research Report: CORR 2007-08
Department of Combinatorics and Optimization
University of Waterloo
Waterloo, ON, Canada

*Research supported in part by Discovery Grants from NSERC, a research grant from Air Canada Vacations and by a Collaborative Research and Development Grant from NSERC.

1 Introduction

Suppose a company has m different product lines and market analysis tells them that there are n distinct customer segments, where customers of a given segment have the "same" purchasing behavior. A key revenue management problem is to determine optimal prices for each product to maximize total revenue, given the customer choice behavior. There are multitudes of models for customer choice behavior [13], but this paper focuses solely on those based on reservation prices.

Let R_{ij} denote the *reservation price* of Segment i for Product j, $i = 1, \ldots, n$, $j = 1, \ldots, m$, which reflects how much customers of Segment i are *willing* and *able* to spend on Product j. If the price of product j is set to $\$\pi_j$, then the *utility* or *surplus* (we will use these terms interchangeably throughout the paper) of Segment i for Product j is the difference between the reservation price and the price, i.e., $R_{ij} - \pi_j$. If there are competitors in the market, then we would need to consider the utility of each segment for the competitors' product as well. Let CS_i denote the maximum surplus of Segment i across all competitor products. We will assume that R_{ij} and CS_i are nonnegative for all i and j without loss of generality. Finally, we assume that reservation prices are the same for every customer in a given segment and each segment pays the same price for each product. Customer choice models based on reservation prices assume that customer purchasing behavior can be fully determined by their reservation price and the price of products.

Even in a reservation price framework, there are several different models for customer choice behavior in the literature [2, 3, 8, 9]. It is often assumed that a segment will only consider purchasing a product with positive utility, but there are ambiguities regarding choice between multiple products with positive utility. This paper is largely inspired by [2], where they study the *maximum utility* or *envy free pricing* model of customer choice behavior. In this model, we assume that a customer segment will purchase the product with the largest surplus. In [6], the authors present a linear mixed-integer programming formulation for bundle pricing using a similar framework as [2]. It is shown that the maximum utility problem is $\mathcal{NP}$-hard [3] as well as $\mathcal{APX}$-hard [5].

In this paper, we present a mixed-integer linear programming formulation for the maximum utility model (similar to [6]), offer further mathematical insight to the model, expand on the heuristics proposed in [2], present several effective mixed-integer cuts, present a stable-set formulation, and illustrate computational results using CPLEX and our heuristic. The purpose of this paper is not to argue that this customer choice model is better than others, nor claim that one should use pricing models based on reservation prices. Our goal is to present mathematical programming and algorithmic approaches to solving these problems efficiently.

The structure of the paper is as follows: Section 2 describes the maximum utility pricing model, presents several mixed-integer optimization formulations and illustrates special mathematical properties of the problem. Section 3 presents heuristics for finding "good" feasible solutions and Section 4 illustrates several valid inequalities for the mixed-integer programming

problem. Section 5 extends the formulation and heuristic algorithms to consider capacity constraints for each product and Section 6 presents further extensions such as a stable set formulation and a semidefinite relaxation of the model. Section 7 illustrates the results of the computational experiments of our heuristic and CPLEX on randomly generated and real data. Finally, Section 8 summarizes our findings and describes our current and future research.

2 Maximum Utility Model

In the *maximum utility* or the *envy-free pricing* model, the assumption is that the customer will choose the product that maximizes his or her utility, given the price of all the products. Thus, if π_j is the price of Product j, $j = 1, \ldots, m$, then Segment i will buy Product j only if:

$$j = \underset{k=1,\ldots,m}{\operatorname{argmax}} \{R_{ik} - \pi_k\}$$

and

$$R_j - \pi_j \geq CS_i.$$

We further make the following assumptions:

(A.1) *Unit Demand*: Each customer segment buys at most one product.

(A.2) *Non-Differentiated Pricing*: Every customer segment pays the same price for each product.

(A.3) *Static Competition*: Competitors do not react to our prices, thus CS_i is a constant and not a function of π_j.

(A.4) *Tie-Breaking*: To eliminate the case of ties in the maximum surplus product and to make the maximum utility model more robust under small changes to R_{ij}, we further assume that the surplus of the product chosen by customer Segment i must be larger than the surplus of any other product by at least some pre-determined amount $\delta_i > 0$. That is, Segment i buys Product j if and only if

$$R_{ij} - \pi_j \geq R_{ik} - \pi_k + \delta_i, \qquad \forall k \neq j \tag{1}$$

and

$$R_{ij} - \pi_j \geq CS_i + \delta_i. \tag{2}$$

We call δ_i the *utility tolerance* for customer Segment i.

The first three assumptions are quite common in many revenue management models. However, the last assumption (A.4) seems to be uncommon in the literature. We felt that such an assumption was needed to be more confident about the applicability of the mathematical model in the real world. Without the usage of such positive δ_i, any time our company's price ties

the price of another company (in terms of the customers' utility) as the best price, we would be declaring that our company wins *all* of these customers (in the corresponding segment) and collects the associated revenue. Another situation is when two or more of our own products are in a *tie* (in terms of the underlying utilities) for the best price for a customer segment. Without the usage of δ_i, an optimal solution of the mathematical model will assign all of the customers in that segment to the product with the highest price. Some revenue management models assume this as a tie-breaking rule, e.g., the so-called MAX-PRICE model assumes that the customer will choose the product with the highest price in case of a tie in the utilities. While this may be true for some customers, this seems unrealistic to us as a sweeping assumption for many applications.

For large values of δ_i, our models may be too conservative; however, because of the uncertainty in the data (e.g., reservation prices and competitor surplus), it would be wise to be more conservative in modeling the revenue management problem. Thus, the δ_i parameter builds in robustness to our model by protecting the solution against data perturbations. For example, suppose Segment i buys Product j in our solution (thus the prices satisfy Eq. (1)), but the R_{ij} decreases by ϵ or R_{ik} increases by ϵ for some other Product k. As long as $\epsilon \leq \delta_i$, Product j will still be the maximum surplus product for Segment i.

A more ambitious modification of our assumptions would assign different δ_i's to better represent the preferences of customer Segment i among the various products available. Namely, we would stipulate that for Product j to be chosen by customer Segment i, the surplus of Product j must be at least $\delta_{ijk} > 0$ larger than that of product k, for every $k \in \{0, 1, \ldots, m\} \setminus \{j\}$, where index 0 represents the competitors' product with the largest surplus. Note that δ_{ijk} is not necessarily the same as δ_{ikj}. In this paper, we only use δ_i.

A consequence of (A.4) is that according to the resulting mathematical model, some customer segments might not buy any of our products even if one (or more) of our products has strictly positive and larger surplus compared to all other products (because none of our products beats every contender by at least δ_i). One alternative to Eq. (1) that avoids this is

$$\begin{aligned} R_{ij} - \pi_j &\geq R_{ik} - \pi_k + \delta_i, \quad \forall k, j, R_{ij} > R_{ik}, \\ R_{ij} - \pi_j + \delta_i &\geq R_{ik} - \pi_k, \quad \forall k, j, R_{ij} < R_{ik}, \\ R_{ij} - \pi_j &\geq R_{ik} - \pi_k, \qquad \forall k \neq j, R_{ij} = R_{ik}. \end{aligned} \tag{3}$$

The above condition gives preference to products with smaller reservation prices, so that the customer will buy the cheaper product in case of ties in the maximum surplus. Since the choice of (1) or (3) does not impact the complexity of our mathematical model, we will use Eq.(1) for the remainder of the paper.

2.1 Basic Optimization Models

We now introduce our mathematical programming formulation of the maximum utility model. Let our decision variables be as follows:

$$\theta_{ij} := \begin{cases} 1, & \text{if Customer Segment } i \text{ buys product } j, \\ 0, & \text{otherwise,} \end{cases}$$

$$\pi_j := \text{Price of Product } j.$$

Eqs.(1) and (2) can then be modeled as:

$$(R_{ij} - \pi_j)\theta_{ij} \geq (R_{ik} + \delta_i)\theta_{ij} - \pi_k, \qquad \forall k \neq j,$$

and

$$(R_{ij} - \pi_j)\theta_{ij} \geq (CS_i + \delta_i)\theta_{ij}, \quad \forall j,$$

respectively.

Incorporating the unit demand assumption, the problem can be modeled as the following nonlinear mixed-integer programming problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} N_i \pi_j \theta_{ij}, && (4) \\ \text{s.t.} \quad & (R_{ij} - \pi_j)\theta_{ij} \geq (R_{ik} + \delta_i)\theta_{ij} - \pi_k, && \forall j, \forall k \neq j, \forall i, \\ & (R_{ij} - \pi_j)\theta_{ij} \geq (CS_i + \delta_i)\theta_{ij}, && \forall j, \forall i, \\ & \textstyle\sum_{j=1}^{m} \theta_{ij} \leq 1, && \forall i, \\ & \theta_{ij} \in \{0, 1\}, && \forall i, j, \\ & \pi_j \geq 0, && \forall j. \end{aligned}$$

To linearize the above model, we introduce a continuous auxiliary variable p_{ij} such that

$$p_{ij} = \begin{cases} \pi_j, & \text{if } \theta_{ij} = 1, \\ 0, & \text{otherwise.} \end{cases}$$

This can be enforced by the constraints

$$\begin{aligned} p_{ij} &\geq 0, \\ p_{ij} &\leq R_{ij}\theta_{ij}, \\ p_{ij} &\leq \pi_j, \\ p_{ij} &\geq \pi_j - \bar{R}_j(1 - \theta_{ij}), \end{aligned} \qquad (5)$$

where

$$\bar{R}_j := \max_i \{R_{ij}\}.$$

The corresponding linearized model is:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} N_i p_{ij}, && (6)\\
\text{s.t.} \quad & R_{ij}\theta_{ij} - p_{ij} \geq (R_{ik} + \delta_i)\theta_{ij} - \pi_k, && \forall j, \forall k \neq j, \forall i,\\
& R_{ij}\theta_{ij} - p_{ij} \geq (CS_i + \delta_i)\theta_{ij}, && \forall j, \forall i,\\
& \textstyle\sum_{j=1}^{m} \theta_{ij} \leq 1, && \forall i,\\
& p_{ij} \leq \pi_j, && \forall i, j,\\
& p_{ij} \geq \pi_j - \bar{R}_j(1 - \theta_{ij}), && \forall i, j,\\
& \theta_{ij} \in \{0, 1\}, && \forall i, j,\\
& \pi_j, p_{ij} \geq 0, && \forall i, j.
\end{aligned}$$

Note that the second set of constraints in (6) implies $p_{ij} \leq (R_{ij} - CS_i - \delta_i)\theta_{ij} \leq R_{ij}\theta_{ij}$, thus the second set of constraints in (5) is not necessary. Moreover, we can preprocess the data such that $R_{ij} \leftarrow \max(0, R_{ij} - CS_i - \delta_i)$ and $CS_i \leftarrow 0$ without changing the above problem. We will work with this preprocessed data for the remainder of the paper.

We may consider aggregating the first set of constraints to reduce the number of constraints. Summing them over all j, $j \neq k$, gives us:

$$\sum_{j \neq k}(R_{ij}\theta_{ij} - p_{ij}) \geq (R_{ik} + \delta_i)\left(\sum_{j \neq k}\theta_{ij}\right) - (m-1)\pi_k, \quad \forall k, \forall i$$

which can be further strengthened to

$$\sum_{j \neq k}(R_{ij}\theta_{ij} - p_{ij}) \geq (R_{ik} + \delta_i)\left(\sum_{j \neq k}\theta_{ij}\right) - \pi_k, \quad \forall k, \forall i. \tag{7}$$

In terms of the LP relaxation, the relative strength of the original versus the aggregated constraint is not clear. Let P_1 be the feasible region of the LP relaxation of (6) and P_2 be that of (6) with its first set of constraints replaced by (7). We find that $P_1 \nsubseteq P_2$ and $P_2 \nsubseteq P_1$ as the following example shows. Suppose $n = 2$, $m = 3$ and the reservation prices are

R_{ij}	Product 1	Product 2	Product 3
Segment 1	800	500	700
Segment 2	600	900	600

and $\delta_1 := \delta_2 := \delta := 1$. To show that $P_1 \nsubseteq P_2$, note that the point $\theta_{12} = 0.5$, $\theta_{13} = 0.5$, $p_{12} = 249.5$, $p_{13} = 349.5$, $\pi_1 = 400$, $\pi_2 = 250$, and $\pi_3 = 350$ (all other variables equal 0) is contained in P_1 but not contained in P_2 since it violates $\sum_{j \neq 1}(R_{1j}\theta_{1j} - p_{1j}) \geq (R_{11} + \delta)(\sum_{j \neq 1}\theta_{1j}) - \pi_1$.

To show that $P_2 \nsubseteq P_1$, note that the point $\theta_{11} = 0.284$, $\theta_{13} = 0.716$, $\theta_{22} = 1$, $p_{11} = 227.5$, $\pi_1 = 227.5$, and $\pi_3 = 199.3$ (all other variables equal 0) is contained in P_2 but not contained in P_1 since it violates $R_{11}\theta_{11} - p_{11} \geq (R_{12} + \delta)\theta_{11} - \pi_2$.

However, our computational experiments showed that the formulation (6) with its first set of constraints replaced by (7) resulted in shorter total computation time in general than the original formulation (6) and the MIP with (7) added to (6). Thus, for the remainder of the paper, we consider the following mixed-integer optimization formulation of the maximum utility model:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} N_i p_{ij}, && (8)\\
\text{s.t.} \quad & \sum_{j\neq k}(R_{ij}\theta_{ij} - p_{ij}) \geq (R_{ik} + \delta_i)(\sum_{j\neq k}\theta_{ij}) - \pi_k, && \forall k, \forall i,\\
& R_{ij}\theta_{ij} - p_{ij} \geq 0, && \forall j, \forall i,\\
& \textstyle\sum_{j=1}^{m}\theta_{ij} \leq 1, && \forall i,\\
& p_{ij} \leq \pi_j, && \forall i, j,\\
& p_{ij} \geq \pi_j - \bar{R}_j(1-\theta_{ij}), && \forall i, j,\\
& \theta_{ij} \in \{0,1\}, && \forall i, j,\\
& \pi_j, p_{ij} \geq 0, && \forall i, j.
\end{aligned}
$$

One may propose another linearized formulation of Model (8) without introducing the p_{ij} variables as follows:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} Rev_i,\\
\text{s.t.} \quad & R_{ij} - \pi_j \geq R_{ik} - \pi_k + \delta_i - M_{ijk}(1-\theta_{ij}), && \forall j, \forall k \neq j, \forall i,\\
& R_{ij}\theta_{ij} + \bar{R}_j(1-\theta_{ij}) - \pi_j \geq 0, && \forall j, \forall i\\
& Rev_i \leq N_i\pi_j + N_i(\max_{k\neq j} R_{ik})(1-\theta_{ij}), && \forall i, j,\\
& Rev_i \leq N_i(\max_k R_{ik})\left(\textstyle\sum_h \theta_{ih}\right), && \forall i, j,\\
& \textstyle\sum_{j=1}^{m}\theta_{ij} \leq 1, && \forall i,\\
& \theta_{ij} \in \{0,1\}, && \forall i, j,\\
& \pi_j, Rev_i \geq 0, && \forall i, j,
\end{aligned}
$$

where

$$M_{ijk} := \bar{R}_j - \min_{\ell}\{R_{\ell k}\} + R_{ik} - R_{ij},$$

and the variable Rev_i corresponds to the revenue earned from Segment i. Although this alternative formulation has fewer variables and its LP relaxation solves faster at each branch-and-bound node in general, its total computation time was consistently longer than that of Model (8). The

main factor was the weakness of its LP relaxation bound due to the many "big-M" constraints, resulting in significantly more branch-and-bound nodes. Thus, for the remainder of this section, we will model the maximum utility model using Formulation (8).

2.2 Price Setting Subproblem

If the optimal values of the binary variables θ_{ij} are given for Model (8), then the problem of finding the corresponding optimal prices π_j breaks down into a shortest path problem. This property was also noted by [2] in their pricing model.

Again, suppose we know what product each customer segment chooses (i.e., θ_{ij} is given). Then, we know that $p_{ij} = \pi_j$ if $\theta_{ij} = 1$ and $p_{ij} = 0$ otherwise. Let C_j be the set of customer segments who buy Product j, i.e.,

$$C_j = \{i : \theta_{ij} = 1\}$$

and let M_j be the total number of customers buying Product j, i.e.,

$$M_j = \sum_{i \in C_j} N_i.$$

Also, let B be the set of products that are bought, i.e.,

$$B = \{j : C_j \neq \emptyset\}.$$

Model (8) simplifies to the following linear programming problem with decision variables π_j:

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} M_j \pi_j, && (9) \\ \text{s.t.} \quad & R_{ij} - \pi_j \geq R_{ik} - \pi_k + \delta_i && \forall j \in B, \forall k \in \{1, \ldots, m\} \setminus \{j\}, \forall i \in C_j, \\ & R_{ij} - \pi_j \geq 0, && \forall j \in B, \forall i \in C_j \\ & \pi_j \geq 0, && \forall i, j, \end{aligned}$$

which further simplifies to

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} M_j \pi_j, && (10) \\ \text{s.t.} \quad & \pi_j - \pi_k \leq \min_{i \in C_j} \{R_{ij} - R_{ik} - \delta_i\} && \forall j \in B, \forall k \in \{1, \ldots, m\} \setminus \{j\}, \\ & \pi_j \leq \min_{i \in C_j} \{R_{ij}\}, && \forall j \in B. \end{aligned}$$

We remove the nonnegativity constraint for π_j since it would be enforced in the optimal solution.

The dual of LP (10) is:

$$\begin{aligned}
\min \quad & \sum_{j,k} r_{jk} x_{jk} + \sum_{j} \gamma_j w_j, \\
\text{s.t.} \quad & \sum_{k \neq j} x_{jk} - \sum_{k \neq j} x_{kj} + w_j = M_j, \quad \forall j \in B, \\
& \textstyle\sum_{k \neq j} x_{kj} = 0, \qquad \forall j \notin B \\
& x_{jk}, w_j \geq 0, \qquad \forall j, k,
\end{aligned}$$

where

$$r_{jk} := \min_{i \in C_j} \{R_{ij} - R_{ik} - \delta_i\}$$

and

$$\gamma_j := \min_{i \in C_j} \{R_{ij}\}.$$

The last set of equality constraints and the nonnegativity constraints imply that

$$x_{kj} = 0 \quad \forall j \notin B,$$

thus, we remove all $x_{kj}, j \notin B$ from the formulation.

By adding a single redundant constraint, the above model becomes:

$$\begin{aligned}
\min \quad & \sum_{j,k} r_{jk} x_{jk} + \sum_{j} \gamma_j w_j, \\
\text{s.t.} \quad & \sum_{k \neq j} x_{jk} - \sum_{k \neq j} x_{kj} + w_j = M_j, \quad \forall j \in B, \\
& -\textstyle\sum_j w_j = -\sum_j M_j, \\
& x_{jk}, w_j \geq 0, \qquad \forall j \in B, k \in B,
\end{aligned} \tag{11}$$

which corresponds to a formulation of $|B|$ shortest path problems in the digraph of Figure 1, where there is a node for each product $j \in B$, an auxiliary node 0, an arc connecting nodes j to k for all $j, k \in B$ with cost r_{jk}, and an arc connecting every node $j \in B$ to node 0 with cost γ_j. Thus, the optimal price of Product j is the length of the shortest path from node j to node 0.

Based on the above network structure, we have

Properties 2.1. The following are some properties of Model (8) and its solutions:

(a) The segment assignments $C_j, \forall j$ correspond to feasible assignments in (8) if and only if the resulting network of (11) has no negative cost cycle.

(b) There exist optimal prices that are integral if all the data are integral.

(c) In every optimal solution, there exists at least one product k such that $\pi_k = \gamma_k = \min_{i \in C_k} \{R_{ik}\}$.

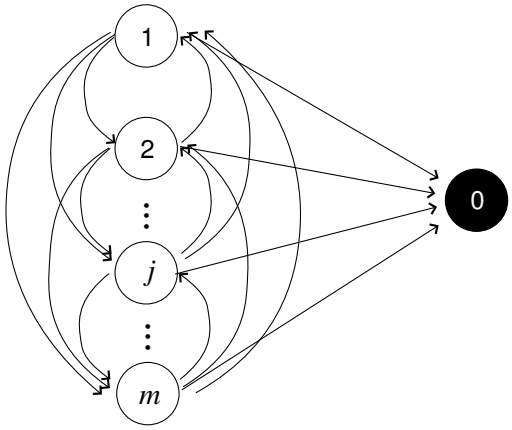

Figure 1: Underlying digraph of (11)

(d) Suppose that $i^* \in C_1$ in an optimal solution and $R_{i^*1} \leq R_{i1}, \forall i$. Then, in that optimal solution, $\pi_1 = R_{i^*1}$.

The proofs are in the appendix.

2.3 Special Cases

Although the maximum utility problem is $\mathcal{APX}$-Hard in general [5], there are some special cases in which the problem can be solved in polynomial time.

$n = 1$ **Case**
The simplest special case is when $n = 1$. Suppose Segment 1 is the sole segment. In the LP relaxation of Model (8), the constraint for θ_{1j} corresponds to a simplex and the objective function is such that it wants to maximize the value of p_{1j} and thus θ_{1j} corresponding to the maximum R_{1j}. Thus, the optimal solution in the LP relaxation is $\theta_{1j^*} = 1$ where $j^* = \arg\max_j R_{1j}$, which is clearly an integer optimal solution.

$n \leq m$ **Case**
Extending the $n = 1$ case where $n \leq m$, the LP relaxation of (8) will result in an integer solution if the maximum reservation product for each customer segment is distinct across all segments and this reservation price is larger than all other reservation prices for that product by at least δ_i. The following is a formal statement of this property:

Lemma 2.1. *For $n \leq m$, the LP relaxation of (8) provides the integer optimal solution if each Segment i ($i = 1, \ldots, n$) can be assigned to a Product j_i where:*

1. $R_{ij_i} \geq R_{ik}$, $\forall k \in \{1, \ldots, m\}$,

2. $j_i \neq j_\ell$, $\forall i, \ell \in \{1, \ldots, n\}, i \neq \ell$,

3. $R_{ij_i} \geq R_{\ell j_i} + \delta_i$, $\forall i, \ell \in \{1, \ldots, n\}, i \neq \ell$.

Proof Sketch. Given the properties in the lemma, it is easy to check that an optimal solution of the LP relaxation is:

$$\pi^*_{j_i} = R_{ij_i}, \quad \theta^*_{ij_i} = 1, \quad \theta^*_{ik} = 0, \quad p^*_{ij_i} = \pi_{j_i}, \quad p^*_{ik} = 0, \quad \forall i \in \{1, \ldots, n\}, \forall k \neq j_i,$$

$$\pi^*_k = \max_i R_{ik}, \quad \forall k \neq j_i, \forall i \in \{1, \ldots, n\}.$$

Clearly, the above solution is integral and thus corresponds to an integer optimal solution of (8) (see the appendix for a complete proof). □

$m = 1$ **Case**
Although the formulation given by (8) will not generally result in an integral LP relaxation when $m = 1$, this special case of the problem can be formulated as an alternative LP problem. From Section 2.2, we know that the solution when $m = 1$ corresponds to a shortest path problem with one arc. Thus, the price is determined by which customer segments purchase the product.

Suppose Product 1 is the sole product on the market. Then the price π_1 is $\min_{i \in C_1} \{R_{i1}\}$. Let $R_{(i)1}$ be the i^{th} order statistic of R_{i1}, $i = 1, \ldots, n$, i.e., $R_{(1)1} \leq R_{(2)1} \leq \cdots \leq R_{(n)1}$. Then the maximum utility problem can be formulated as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \left(\sum_{\ell=i}^{n} N_\ell \right) R_{(i)1} z_i, \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{n} z_i = 1, \\ & z_i \geq 0, \qquad i = 1, \ldots, n, \end{aligned} \tag{12}$$

which clearly gives an integer solution $z_{i^*} = 1$ where $i^* = \arg\max_i (\sum_{\ell=i}^{n} N_\ell) R_{(i)1}$.

$m \leq n$ **Case**
The LP in the $m = 1$ case can lead to an optimal solution for $m > 1$ and $m \leq n$ in very special cases. Suppose C^*_j is the set of customer segments that buy Product j (12). If there are no overlaps in C^*_j, then it corresponds to the optimal assignment for Model (8).

Lemma 2.2. *Suppose C^*_j is the set of segments that buy Product j in the solution of (12) and $C^*_j \bigcap C^*_k = \emptyset, \forall j, k \in \{1, \ldots, m\}, j \neq k$. Then C^*_j is the optimal segment-product assignment of (8) for $\delta_i \leq \min_{j=1,\ldots,m}\{\min_{l \in C^*_j}\{R_{lj}\} - R_{ij}\}$, where this upperbound in strictly positive. Furthermore, the optimal prices are $\pi^*_j = \min_{i \in C^*_j} R_{ij}$.*

The proof is in the appendix.

3 Heuristics

The network substructure in Model (8) motivates heuristics that first find an assignment of customer segments to products (i.e., determine the C_j's) then find the corresponding prices by solving $|B|$ shortest path problems modeled by (11). For the remainder of this section, we will assume that $\delta_i = 0\ \forall i$, unless otherwise stated. A simple heuristic, which we call *Maximum Reservation Price Heuristic* (MaxR), is as follows:

Maximum Reservation Price Heuristic (MaxR)

1: Set $C_j = \{i : j = \arg\max_k\{R_{ik}\}\}, \forall j$.
2: Solve shortest path problems on the network defined by $C_j, \forall j$.

Note that in the above heuristic, $C_j = \emptyset$ if Product j does not have the maximum R_{ij} for any i. If $\delta_i = 0\ \forall i$, the main advantage of MaxR is that it is guaranteed to produce a feasible product-segment assignment for the price setting subproblem (10) since all of the arc costs in the shortest path network (11) will be nonnegative (this may not be the case for $\delta_i > 0$). In addition, this heuristic is guaranteed to produce the optimal solution in certain cases.

Lemma 3.1. *Suppose that $\delta_i = 0$ for all $i \in \{1, 2, \ldots, n\}$ and the conditions in Lemma 2.1 are satisfied. Then, MaxR produces an optimal solution.*

Proof. This follows easily from the proof of Lemma 2.1. □

However, there are clearly several weaknesses to the above heuristic. One such weakness is that it requires every segment to buy a product, which may be sub-optimal. For example, suppose $n = 2$, $m = 2$, $N_1 = 1$, and $N_2 = 1$ with the reservation prices given by the table below:

R_{ij}	Product 1	Product 2
Segment 1	100	99
Segment 2	1	2

The optimal solution is to assign Segment 1 to Product 1 and Segment 2 to buy nothing with the total revenue of 100. However, the heuristic will assign Segment 1 to Product 1 and Segment 2 to Product 2, thus Model (11) will set $\pi_1 = 3$ and $\pi_2 = 2$, resulting in a total revenue of 5. It is apparent from the reservation prices that Segment 2 is not a profitable segment and we should not cater to them.

To counteract the weaknesses of the MaxR heuristic, we can perform sequences of local reassignments and segment deletions (i.e., have a segment not purchase any product) to improve the feasible solution, as done in [2]. Specifically, given a feasible segment-product assignment C_j's and its corresponding optimal spanning tree solution from (11), we reassign a segment that

constrains the price of its product to its parent product in the spanning tree. Such reassignments always guarantee a feasible product-segment assignment and the prices of the products can only increase. We call this extension the *Dobson-Kalish Reassignment Heuristic*:

Dobson-Kalish Reassignment Heuristic

Require: a feasible product-segment assignment and its corresponding optimal spanning tree solution from solving (11) (e.g., via MaxR heuristic)

repeat
  for all products/nodes j where $C_j \neq \emptyset$ **do**
    Suppose arc (j, k) is in the spanning tree solution.
    if $k \neq 0$ **then**
      For every i^* such that $i^* = \arg\min_{i \in C_j}\{R_{ij} - R_{ik}\}$, reassign Segment i^* to product/node k.
    else
      For every i^* such that $i^* = \arg\min_{i \in C_j} R_{ij}$, delete Segment i^* (i.e., Segment i^* buys no products).
    end if
    Resolve the shortest path problem on the new network and record change in the objective value.
    Restore the original network.
  end for
  Perform the reassignment that resulted in the maximum increase in the objective value. Resolve shortest path problems and update the optimal spanning tree.
until no reassignments improves objective value.

Figure 2 illustrates this heuristic on an example with four products.

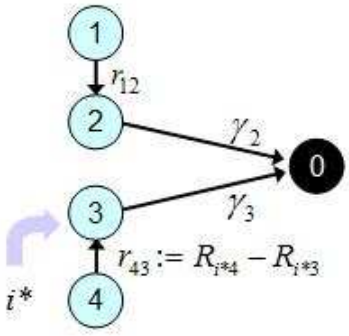

Figure 2: Illustration of the Dobson-Kalish Reassignment Heuristic. The figure shows the optimal spanning tree of a feasible product-segment assignment with four products. Suppose $i^* = \arg\min_{i \in C_4}\{R_{i4} - R_{i3}\}$. Then i^* can be reassigned to Product 3.

Lemma 3.2. *When $\delta_i = 0$ for every i, each reassignment in the Reassignment Heuristic results in a feasible product-segment assignment.*

Proof. We will establish the feasibility of the assignment by producing a feasible solution of (10).

In particular, we will show that the old solution, call it $\bar{\pi}$, stays feasible for (10) under the new assignment. Suppose Segment i^*, $i^* \in C_u$, is being reassigned. If the parent of u is node 0, then i^* is deleted and the corresponding product-segment assignment is clearly feasible. Otherwise, suppose node v is the parent of u and i^* is the unique arg-minimizer of $\min_{i \in C_u}\{R_{iu} - R_{iv}\}$. In the shortest path formulation (11), $x_{uv} > 0$; thus, by complimentary slackness, the constraint $R_{i^*u} - \pi_u \geq R_{i^*v} - \pi_v$ of (9) is active (i.e., both Product u and v offer the maximum utility for Segment i^*). Let us denote the new assignment by C'. Then,

$$C'_u = C_u \setminus \{i^*\}, \qquad C'_v = C_v \cup \{i^*\}, \qquad C'_j = C_j, \forall j \in \{1, 2, \ldots, m\} \setminus \{u, v\}.$$

Consider Problem (10) for the new assignment given by C'. Then $\bar{\pi}$ clearly satisfies all the constraints for every $j \in B \setminus \{v\}$ and for every $k \in B$. It suffices to verify the remaining two constraints:

$$\min_{i \in C'_v}\{R_{iv} - R_{iu}\} = \min\left\{\underbrace{R_{i^*v} - R_{i^*u}}_{=\bar{\pi}_v - \bar{\pi}_u}, \underbrace{\min_{i \in C_v}\{R_{iv} - R_{iu}\}}_{\geq \bar{\pi}_v - \bar{\pi}_u}\right\} \geq \bar{\pi}_v - \bar{\pi}_u,$$

$$\min_{i \in C'_v}\{R_{iv}\} = \min\left\{\underbrace{R_{i^*v}}_{\geq \bar{\pi}_v}, \underbrace{\min_{i \in C_v}\{R_{iv}\}}_{\geq \bar{\pi}_v}\right\} \geq \bar{\pi}_v,$$

where we used the fact that Segment i^* had a positive surplus for Product u, hence

$$R_{i^*v} = \bar{\pi}_v + \underbrace{(R_{i^*u} - \bar{\pi}_u)}_{\geq 0}.$$

Therefore, the old prices $\{\bar{\pi}_j\}$ are still feasible with respect to the reassignment $\left\{C'_j\right\}$. This argument also applies in the case when there are multiple minimizers of $\min_{i \in C_u}\{R_{iu} - R_{iv}\}$ and all of the corresponding segments are reassigned to v. □

Remark 3.1. It is easy to see from the above proof that we can set the price of Product u to $(\bar{\pi}_u + \epsilon)$ for any

$$\epsilon \in \left[0, \min\left\{\min_{k \neq u}\left\{\min_{i \in C'_u}\{R_{iu} - R_{ik}\} + \bar{\pi}_k\right\} - \bar{\pi}_u, \min_{i \in C'_u}\{R_{iu}\} - \bar{\pi}_u\right\}\right]$$

and maintain feasibility with respect to the new assignment $\left\{C'_j\right\}$.

Lemma 3.3. *When $\delta_i = 0$ for every i, if Segment i^* is reassigned from Product u to Product v in the Dobson-Kalish Reassignment Heuristic, then the prices of Product u and its children in the spanning tree may increase while all other prices remain the same.*

Proof. First note that the only arc costs that can change after the reassignment are:

- arcs leaving node v (whose costs may decrease or stay the same), and
- arcs leaving node u (whose costs may increase or stay the same).

Thus, if the length of the shortest path to node 0 were to decrease, then the new path would have to go through node v.

Now we show that the shortest path from node v to node 0 will not change. Suppose for the sake of contradiction that the shortest path from v does change and it includes the arc $(v,\ h)$. Let $\bar{\pi}_v$ be the length of the shortest path from node v to node 0 in the original network. First let us assume that $v \neq 0$. We know that the shortest path from h to 0 remains the same, otherwise it must go through v and thus form a cycle, contradicting Lemma 3.2. Thus, our claim implies $\bar{\pi}_v > \bar{\pi}_h + R_{i^*v} - R_{i^*h}$. However, we know that in the old network $R_{i^*u} - \bar{\pi}_u = R_{i^*v} - \bar{\pi}_v$ and $R_{i^*u} - \bar{\pi}_u \geq R_{i^*h} - \bar{\pi}_h$, giving us $\bar{\pi}_v \leq \bar{\pi}_h + R_{i^*v} - R_{i^*h}$ which contradicts our claim. Furthermore, the *length* of the shortest path from v to node 0 does not change. Suppose that the shortest path of the original network included the arc $(v,\ \ell)$ with cost $r_{v\ell}$ (again, $r_{v\ell} = \min_{i \in C_v}\{R_{iv} - R_{i\ell}\}$). Since the shortest path network included the arc (u, v), this implies that $r_{v\ell} + r_{uv} \leq r_{u\ell}$, i.e., $r_{v\ell} + (R_{i^*u} - R_{i^*v}) \leq \min_{i \in C_u}\{R_{iu} - R_{i\ell}\} \leq R_{i^*u} - R_{i^*\ell}$. Thus, $r_{v\ell} \leq R_{i^*v} - R_{i^*\ell}$, so the cost of $(v,\ \ell)$ does not change when Segment i^* is moved to node v. Now let us assume $\ell = 0$, and suppose for the sake of contradiction that $\bar{\pi}_v = \gamma_v = \min_{i \in C_v}\{R_{iv}\} > R_{i^*v}$. However, we know that $\bar{\pi}_u = \bar{\pi}_v + R_{i^*u} - R_{i^*v} \leq R_{i^*u}$ from the upperbound constraints in (10), thus $\bar{\pi}_v \leq R_{i^*v}$. Thus, the cost of arc $(v,\ \ell)$ does not change. Similar arguments hold for the case where $v = 0$.

Thus, the shortest path from node v to node 0 remains the same. From this, we know that all shortest paths that did not go through u will remain the same. □

This result is also useful from an implementation perspective. In implementing the heuristic, we solved the shortest path problem using the Bellman-Ford-Moore algorithm. When we reassign a customer segment from node/product u to node/product v, we only need to update the shortest paths of node/product u and its children, speeding up the total computational time.

In [2], the authors claim that the Dobson-Kalish Reassignment Heuristic runs in polynomial time, namely $O(m^4 n)$ where m is the number of products and n is the number of customers. To show this result, a key claim that they make is "Each segment can be reassigned a maximum of m times before it is eliminated (assigned to 0)". However, we will show in the following counter-example that this is not the case. Let $m = 2$ and $n = 14$. Consider a problem instance with the following reservation prices and segment sizes:

Segment	R_{i1}	R_{i2}	N_i
1	0	1	1
2	ϵ	$1+\epsilon$	1
3	4	5	1
4	$4+5\epsilon$	$5+5\epsilon$	1
5	8	9	1
6	$8+9\epsilon$	$9+9\epsilon$	1
7	12	13	90
8	5	2	1
9	$5+5\epsilon$	$2+5\epsilon$	1
10	9	6	1
11	$9+9\epsilon$	$6+9\epsilon$	1
12	13	10	1
13	$13+13\epsilon$	$10+13\epsilon$	90
14	101	100	10

where we set $\epsilon = \frac{1}{100}$. In this example, $MaxR$ will initially assign Segments 1 through 7 to Product 2, and the rest to Product 1. In the Reassignment Heuristic, Segment 14 would move from Product 2 to 1. Then Segment 1 would be deleted, followed by Segment 2 being deleted, at which point Segment 14 moves back to Product 1. Then Segment 7 would be deleted, then followed by Segment 8 being deleted, at which point Segment 14 moves to Product 2. This process repeats, where Segment 14 moves to Product v, then two Segments of Product v are deleted, then Segment 14 moves back to the other Product, until only Segment 7, 13 and 14 remains. Segment 14 is ultimately reassigned a total of *six* times in the Dobson-Kalish Reassignment Heuristic, thus, the claim made in [2] is false.

At this time, there is no clear polynomial bound on the running time of the heuristic and it may be that the algorithm takes exponential time in the worst-case. We believe that the bound on the number of reassignments can be as bad as $\Theta(n^m)$. We leave further analysis for a future work.

However, the heuristic appears to make very few reassignments in practice. Table 1 shows the running time of the Dobson-Kalish heuristic on randomly generated reservation prices, the same data used in our more extensive computational experiments shown in Section 7. It is interesting to note that given n constant, the number of reassignments appear to decrease as m increases (excluding $m = 2$). This may be due to the property stated in Lemma 3.1, where the randomly generated reservation prices are more likely to have the special property of Lemma 3.1 when $n \ll m$. Thus, the MaxR heuristic may be more likely to yield a good solution when $n \ll m$, as a result, requiring fewer reassignments.

Table 2 shows the effect of initializing the branch-and-bound procedure with the Dobson-

n	m	# Reassignments	Time (CPU secs)
2	2	0	0.000
2	5	0	0.000
2	10	0	0.000
2	20	0	0.000
2	40	0	0.000
2	60	0	0.000
2	80	0	0.000
2	100	0	0.000
5	2	0	0.000
5	5	0	0.004
5	10	0	0.004
5	20	0	0.000
5	40	0	0.000
5	60	0	0.000
5	80	0	0.000
5	100	0	0.000
10	2	1	0.004
10	5	2	0.000
10	10	1	0.000
10	20	0	0.000
10	40	0	0.000
10	60	0	0.000
10	80	0	0.004
10	100	0	0.000
20	2	0	0.000
20	5	2	0.000
20	10	2	0.000
20	20	3	0.000
20	40	0	0.000
20	60	0	0.004
20	80	0	0.004
20	100	0	0.000

n	m	# Reassignments	Time (CPU secs)
40	2	2	0.000
40	5	5	0.000
40	10	6	0.004
40	20	4	0.004
40	40	1	0.004
40	60	1	0.008
40	80	0	0.004
40	100	0	0.004
60	2	1	0.000
60	5	8	0.000
60	10	9	0.008
60	20	5	0.008
60	40	8	0.024
60	60	5	0.024
60	80	5	0.032
60	100	4	0.032
80	2	5	0.000
80	5	2	0.000
80	10	31	0.028
80	20	13	0.028
80	40	9	0.032
80	60	8	0.044
80	80	5	0.040
80	100	7	0.072
100	2	2	0.000
100	5	10	0.004
100	10	21	0.020
100	20	29	0.060
100	40	11	0.048
100	60	14	0.104
100	80	18	0.164
100	100	5	0.064

Table 1: Number of reassignments and running time (in CPU seconds) of the Dobson-Kalish Reassignment Heuristic on randomly generated reservation prices.

		Branch-and-Bound Nodes		Time (CPU sec)		Best Feasible ObjVal		DK
n	m	without	with	without	with	without	with	ObjVal
10	5	54	45	0.11	0.10	5,563,941	5,563,941	5,538,951
10	20	0	0	0.11	0.11	6,462,579	6,462,579	6,411,532
10	60	0	0	1.04	1.08	6,535,702	6,535,702	6,535,702
10	100	0	0	3.74	3.91	6,189,501	6,189,501	6,189,501
20	5	396	283	1.18	0.94	11,408,143	11,408,143	11,408,143
20	20	281	281	2.32	2.27	12,619,821	12,619,821	12,569,173
20	60	1	1	3.26	3.40	12,535,858	12,535,858	12,524,341
20	100	0	0	12.19	12.88	13,042,924	13,042,924	13,042,924
40	5	458,801	587,676	3600.00	3600.00	21,203,889	21,203,889	21,203,889
40	20	254,801	235,301	3600.00	3600.00	24,428,264	24,428,264	24,339,805
40	60	25	25	11.56	11.46	26,914,470	26,914,470	26,889,864
40	100	51	51	38.98	40.79	25,350,493	25,350,493	25,344,259
60	5	282,647	289,701	3600.00	3600.00	30,718,212	30,718,212	30,668,313
60	20	53,357	66,501	3600.00	3600.00	37,953,896	37,953,896	37,846,041
60	60	28,301	45,701	3600.00	3600.00	38,848,221	38,848,221	38,840,190
60	100	24,101	25,981	3600.00	3600.00	40,591,695	40,591,695	40,577,937
100	5	173,901	161,547	3600.00	3600.00	52,002,035	52,047,301	50,410,990
100	20	41,015	39,001	3600.00	3600.00	58,890,675	59,713,318	59,707,018
100	60	8,778	10,301	3600.00	3600.00	65,731,774	65,848,403	65,813,404
100	100	11,459	7,223	3600.00	3600.00	67,219,374	67,215,991	67,168,897

Table 2: Effect of initializing the branch-and-bound procedure with and without the Dobson-Kalish Reassignment Heuristic solution with a one hour time limit on CPLEX. The column "without"/"with" corresponds to solving (8) without/with feeding CPLEX the heuristic solution as an initial feasible solution.

Kalish Reassignment Heuristic solution. We solved the MIP (8) and set a time limit of one hour. The column labeled "without" is the MIP result with CPLEX defaults and the column labeled "with" is the result when the Dobson-Kalish Reassignment Heuristic solution is given as an initial feasible solution. "Best Feasible ObjVal" is the best objective value that CPLEX found within the time limit. "DK ObjVal" is the objective value returned by the Dobson-Kalish Reassignment Heuristic solution. In some instances, starting with the heuristic solution results in fewer branch-and-bound nodes. For the ten instances where CPLEX could not prove optimality within one hour, starting with the heuristic solution yielded better objective values in three of the cases and even resulted in a worse solution in one case. Thus, it appears that this heuristic succeeds in finding a "good" feasible solution efficiently, but proving optimality is still difficult. To improve the latter situation, we explore various valid inequalities in Section 4.

3.1 Generalized Framework for Heuristic Algorithms

Based on our assumptions (A.1)–(A.4), any $\pi \in \mathbb{R}^m_+$ defines a corresponding product-segment assignment $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ uniquely when $\delta_i > 0, \forall i$. Let us denote such an assignment by $\mathcal{C}(\pi)$. Then, by construction, the assignment $\mathcal{C}(\pi)$ is feasible.

Once a feasible assignment $\mathcal{C}$ is given, we can easily compute the corresponding "optimal prices" by solving the shortest path problem on the underlying network. We denote these "optimal prices" by $\Pi(\mathcal{C})$. For the convenience of our design of heuristics, we would like to define the mapping $\Pi : \{0,1\}^{mn} \to \mathbb{R}^m$ so that it produces an m-vector. For those products u for which $C_u = \emptyset$, we set π_u to the "highest price" that does not change $\mathcal{C}$. That is, we need

$$\pi_u > \min_i \left\{ R_{iu} - \min_{j: i \in C_j} \{R_{ij} - \pi_j\} + \delta_i \right\}.$$

Note that even if $C_i \neq \emptyset$, for every i, it is rarely the case that for an arbitrary $\pi \in \mathbb{R}^m_+$, $\Pi(\mathcal{C}(\pi)) = \pi$.

We generalize the MaxR Heuristic to deal with positive δ_i. Following the rough idea of the MaxR Heuristic, we try to assign Segments to Products by maximizing the corresponding R_{ij} subject to maintaining the nonnegativity of the weights of all the arcs in the network representation.

Generalized Maximum Reservation Price Heuristic (GenMaxR)

Set $C_j := \emptyset$, for every j; $\mathcal{I} := \emptyset$.
repeat
  Find $i^* \in \{1, 2, \ldots, n\} \setminus \mathcal{I}$ and $u \in \{1, 2, \ldots, m\}$ such that R_{i^*u} is the maximum among all R_{ij} with $i^* \in \{1, 2, \ldots, n\} \setminus \mathcal{I}$ such that assigning Segment i^* to Product u maintains the nonnegativity of all weights on all the arcs.
  if no such i^*, u exist, **then**
    solve the shortest path problems on the network defined by the current assignment $\{C_j\}$ to determine the prices and STOP.
  else
    $\mathcal{I} := \mathcal{I} \bigcup \{i^*\}$, $C_u := C_u \bigcup \{i^*\}$.
  end if
until Maximum number of iterations.

Now, we are ready to describe the Generalized Reassignment Heuristic (GRH). Let $\mathcal{D}_\pi \subset \mathbb{R}^m$ denote the set of *search directions* to be considered. For $d \in \mathcal{D}_\pi$, $\alpha_d \geq 0$ will denote the *step size* along the direction d. At each major iteration, we start with prices π (optimal with respect to the current product-segment assignment). For the first iteration, we can begin with the output of GenMaxR Heuristic. For each $d \in \mathcal{D}_\pi$, we find a suitable $\alpha_d > 0$ and let

$$\pi'_d := \pi + \alpha_d d.$$

We then compute $\Pi\left(\mathcal{C}\left(\pi'_d\right)\right)$ and the corresponding objective function value. At the end of each major iteration, we set π to $\Pi\left(\mathcal{C}\left(\pi'_{d^*}\right)\right)$ which maximizes the total revenue among all $\Pi\left(\mathcal{C}\left(\pi'_d\right)\right)$ for $d \in \mathcal{D}_\pi$.

Generalized Reassignment Heuristic (GRH)

Require: a feasible product-segment assignment and its corresponding optimal spanning tree solution from solving (11) (e.g., via GenMaxR heuristic), including prices $\bar{\pi}$ and the corresponding objective function value $\bar{z}$.
repeat
  For each $d \in \mathcal{D}_{\bar{\pi}}$ compute $\Pi\left(\mathcal{C}(\bar{\pi} + \alpha_d d)\right)$, and the corresponding objective value $\bar{z}_d$.
  if $\max_{d \in \mathcal{D}_{\bar{\pi}}} \{\bar{z}_d\} \leq \bar{z}$, **then**
    STOP and return $\bar{\pi}$ and the corresponding assignment $\mathcal{C}(\bar{\pi})$.
  end if
  Let $\bar{z} := \max_{d \in \mathcal{D}_{\bar{\pi}}} \{\bar{z}_d\}$, $\bar{d} := \operatorname{argmax}_{d \in \mathcal{D}_{\bar{\pi}}}$, $\bar{\pi} := \Pi\left(\mathcal{C}(\bar{\pi} + \alpha_{\bar{d}} \bar{d})\right)$.
until Maximum number of iterations.

Remark 3.2. Dobson-Kalish Reassignment Heuristic is a special case of the above algorithm, where

$$\mathcal{D}_\pi := \{e_i : i \in B\} \bigcup \{0\},$$

independent of π.

Our approach to the problem exposes possibilities for improving the Dobson-Kalish heuristic within the framework of the general algorithm above. We can increase some prices and decrease

some others at the same time. We can perform these increases/decreases proportional to the current prices, or proportional to some "target" R_{ij} (which may be determined by N_i, M_j or N_iM_j) for each j, or proportional to the distance to such a target $(R_{ij} - \pi_j)$.

GRH also has the ability to "revive" products or customers that were eliminated. In the Dobson-Kalish heuristic, once a product is "deleted" (assigned no customers), that product will never be purchased in any of the subsequent iterations (similarly for segments that are reassigned to the 0 node/product). In the GRH framework, the prices can be modified such that these products and customers may become "active" again.

Another advantage of our heuristics GenMaxR and GRH over the Dobson-Kalish heuristic is that both of these algorithms can handle the cases $\delta > 0$ and $\delta_i > 0$ as well as the model based on δ_{ijk} (as mentioned in Section 2). Note that GenMaxR always ensures that the weights on the arcs are nonnegative (hence, there can never be a negative cost cycle) with or without δ. GRH is driven by the prices π_j rather than the assignments θ_{ij}; therefore, GRH also easily handles the various models based on nonzero δ, δ_i, δ_{ijk}. Extensive theoretical and computational studies of GRH and its variants are left to future work.

4 Valid Inequalities

Even with good feasible integer solutions, our experiments show that some problem instances are still intractable due to the weak lowerbounds produced by the LP relaxations (e.g., see Table 2 in Section 3). Thus, the key to improving the branch-and-bound procedure would be to generate good cutting planes for the LP relaxation of (8). The following are some valid inequalities for the mixed-integer programming problem.

4.1 Lower Bound Inequalities

These cuts use the fact that there exists a non-trivial lowerbound for product prices.

Lemma 4.1. *Let $\pi_j, \theta_{ij}, p_{ij}$, $i = 1, \ldots, n$, and, $j = 1, \ldots, m$ be optimal for (8). Then $\pi_j \geq \ell_j$, $\forall j = 1, \ldots, m$, where*

$$\ell_j := \min_{i=1,\ldots,n} \{R_{ij}\}.$$

Proof. From Lemma 2.1 (c), we know that there exist k such that $\pi_k = \min_{i \in C_k} \{R_{ik}\}$ where C_k is the set of customer segments who buy Product k. Suppose $i^* = \arg\min_{i \in C_k} \{R_{ik}\}$. Clearly, $\pi_k \geq \ell_k$. For all other feasible π_j where $j \neq k$, we have

$$R_{i^*k} - \pi_k \geq R_{i^*j} - \pi_j + \delta_{i^*}, \quad \forall j \neq k$$

$$\Rightarrow 0 \geq R_{i^*j} - \pi_j + \delta_{i^*}, \quad \forall j \neq k$$

$$\Rightarrow \pi_j \geq R_{i^*j} + \delta_{i^*} \geq \ell_j, \quad \forall j \neq k$$

which concludes the proof. □

From Lemma 4.1, we get the following valid inequalities:

$$p_{ij} \geq (\min_{l=1,\ldots,n} R_{lj})\theta_{ij}, \quad \forall i = 1,\ldots,n; j = 1,\ldots,m. \tag{13}$$

4.2 Negative Cost Cycle Inequalities

The following set of constraints eliminate negative 2-cycles in the underlying network in the solution of (8):

$$\theta_{ij} + \theta_{lk} \leq 1, \quad \forall i,l,\forall j,k, \text{ such that } (R_{ij} - R_{ik} - \delta_i) + (R_{lk} - R_{lj} - \delta_l) < 0. \tag{14}$$

Clearly, the Formulation (8) prevents negative cost cycles in the resulting network (11), but there are instances where the LP relaxation violates the above valid inequality.

In general, we cannot hope to generate negative k-cycles efficiently since for large k, the problem of determining the existence of such a cycle is $\mathcal{NP}$-complete. However, given a network, there are many algorithms which will efficiently retrieve many negative cycles of various lengths.

One approach is, after we solve the current LP relaxation of our MIP, we focus on the θ part of the optimal solution. Let $G(\theta, \epsilon)$ denote the graph constructed by assuming that for every $\theta_{ij} \geq \epsilon$, customer segment i buys product j. On this graph we run algorithms such as Barahona-Tardos modification [1] of Weintraub's [14] to retrieve some negative cost cycles in polynomial-time, and add the resulting inequalities on θ_{ij} to our current MIP formulation. We might want to start with $\epsilon = 1/2$ and gradually halve it if not enough negative cycle inequalities are generated.

We implemented another approach: For each node, we grew a path of negative total cost, making sure that every time we add an arc to the path, the current cost of the path remained negative. We extracted a negative cost cycle by using the Bellman-Ford-Moore algorithm.

4.3 Flow Cuts

Flow cuts [4, 11] are relevant cuts for (8) and are frequently generated automatically by CPLEX's mixed-integer programming solver [7]. Let $b_i = \max_j \{R_{ij}\}$ and $a_{ij} = R_{ij}$ for $i = 1,\ldots,n$ and $j = 1,\ldots,m$. From (8), we know that $\forall i = 1,\ldots,n$,

$$\sum_{j=1}^{m} p_{ij} \leq b_i, \qquad \text{and} \qquad p_{ij} \leq a_{ij}\theta_{ij}, \quad \forall j,$$

are valid inequalities for feasible mixed-integer solutions. We call the set $S_i \subset \{1,\ldots,m\}$ a *cover* if $\sum_{j\in S_i} a_{ij} > b_i$, and let the associated surplus be $\lambda_i = \sum_{j\in S_i} a_{ij} - b_i$. Applying the theory for flow cuts, we have the following valid inequality for (8):

$$\sum_{j\in S_i \bigcup L_i} p_{ij} + \sum_{j\in S_i} (a_{ij} - \lambda_i)_+ (1-\theta_{ij}) \le b_i + \sum_{j\in L_i} (\bar{a}_{ij} - \lambda)\theta_{ij}, \quad \forall i, \tag{15}$$

where S_i is a cover, $L_i := \{1,2,\ldots,m\} \setminus S_i$, $\bar{a}_{ij} := \max\{\max_{j\in S_i} a_{ij}, a_{ij}\}$, and $(x)_+ := \max(0,x)$.

4.4 First-Order Inter-Segment Inequalities

One of the weaknesses in the original formulation is that p_{ij}'s do not interact directly with each other across different segments. The variables π_j are essentially the only variables linking the constraints of different segments. The following "inter-segment inequalities" are valid inequalities that link θ variables of different segments in the same constraint.

Proposition 4.1. *Suppose* $\delta_i = 0, \forall i$,

$$R_{1j} - R_{2j} > R_{1k} - R_{2k}, \forall j \in \{1,2,\ldots,m\} \setminus \{k\}, \text{ and } R_{2k} \ge R_{1k}. \tag{16}$$

Then the inequality

$$\theta_{2k} \ge \theta_{1k} \tag{17}$$

is valid for the feasible region of (8).

Proof. If $\theta_{1k} = 0$ then the inequality $\theta_{2k} \ge \theta_{1k}$ is clearly valid. So, we may assume $\theta_{1k} = 1$. Then

$$\pi_j \ge R_{1j} - R_{1k} + \pi_k, \forall j \in \{1,2,\ldots,m\}.$$

Since $R_{1j} - R_{2j} > R_{1k} - R_{2k}, \forall j \in \{1,2,\ldots,m\} \setminus \{k\}$, we conclude

$$\pi_j > R_{2j} - R_{2k} + \pi_k, \forall j \in \{1,2,\ldots,m\},$$

i.e., Product k has the largest surplus for customer Segment 2. Since $R_{2k} \ge R_{1k}$ and $R_{1k} - \pi_k \ge 0$ (we assumed $\theta_{1k} = 1$), we have $R_{2k} \ge \pi_k$, i.e., the surplus is nonnegative for Segment 2 as well. Therefore, $\theta_{2k} = 1$. □

An algorithm to generate these inequalities easily follows: Form the vectors $(R_{1\cdot} - R_{i\cdot})$ for each $i \in \{2,3,\ldots,n\}$, and let

$$\underline{k}_i := \operatorname{argmin}\{R_{1j} - R_{ij} : j \in \{1,2,\ldots,n\}\},$$

$$\overline{k}_i := \operatorname{argmax}\{R_{1j} - R_{ij} : j \in \{1,2,\ldots,n\}\}.$$

If the argmin and argmax above are unique, then include the inequalities

$$\theta_{i\underline{k}_i} \geq \theta_{1\underline{k}_i} \text{ and } \theta_{i\overline{k}_i} \leq \theta_{1\overline{k}_i}$$

in the MIP formulation.

A few other remarks are in order. Firstly, we have at most $n(n-1)$ inequalities of this type. Secondly, if some of the strict inequalities in (16) are equations, we can still generate a similar valid inequality; but the resulting inequality would involve more than two θ_{ij} variables. Finally, we can easily generalize the above cut to positive δ_i.

4.5 Second-Order and Higher-Order Inter-Segment Inequalities

We can extend the last class of valid inequalities to involve more than two segments.

One apparent generalization is as follows:

Proposition 4.2. *Suppose that $\delta_i = 0, \forall i$, $R_{1k} \leq R_{3k}$ and $R_{2\ell} \leq R_{3\ell}$; further assume that for every $j \in \{1, 2, \ldots, m\} \setminus \{k, \ell\}$,*

- *either $R_{1j} - R_{3j} > R_{1k} - R_{3k}$*
- *or $R_{2j} - R_{3j} > R_{2\ell} - R_{3\ell}$*
- *possibly both.*

Then the inequality

$$\theta_{3k} + \theta_{3\ell} \geq \theta_{1k} + \theta_{2\ell} - 1$$

is valid for the feasible region of (8).

Proof. If $\theta_{1k} = 0$ or $\theta_{2\ell} = 0$ then the above inequality is clearly valid. So, we may assume $\theta_{1k} = \theta_{2\ell} = 1$. Then

$$\pi_j \geq R_{1j} - R_{1k} + \pi_k \text{ and } \pi_j \geq R_{2j} - R_{2\ell} + \pi_\ell, j \in \{1, 2, \ldots, m\}.$$

For each $j \notin \{k, \ell\}$,

$$R_{1j} - R_{3j} > R_{1k} - R_{3k} \text{ or } R_{2j} - R_{3j} > R_{2\ell} - R_{3\ell}.$$

Thus, for each such j

$$\pi_j > R_{3j} - R_{3k} + \pi_k \text{ or } \pi_j > R_{3j} - R_{3\ell} + \pi_\ell.$$

That is, for Segment 3, each product is either dominated by k or by ℓ. Since, $\pi_k \leq R_{1k} \leq R_{3k}$ and $\pi_\ell \leq R_{2\ell} \leq R_{3\ell}$, both products k and ℓ have nonnegative surplus for Segment 3. Therefore, $\theta_{3k} + \theta_{3\ell} \geq 1$ as desired. □

In general,

Theorem 4.1. *Suppose that* $\delta_i = 0, \forall i$, $R_{1,j_1} \leq R_{r+1,j_1}$, $R_{2,j_2} \leq R_{r+1,j_2}$, $\ldots$, $R_{r,j_r} \leq R_{r+1,j_r}$; *further assume that for every* $j \in \{1, 2, \ldots, m\} \setminus \{j_1, \ldots, j_r\}$,

- *either* $R_{1j} - R_{r+1,j} > R_{1j_1} - R_{r+1,j_1}$
- *or* $R_{2j} - R_{r+1,j} > R_{2j_2} - R_{r+1,j_2}$
- *or* $\cdots$
- *or* $R_{rj} - R_{r+1,j} > R_{rj_r} - R_{r+1,j_r}$.

Then the inequality

$$\sum_{k=1}^{r} \theta_{r+1,j_k} \geq \left(\sum_{k=1}^{r} \theta_{kj_k} \right) - (r-1)$$

is valid for the feasible region of (8).

Proof. Analogous to the proof of Proposition 4.2. □

Indeed, as r increases, we expect that the "strength" of these inequalities will decrease.

4.6 Other Valid Inequalities

The following are other valid inequalities specific to the problem.

With the original formulation, it is possible that the LP relaxation will result in p_{ij} larger than R_{lj} with θ_{ij} and θ_{lj} both strictly positive. The following is a valid inequality that attempts to cut such fractional solutions:

$$p_{ij} \leq R_{lj}\theta_{lj} + R_{ij}(1 - \theta_{lj}), \quad \forall i \neq l, \forall j. \tag{18}$$

Another observation is that if Segment i buys Product j and $R_{lj} > R_{ij}$, then Segment l must buy a product (as opposed to buying nothing). The following inequality reflects this property:

$$\sum_{k=1}^{m} \theta_{lk} \geq \theta_{ij}, \qquad \text{if} \quad R_{ij} < R_{lj}, \quad \forall i \neq l, \forall j. \tag{19}$$

5 Product Capacity

In all of our discussions thus far, we have assumed that there are no capacity limits for our products. However, this is an unrealistic assumption for products such as airline seats and hotel rooms. Furthermore, in some cases, companies may want to penalize against under-shooting a capacity. For example, if there is a large fixed cost or initial investment for Product j, the company may sacrifice revenue and decrease its price to ensure that all of the product is sold. We call such products *risk* products.

Let us assume that the company can sell up to Cap_j units of Product j, $Cap_j \geq 0$, $j = 1, \ldots, m$. Incorporating the capacity constraint to (8) is straightforward. We simply enforce the following linear constraint:

$$\sum_{i=1}^{n} N_i \theta_{ij} \leq Cap_j, \quad \forall j. \tag{20}$$

If Product j is a risk product, then let s_j be the slack of the above constraint and add $-\ell_j s_j$ to the objective function where $\ell_j \geq 0$ is a user-defined penalty for undershooting the capacity.

In some applications, overshooting the capacity slightly (at least during the planning stages) is allowed or even desired (e.g., the airline seats). To incorporate such features and to penalize them appropriately, we can utilize a goal programming approach. We can treat Cap_j as the *goal* and use the nonnegative variables s_j^- and s_j^+ to denote the amounts of negative and positive deviations from this goal. Then we still have a linear equation:

$$\sum_{i=1}^{n} N_i \theta_{ij} + s_j^- - s_j^+ = Cap_j, \quad \forall j.$$

Given $u_j > 0$, the unit cost of overshooting the capacity constraint for product j, we include the terms $-\ell_j s_j^- - u_j s_j^+$ in the objective function.

The heuristics for the maximum utility model, discussed in Section 3, can incorporate capacity constraints as well. Suppose the heuristic produces a product-segment assignment that overshoots the capacity for Product j. Then π_j can be increased until a segment switches to another product that gives them higher surplus. Thus, π_j will be increased until enough segments switch to another product so that the capacity constraint for Product j is met. If on the other hand, Product j is a risk product and its capacity was under utilized, then we may consider decreasing π_j until enough segments switch to Product j.

6 Extensions

The following subsections discuss extensions to our current model and give motivations for future work.

6.1 Stable-Set Formulation

Note that our arguments in Section 2.2 proved that the revenue management model always has some optimal prices which arise as the lengths of some paths in a network, where the arc lengths are given by r_{jk} and γ_j. Therefore, whenever the data for this subproblem is integer, namely $r_{jk} \in \mathbb{Z}, \forall j, k$ and $\gamma_j \in \mathbb{Z}, \forall j$, the optimal solution computed as the lengths of shortest paths will also be integer. If we further assume that

$$\delta_i = \delta, \quad \forall i \in \{1, 2, \ldots, n\}, \text{ and}$$

$$r_{jk} \text{ and } \gamma_k \text{ are integer multiples of } \delta, \forall j, k,$$

then there exist optimal prices π_j that are integer multiples of δ (Of course, this property will continue to hold if we add further constraints to the pricing subproblem, *as long as the underlying coefficient matrix continues to be Totally Unimodular*). Since it is easy to find simple upper and lower bounds on optimal values of π_j, under the assumption that R_{ij} and CS_i are all integer multiples of δ, we can restrict our attention to a finite list of prices. There could be other practical reasons to restrict potential prices to a (relatively small) list.

Given the above motivation, in this subsection, we suppose that each Product j can take on a finite set of prices, Π_j. Then we may wish to consider a stable set formulation of the maximum utility problem.

Let us first formulate this as a pure 0-1 programming problem where

$$y_{ij\rho} = \begin{cases} 1, & \text{if Customer } i \text{ buys Product } j \text{ at price } \rho, \\ 0, & \text{otherwise.} \end{cases}$$

Then the maximum utility problem can be formulated as:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{\rho \in \Pi_j} N_i \rho y_{ij\rho}, && && (21) \\ \text{s.t.} \quad & y_{ij\rho} + y_{i'j'\rho'} \leq 1, && \forall i, i', \forall j, j', \forall \rho \in \Pi_j, \rho' \in \Pi_{j'} : \\ & && R_{ij} - \rho < R_{ij'} - \rho' + \delta_i, \\ & y_{ij\rho} = 0, && \forall i, j, \rho : R_{ij} < \rho, \\ & \textstyle\sum_{j,\rho} y_{ij\rho} \leq 1, && \forall i, \\ & y_{ij\rho} + y_{i'j\rho'} \leq 1, && \forall i, i', \forall j, \forall \rho, \rho' \in \Pi_j : \rho \neq \rho', \\ & y_{ij\rho} \in \{0, 1\}, && \forall i, j, \rho, \end{aligned}$$

where the data parameters are as before.

Let $G = (V, E)$ be an undirected graph with nodes $(i, j, \rho) \in V$ for each Segment i, Product j and price $\rho \in \Pi_j$ where $R_{ij} - \rho \geq 0$. There exists an edge from node (i, j, ρ) to node (i', j', ρ') if:

1. $R_{ij} - \rho < R_{ij'} - \rho' + \delta$,
2. $i = i'$,
3. $j = j'$ and $\rho \neq \rho'$.

Suppose node $(i, j, \rho) \in V$ has weight $N_i\rho$. Then solving the maximum weight stable set problem on G with these weights above corresponds to solving the pure 0-1 problem (21).

6.2 Semidefinite Programming Relaxation

We can clearly explore semidefinite programming relaxations of our formulations as well.

The quadratic inequality

$$\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + 2\boldsymbol{a}^T\boldsymbol{x} + \alpha \leq 0$$

can be relaxed to

$$\text{tr}\left(\begin{bmatrix} \alpha & \boldsymbol{a}^T \\ \boldsymbol{a} & \boldsymbol{A} \end{bmatrix} Y\right) \leq 0,$$

where $Y \succeq 0$, $Y_{00} = 1$. Indeed, whenever $\text{rank}(Y) = 1$ we have an exact representation of the original quadratic inequality. Since we also have some 0,1 variables we can incorporate SDP relaxations of these as well.

To have an exact formulation, we would use the rank one matrix

$$Y := \begin{bmatrix} 1 \\ \boldsymbol{\theta} \\ \boldsymbol{\pi} \end{bmatrix} \begin{bmatrix} 1, \boldsymbol{\theta}^T, \boldsymbol{\pi}^T \end{bmatrix}$$

and stipulate

$$\text{diag}\left(\boldsymbol{\theta}\boldsymbol{\theta}^T\right) = \boldsymbol{\theta}.$$

Note that all the constraints and the objective function of problem (4) are linear in terms of the matrix variable Y. The nonconvex constraints $\theta_{ij} \in \{0, 1\}$ are replaced by the nonconvex constraint "$\text{rank}(Y) = 1$." Relaxing the rank-1 condition and replacing the rank-1 matrix by a positive semidefinite matrix and enforcing all the constraints as before, we have an SDP relaxation.

Application of a lift-and-project procedure based on semidefinite programming relaxations, like N_+ (see [10]) should be useful in obtaining tighter relaxations of the MILP due to the similarity of the combinatorial constraints to the stable set problem (and due to the power of N_+ on many stable set problems).

7 Computational Experiments

We tested the Dobson-Kalish Reassignment heuristic and our mixed-integer programming formulations of the maximum utility problem on randomly generated and real data sets, with $\delta_i = 0, \forall i$. On an SGI Altix 3800 with 64 Intel Itanium-2 processors each running at 1.3GHz and 122GB of RAM running SuSE Linux with SGI ProPack 4, we serially ran our implementation of the Dobson-Kalish heuristic against each test case and used the solutions obtained therefrom as initial feasible solutions for a run of the CPLEX 10.0 mixed-integer programming solver. The Dobson-Kalish code was implemented in C++ and compiled with gcc 4.1.0. The Dobson-Kalish pass was permitted to run to completion (see Table 1 for running time), while the MIP solver was terminated after three hours of execution (except this, CPLEX was run with default parameters).

7.1 Randomly Generated Data

For $n = 2, 5, 10, 20, 40, 60, 80, 100$ and $m = 2, 5, 10, 20, 40, 60, 80, 100$, we generated one random dense test case. In these random test cases, each R_{ij} is a random integer chosen from a uniform distribution on the integers between 512 and 1023, inclusive, and each N_i is a random integer between 500 and 799, inclusive. All of our computational experiments were run with $CS_i = 0$ for each customer segment i. The results are shown in Tables 3 to 6.

We present the results of solving the MIP formulation (8) with CPLEX. All experiments included the lower-bound inequalities (13) and the valid inequalities (18). The results under the column "inter" are obtained using the model that also includes the first-order inter-segment inequalities (17). The results under the column "cycle" include the negative cycle inequalities of Section 4.2 instead. The results under the column "both" include both sets of inequalities. The results under the column "neither" include neither inequalities. None of the other inequalities of Section 4 significantly strengthened the bounds in the branch-and-bound procedure.

Tables 3 and 4 show the total number of branch-and-bound nodes explored and the total running time in CPU seconds up to the three hour time limit. Table 5 shows the relative gap between the objective value of the Dobson-Kalish solution and the best lower-bound (i.e., the best feasible objective value) found by CPLEX. In Table 6, the column "DK vs Best UB" illustrates the relative gap between the objective value of the Dobson-Kalish solution and the best upper-bound found by CPLEX and the column "Best LB vs Best UB" illustrates the relative gap between the best lower-bound and the best upper-bound found by CPLEX (i.e., the MIP optimality gap). The results are not shown for $n \leq 20$ since all those instances solved to optimality within the time limit, thus the best lower-bound and upper-bound were equal (to a factor of 0.000001), and the "DK vs Best UB" would be the same as the values in Table 5 and "Best LB vs Best UB" would be 0.

The MIP formulation for "none", which is the MIP (8) with valid inequalities (13) and (18), has $2nm + m$ columns (nm of which correspond to binary variables), $5nm + n(n-1)m + n$ rows,

and $2m^2n + 9nm + 2n(n-1)m$ non-zero elements. The size of the formulation for each problem instance after the CPLEX presolve are reported in Tables 7 and 8. Presolve was able to reduce the problem size for these particular data sets but not significantly.

Results

From Table 3, we see that all instances with $n \leq 20$ solved to optimality within three hours, most of them solving at the root node (after the CPLEX generated its default cuts). There seems to be no significant difference in using or not using the inter-segment and negative cycle inequalities, except for $(n, m) = (20, 2)$ and $(20, 5)$ where the cuts do appear to help. It does appear that the negative cycle inequalities do decrease the number of nodes required to prove optimality (e.g., for $(n, m) = (10, 5), (20, 2), (20, 5), (20, 10), (20, 20)$) but the total computation time was higher than that for "inter". Although we are dynamically separating the negative cycle inequalities, the addition of the inequalities lead to larger subproblems and thus longer per node computation time.

Table 4 illustrates a more significant impact of the cuts. The instance $(n, m) = (40, 5)$ highlights this, where "neither" could not solve the problem to provable optimality in three hours but "both" solved it to optimality in about 20 minutes. The instances $(n, m) = (60, 2)$, (80,2), and (100,2) also highlight the advantages of having cuts since the implementation with cuts required significantly fewer branch-and-bound nodes to prove optimality. Once again, it appears that the negative cost cycle inequalities are effective in tightening the bounds but consequently slow down the per node running time. Thus, "inter" often dominates with respect to solution time. The larger problem sizes illustrate the tradeoff between improving bounds with cuts and increased per node computation time. For example, when $n = 100$, the difference in the number of nodes that CPLEX explored in three hours is lower when cuts are included, especially with negative cost cycle inequalities.

Table 5 shows that the Dobson-Kalish solution finds good feasible solutions in a fraction of a second (see Table 1 for the running times). Except for $(n, m) = (5, 10)$, $(80, 2)$, $(80, 5)$, $(100, 5)$, the heuristic found a solution within 1% of the best feasible solution found by CPLEX.

Most of the instances in Table 6 did not solve to provable optimality within the time limit. For $m = 2, 5$, "both" succeeds in finding the solution with the smallest optimality gap. However, "neither" and "inter" result in the smallest optimality gap for almost all other cases. This is probably because these formulations have much faster per node computation time than formulations with the negative cost cycle inequality, and thus are able to explore more nodes within the time limit. Interestingly, given n constant, it appears that the optimality gap decreases as m increases. It is possible that the Dobson-Kalish heuristic performs better when $n \ll m$ or the LP relaxation is tighter for $n \ll m$ or both, reflecting the properties of Lemmas 2.1 and 3.1.

		Number of Nodes				Time (CPU sec)			
n	**m**	**neither**	**inter**	**cycle**	**both**	**neither**	**inter**	**cycle**	**both**
2	2	0	0	0	0	0.00	0.00	0.00	0.00
2	5	0	0	0	0	0.00	0.01	0.00	0.00
2	10	0	0	0	0	0.01	0.01	0.01	0.01
2	20	0	0	0	0	0.02	0.02	0.02	0.02
2	40	0	0	0	0	0.06	0.06	0.06	0.06
2	60	0	0	0	0	0.14	0.14	0.14	0.15
2	80	0	0	0	0	0.25	0.25	0.26	0.26
2	100	0	0	0	0	0.41	0.42	0.42	0.43
5	2	0	0	0	0	0.01	0.00	0.01	0.00
5	5	0	0	0	0	0.01	0.01	0.01	0.02
5	10	57	62	61	65	0.11	0.12	0.16	0.18
5	20	0	0	0	0	0.07	0.07	0.07	0.08
5	40	0	0	0	0	0.33	0.32	0.34	0.35
5	60	0	0	0	0	0.44	0.44	0.45	0.45
5	80	0	0	0	0	0.96	0.96	1.00	1.00
5	100	0	0	0	0	1.15	1.15	1.18	1.18
10	2	0	0	0	0	0.07	0.08	0.06	0.08
10	5	21	38	16	16	0.16	0.21	0.18	0.21
10	10	12	18	12	18	0.23	0.27	0.27	0.32
10	20	0	0	0	0	0.17	0.19	0.19	0.20
10	40	0	0	0	0	0.56	0.57	0.59	0.61
10	60	0	0	0	0	1.32	1.35	1.43	1.42
10	80	0	0	0	0	2.98	3.09	3.55	3.50
10	100	0	0	0	0	4.40	4.46	4.75	4.73
20	2	51	27	20	23	0.67	0.65	0.60	0.66
20	5	162	94	128	60	1.98	1.92	2.74	2.26
20	10	392	376	253	286	4.74	4.68	8.15	8.06
20	20	458	459	219	219	5.92	6.29	8.94	9.20
20	40	214	214	229	229	7.54	8.06	22.35	22.77
20	60	1	1	1	1	5.95	6.04	7.32	7.42
20	80	0	0	0	0	5.45	5.85	6.56	6.66
20	100	0	0	0	0	9.63	9.97	11.82	11.64

Table 3: Number of branch-and-bound nodes and total running time on **randomly generated data** for $n = 2$ to 20. n is the number of customer segments and m is the number of products. "neither" is the result of solving (8) with (13) and (18). "inter" and "cycle" are the same as "neither" but with first-order inter-segment inequalities (17) and negative cost cycle inequalities, respectively. "both" is the combination of "inter" and "cycle".

		Number of Nodes				Time (CPU sec)			
n	**m**	**neither**	**inter**	**cycle**	**both**	**neither**	**inter**	**cycle**	**both**
40	2	157	93	56	68	4.05	5.14	4.32	4.87
40	5	759,774	659,612	91,554	19,616	10800.00	6392.82	6432.44	882.34
40	10	220,501	447,671	46,459	49,541	10800.00	10800.00	10800.00	10800.00
40	20	336,701	269,771	58,788	57,832	10800.00	10800.00	10800.00	10800.00
40	40	90	90	80	81	26.43	28.84	62.38	65.89
40	60	29	22	29	26	24.48	25.98	59.34	58.52
40	80	3,234	2,327	2,287	2,726	558.53	431.21	3151.82	3722.86
40	100	52	89	52	89	72.67	80.41	210.88	305.09
60	2	12,860	744	967	294	141.57	25.45	40.83	19.13
60	5	271,132	245,601	40,269	60,975	10800.00	10800.00	10800.00	10800.00
60	10	111,232	131,772	17,697	16,247	10800.00	10800.00	10800.00	10800.00
60	20	83,790	67,500	11,056	9,419	10800.00	10800.00	10800.00	10800.00
60	40	56,901	53,901	4,901	5,624	10800.00	10800.00	10800.00	10800.00
60	60	40,842	37,939	3,645	3,401	10800.00	10800.00	10800.00	10800.00
60	80	33,478	29,564	2,117	2,121	10800.00	10800.00	10800.00	10800.00
60	100	19,921	18,820	1,209	1,130	10800.00	10800.00	10800.00	10800.00
80	2	66,306	1,636	3,933	455	1375.45	83.96	241.03	50.25
80	5	99,801	63,906	17,741	20,790	10800.00	10800.00	10800.00	10800.00
80	10	48,753	31,401	7,204	6,089	10800.00	10800.00	10800.00	10800.00
80	20	22,601	19,411	3,389	2,642	10800.00	10800.00	10800.00	10800.00
80	40	15,210	28,101	1,640	1,528	10800.00	10800.00	10800.00	10800.00
80	60	16,426	13,609	1,260	1,414	10800.00	10800.00	10800.00	10800.00
80	80	1,802	1,082	691	610	1497.25	1279.04	6409.94	5996.34
80	100	14,058	12,831	481	490	10800.00	10800.00	10800.00	10800.00
100	2	239,908	1,990	5,778	836	6212.91	179.00	651.92	111.93
100	5	40,545	35,375	8,288	8,274	10800.00	10800.00	10800.00	10800.00
100	10	28,001	16,916	3,721	3,604	10800.00	10800.00	10800.00	10800.00
100	20	15,101	12,501	952	1,184	10800.00	10800.00	10800.00	10800.00
100	40	8,911	8,629	853	873	10800.00	10800.00	10800.00	10800.00
100	60	5,441	4,368	319	252	10800.00	10800.00	10800.00	10800.00
100	80	4,331	3,440	152	162	10800.00	10800.00	10800.00	10800.00
100	100	7,051	7,011	165	128	10800.00	10800.00	10800.00	10800.00

Table 4: Number of branch-and-bound nodes and total running time on **randomly generated data** for $n = 40$ to 100. n is the number of customer segments and m is the number of products. "neither" is the result of solving (8) with (13) and (18). "inter" and "cycle" are the same as "neither" but with first-order inter-segment inequalities (17) and negative cost cycle inequalities, respectively. "both" is the combination of "inter" and "cycle".

		DK vs Best LB						DK vs Best LB			
n	**m**	**neither**	**inter**	**cycle**	**both**	**n**	**m**	**neither**	**inter**	**cycle**	**both**
2	2	0.00	0.00	0.00	0.00	40	2	0.39	0.39	0.39	0.39
2	5	0.00	0.00	0.00	0.00	40	5	0.00	0.00	0.00	0.00
2	10	0.00	0.00	0.00	0.00	40	10	0.39	0.39	0.39	0.39
2	20	0.00	0.00	0.00	0.00	40	20	0.36	0.36	0.36	0.36
2	40	0.00	0.00	0.00	0.00	40	40	0.06	0.06	0.06	0.06
2	60	0.00	0.00	0.00	0.00	40	60	0.09	0.09	0.09	0.09
2	80	0.00	0.00	0.00	0.00	40	80	0.12	0.12	0.12	0.12
2	100	0.00	0.00	0.00	0.00	40	100	0.03	0.03	0.03	0.03
5	2	0.00	0.00	0.00	0.00	60	2	0.00	0.00	0.00	0.00
5	5	0.00	0.00	0.00	0.00	60	5	0.03	0.16	0.03	0.16
5	10	1.14	1.14	1.14	1.14	60	10	0.19	0.28	0.19	0.00
5	20	0.00	0.00	0.00	0.00	60	20	0.28	0.28	0.28	0.28
5	40	0.00	0.00	0.00	0.00	60	40	0.24	0.24	0.00	0.22
5	60	0.00	0.00	0.00	0.00	60	60	0.02	0.02	0.01	0.02
5	80	0.00	0.00	0.00	0.00	60	80	0.05	0.05	0.05	0.05
5	100	0.00	0.00	0.00	0.00	60	100	0.03	0.03	0.03	0.03
10	2	0.00	0.00	0.00	0.00	80	2	4.31	4.31	4.31	4.31
10	5	0.45	0.45	0.45	0.45	80	5	1.28	1.28	0.38	0.38
10	10	0.27	0.27	0.27	0.27	80	10	0.40	0.40	0.00	0.00
10	20	0.79	0.79	0.79	0.79	80	20	0.15	0.08	0.00	0.00
10	40	0.00	0.00	0.00	0.00	80	40	0.05	0.08	0.00	0.00
10	60	0.00	0.00	0.00	0.00	80	60	0.13	0.13	0.11	0.10
10	80	0.09	0.09	0.09	0.09	80	80	0.04	0.04	0.04	0.04
10	100	0.00	0.00	0.00	0.00	80	100	0.12	0.12	0.07	0.09
20	2	0.00	0.00	0.00	0.00	100	2	0.62	0.62	0.62	0.62
20	5	0.00	0.00	0.00	0.00	100	5	2.47	2.47	2.16	2.52
20	10	0.00	0.00	0.00	0.00	100	10	0.61	0.60	0.00	0.24
20	20	0.40	0.40	0.40	0.40	100	20	0.06	0.24	0.00	0.00
20	40	0.04	0.04	0.04	0.04	100	40	0.03	0.03	0.00	0.00
20	60	0.09	0.09	0.09	0.09	100	60	0.03	0.03	0.00	0.00
20	80	0.04	0.04	0.04	0.04	100	80	0.12	0.06	0.05	0.07
20	100	0.00	0.00	0.00	0.00	100	100	0.08	0.08	0.03	0.01

Table 5: Relative gap between the objective value of Dobson-Kalish heuristic and the best feasible objective value found by CPLEX within the three hour time limit on **randomly generated data**. n is the number of customer segments and m is the number of products. "neither" is the result of solving (8) with (13) and (18). "inter" and "cycle" are the same as "neither" but with first-order inter-segment inequalities (17) and negative cost cycle inequalities, respectively. "both" is the combination of "inter" and "cycle".

		DK vs Best UB				Best LB vs Best UB			
n	**m**	**neither**	**inter**	**cycle**	**both**	**neither**	**inter**	**cycle**	**both**
40	2	0.39	0.39	0.39	0.39	0.00	0.00	0.00	0.00
40	5	6.35	0.01	0.01	0.01	6.35	0.01	0.01	0.01
40	10	2.98	2.46	2.84	2.91	2.59	2.07	2.46	2.53
40	20	0.74	0.62	0.79	0.76	0.38	0.26	0.43	0.40
40	40	0.07	0.07	0.07	0.07	0.01	0.01	0.01	0.01
40	60	0.10	0.10	0.10	0.10	0.01	0.01	0.01	0.01
40	80	0.13	0.13	0.13	0.13	0.01	0.01	0.01	0.01
40	100	0.04	0.04	0.04	0.04	0.01	0.01	0.01	0.01
60	2	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00
60	5	8.80	6.42	6.93	3.66	8.77	6.26	6.90	3.50
60	10	5.23	5.36	5.18	5.02	5.06	5.09	5.00	5.02
60	20	1.35	1.38	1.43	1.27	1.07	1.10	1.15	0.99
60	40	0.57	0.54	0.59	0.61	0.33	0.31	0.59	0.39
60	60	0.19	0.21	0.23	0.24	0.17	0.18	0.22	0.22
60	80	0.16	0.20	0.21	0.23	0.11	0.16	0.17	0.19
60	100	0.16	0.12	0.19	0.18	0.13	0.09	0.16	0.15
80	2	4.32	4.32	4.32	4.31	0.01	0.01	0.01	0.00
80	5	9.81	8.15	8.32	6.14	8.64	6.95	7.97	5.79
80	10	5.88	5.80	5.79	5.80	5.50	5.42	5.79	5.80
80	20	3.17	3.17	3.21	3.28	3.02	3.09	3.21	3.28
80	40	0.67	0.65	0.71	0.70	0.62	0.58	0.71	0.70
80	60	0.28	0.26	0.30	0.28	0.15	0.13	0.19	0.18
80	80	0.05	0.05	0.05	0.05	0.01	0.01	0.01	0.01
80	100	0.28	0.28	0.30	0.30	0.16	0.16	0.23	0.21
100	2	0.63	0.62	0.63	0.62	0.01	0.01	0.01	0.00
100	5	14.61	12.69	13.88	11.27	12.45	10.48	11.97	8.98
100	10	7.62	7.61	7.74	7.71	7.06	7.06	7.74	7.49
100	20	6.14	6.16	6.21	6.24	6.08	5.93	6.21	6.24
100	40	0.41	0.41	0.43	0.42	0.38	0.37	0.43	0.42
100	60	0.68	0.68	0.71	0.69	0.66	0.65	0.71	0.69
100	80	0.48	0.48	0.49	0.49	0.36	0.43	0.44	0.42
100	100	0.19	0.19	0.21	0.20	0.12	0.12	0.18	0.20

Table 6: Relative gap between the objective value of Dobson-Kalish heuristic versus the best upper-bound found by CPLEX ("DK vs Best UB") and relative gap between the best feasible objective value versus the best upper-bound found by CPLEX ("Best LB vs Best UB") within the three hour time limit on **randomly generated data** for n=40 to 100. n is the number of customer segments and m is the number of products. "neither" is the result of solving (8) with (13) and (18). "inter" and "cycle" are the same as "neither" but with first-order inter-segment inequalities (17) and negative cost cycle inequalities, respectively. "both" is the combination of "inter" and "cycle".

		From Formulation			After CPLEX Presolve		
n	**m**	**ncols**	**nrows**	**nnz**	**ncols**	**nrows**	**nnz**
2	2	10	26	60	8	16	34
2	5	25	62	210	20	42	145
2	10	50	122	620	40	82	440
2	20	100	242	2,040	80	162	1,480
2	40	200	482	7,280	158	316	5,303
2	60	300	722	15,720	240	482	11,627
2	80	400	962	27,360	320	642	20,313
2	100	500	1,202	42,200	400	802	31,376
5	2	22	95	210	20	67	148
5	5	55	230	675	50	165	520
5	10	110	455	1,850	100	325	1,490
5	20	220	905	5,700	200	645	4,772
5	40	440	1,805	19,400	400	1,284	16,748
5	60	660	2,705	41,100	600	1,925	35,910
5	80	880	3,605	70,800	800	2,565	62,271
5	100	1,100	4,505	108,500	1,000	3,202	95,825
10	2	42	290	620	40	192	418
10	5	105	710	1,850	100	469	1,343
10	10	210	1,410	4,700	200	929	3,638
10	20	420	2,810	13,400	400	1,847	11,068
10	40	840	5,610	42,800	800	3,688	37,330
10	60	1,260	8,410	88,200	1,200	5,524	78,763
10	80	1,680	11,210	149,600	1,597	7,354	135,140
10	100	2,100	14,010	227,000	1,999	9,202	207,135
20	2	82	980	2,040	80	591	1,256
20	5	205	2,420	5,700	200	1,455	3,745
20	10	410	4,820	13,400	400	2,885	9,428
20	20	820	9,620	34,800	800	5,751	26,650
20	40	1,640	19,220	101,600	1,599	11,485	84,446
20	60	2,460	28,820	200,400	2,400	17,211	173,480
20	80	3,280	38,420	331,200	3,198	22,943	293,440
20	100	4,100	48,020	494,000	3,999	28,684	444,866

Table 7: Size of the mixed-integer programming problem (8) with (13) and (18) on **random data**. "From Formulation" is the data-independent size calculated from the formulation (i.e., ncols=$2nm + m$, nrows=$5nm + n(n-1)m + n$, nnz= $2m^2n + 9nm + 2n(n-1)m$) and "After CPLEX Presolve" is the size *after* CPLEX preprocess.

		From Formulation			After CPLEX Presolve		
n	**m**	**ncols**	**nrows**	**nnz**	**ncols**	**nrows**	**nnz**
40	2	162	3,560	7,280	160	1,989	4,132
40	5	405	8,840	19,400	400	4,916	11,519
40	10	810	17,640	42,800	799	9,791	26,978
40	20	1,620	35,240	101,600	1,599	19,544	69,764
40	40	3,240	70,440	267,200	3,199	39,047	202,688
40	60	4,860	105,640	496,800	4,796	58,556	398,505
40	80	6,480	140,840	790,400	6,395	78,056	657,541
40	100	8,100	176,040	1,148,000	7,998	97,598	980,150
60	2	242	7,740	15,720	240	4,189	8,612
60	5	605	19,260	41,100	600	10,376	23,305
60	10	1,210	38,460	88,200	1,199	20,693	52,550
60	20	2,420	76,860	200,400	2,398	41,325	128,838
60	40	4,840	153,660	496,800	4,798	82,589	352,796
60	60	7,260	230,460	889,200	7,197	123,851	671,805
60	80	9,680	307,260	1,377,600	9,596	165,148	1,085,911
60	100	12,100	384,060	1,962,000	11,992	206,377	1,594,735
80	2	322	13,520	27,360	320	7,182	14,676
80	5	805	33,680	70,800	798	17,834	39,071
80	10	1,610	67,280	149,600	1,600	35,599	86,122
80	20	3,220	134,480	331,200	3,199	71,070	203,895
80	40	6,440	268,880	790,400	6,396	142,113	534,875
80	60	9,660	403,280	1,377,600	9,594	213,128	992,783
80	80	12,880	537,680	2,092,800	12,792	284,115	1,577,644
80	100	16,100	672,080	2,936,000	15,992	355,143	2,289,883
100	2	402	20,900	42,200	400	10,968	22,330
100	5	1,005	52,100	108,500	999	27,280	58,825
100	10	2,010	104,100	227,000	1,997	54,448	127,556
100	20	4,020	208,100	494,000	3,999	108,835	294,972
100	40	8,040	416,100	1,148,000	7,997	217,593	748,973
100	60	12,060	624,100	1,962,000	11,992	326,295	1,361,610
100	80	16,080	832,100	2,936,000	15,990	435,028	2,133,388
100	100	20,100	1,040,100	4,070,000	19,989	543,781	3,064,337

Table 8: Size of the mixed-integer programming problem (8) with (13) and (18) on **random data**. "From Formulation" is the data-independent size calculated from the formulation (i.e., ncols=$2nm + m$, nrows=$5nm + n(n-1)m + n$, nnz= $2m^2n + 9nm + 2n(n-1)m$) and "After CPLEX Presolve" is the size *after* CPLEX preprocess.

7.2 Real Data

We have been collaborating with a company in the tourism sector who provided us with some raw customer purchase order data. Using some data mining and optimization techniques, we estimated the reservation prices R_{ij} from the raw data (see [12] for some of the details).

For a given season in their demand cycle, we clustered their customers into 3,095 segments and classified their product line into 2274 products, i.e., $n = 3095$ and $m = 2274$ (due to a non-disclosure agreement, we cannot provide more detailed information). Our implementation of the Dobson-Kalish reassignment heuristic spent 4,833 seconds of CPU time and performed 462 reassignments. We ran CPLEX on MIP formulation (8) with only the lower-bound inequalities (13) due to the size of the problem. This formulation corresponds to 14,078,334 variables (7,038,030 of them being binary), 35,193,245 constraints, and 32,058,235,114 non-zeros, independent of the data. After CPLEX pre-solve, 23,252 variables, 42,295 constraints, and 242,533 non-zeros remained for this particular data set; most of the reduction came from the many zero-valued reservation prices in this data. The solution delivered by the heuristic was within 11.12% of the optimal value of the LP relaxation, and proven to be 7.23% of the optimal MIP objective value after 10 minutes and 7.05% of the optimal MIP objective value after seven days of running CPLEX. After one week of computation, CPLEX was unable to find a better feasible solution than the heuristic solution.

From this large data set, we generated some smaller instances of the problem. Our goal is to compare the performance of our approach on such data sets to see whether the performance of our algorithm on randomly generated data of the previous subsection is significantly different. Given n and m, we applied a simple hill-climbing heuristic to extract a reasonably dense $n \times m$ sub-matrix of the matrix of reservation prices present in this "real" data. Specifically, for $n = 2, 5, 10, 20, 40, 60, 80, 100$ and $m = 2, 5, 10, 20, 40, 60, 80, 100$, we extracted an $n \times m$ sub-matrix of this matrix of reservation prices and repeated the same procedure as was applied to the random data. We included the first-order inter-segment inequalities (17) but left out the negative cycle inequalities (14) since "inter" appeared to result in the best total running time in general.

Tables 9 and 10 illustrate the results, where n is the number of customer segments, m is the number of products, "ncols" is the resulting number of columns after the CPLEX presolve, "nrows" is the resulting number of rows after the CPLEX presolve, "nnz" is the resulting number of nonzero elements after the CPLEX presolve, "Nodes" is the total number of branch-and-bound nodes explored, "Time" is the total CPU seconds, "DKvsLB" is the relative gap between the heuristic objective value and the best lower-bound, "DKvsUB" is the relative gap between the heuristic objective value and the best upper-bound, and "LBvsUB" is the relative gap between the best lower-bound and the best upper-bound. Again, CPLEX was run with a three hour time limit.

Interestingly, the computational results on the real data are starkly different from that of the random data. This is especially highlighted for $n \geq 20$. With random data, the instances

$(n, m) = (20, 10), (20, 20), (20, 40), (20, 60), (20, 80)$ all solved within the time limit, whereas the real data could not. The relative gap of the heuristic solution and the final optimality gap is significantly larger with the real data than with the random data. This may be due to the fact that it is much more likely for randomly generated data to have the property of Lemma 2.1 (or close to it) so that the heuristic could find strong feasible solutions and the LP relaxation would be tight. However, this is highly unlikely for real data since it is most likely that there would be strong dependencies of the reservation prices across customers and products.

Pre-solve succeeds in significantly reducing the size of the problem due to the existence of many reservation prices with value 0 in these data sets. Tables 11 and 12 illustrate the size of the formulation (8) with valid inequalities (13) and (18) on the real data sets. The data-independent size of the MIP formulation is $2nm+m$ columns (nm of which are binary), $5nm+n(n-1)m+n$ rows, and $2m^2n + 9nm + 2n(n-1)m$ non-zero elements, but we see that CPLEX Pre-solve is able to significantly reduce the problem size, unlike the randomly generated instances where all the reservation prices were non-zero (see Tables 7 and 8).

7.3 Day-to-Day Updating of Optimal Prices

Our heuristic found a solution within 7.05% optimality in about an hour and a half for this large data set above with thousands of customer segments and thousands of products. While this is impressive given the size of the MIP formulation, in practice, we may need to recompute near-optimal prices many times per day — either responding to the changes in the consumer market, the supplier market or to competitor prices or products. In such operations, our framework for the Generalized Reassignment Heuristic is useful. As the data changes, we take the most recent prices, π. Then, $\mathcal{C}(\pi)$, with respect to the new data, results in the corresponding segment-product assignments. We resolve the shortest path problem utilizing the most recent shortest path tree as much as possible and re-start the Dobson-Kalish algorithm.

We test the computational requirements for updating prices on two potential scenarios:

(a) The company introduces a new product,

(b) The price of a competitor product changes.

Scenario (a) corresponds to incrementing m and incorporating the reservation prices for each customer segment for this new product. In scenario (b), if the price change increases or decreases the competitor surplus for some set of customers by Δ, then this corresponds to subtracting or adding, respectively, Δ from/to R_{ij}'s $\forall j$ of these customers. For each scenario, we ran the Dobson-Kalish heuristic on the modified data starting from the previous solution instead of starting from scratch (i.e., from the MaxR solution).

For scenario (a), we generated new reservation prices and set the price of the new product to the minimum price such that none of the previously purchased products would be "eliminated".

n	m	ncols	nrows	nnz	nodes	time	DKvsLB	DKvsUB	LBvsUB
2	2	8	9	20	0	0.00	0.00	100.00	100.00
2	5	20	43	145	0	0.01	0.00	0.00	0.00
2	10	40	84	436	0	0.01	0.00	100.00	100.00
2	20	64	119	812	0	0.01	0.00	100.00	100.00
2	40	100	135	1,380	0	0.02	0.00	100.00	100.00
2	60	100	135	1,445	0	0.02	0.00	100.00	100.00
2	80	96	133	1,221	0	0.02	9.01	9.01	0.00
2	100	104	137	1,256	0	0.02	0.00	100.00	100.00
5	2	20	72	156	1	0.02	13.38	13.38	0.00
5	5	50	162	491	27	0.07	0.00	0.00	0.00
5	10	100	328	1,447	172	0.22	7.74	7.74	0.00
5	20	172	398	1,952	0	0.02	0.00	100.00	100.00
5	40	315	766	6,610	875	1.22	0.00	0.00	0.00
5	60	430	763	6,892	0	0.07	0.00	100.00	100.00
5	80	520	781	8,524	0	0.07	0.00	100.00	100.00
5	100	575	946	11,177	14	0.26	0.10	0.10	0.00
10	2	40	222	474	4	0.08	2.14	2.14	0.00
10	5	100	508	1,386	710	1.06	2.92	2.92	0.00
10	10	200	966	3,689	16,385	29.00	31.38	31.39	0.01
10	20	388	1,742	9,592	64,517	170.39	0.24	0.25	0.01
10	40	650	2,289	15,248	5,021	12.88	0.15	0.16	0.01
10	60	870	2,491	17,252	27	0.85	0.41	0.41	0.00
10	80	1,090	2,559	23,094	332	1.88	1.55	1.55	0.00
10	100	1,280	2,846	22,025	406	3.03	4.82	4.82	0.00
20	2	80	724	1,524	11	0.51	2.06	2.06	0.00
20	5	199	1,607	4,000	71,411	201.72	14.59	14.59	0.01
20	10	397	2,961	9,265	1,642,101	10,800.00	41.88	49.44	13.01
20	20	746	5,202	21,084	1,128,301	10,800.00	19.80	41.16	26.64
20	40	1,271	7,975	36,921	637,701	10,800.00	3.90	19.91	16.67
20	60	1,691	8,720	44,107	805,901	10,800.00	0.21	8.55	8.36
20	80	2,111	9,089	51,135	776,701	10,800.00	5.03	8.90	4.08
20	100	2,531	9,313	53,147	30,400	447.22	9.84	9.85	0.01

Table 9: Computational results on **real data** for $n \leq 20$ on the MIP formulation (8) with (13), (18) and (17) with a three hour time limit. n is the number of customer segments and m is the number of products. "ncols", "nrows", "nnz" are the resulting number of columns, rows and non-zeros, respectively, of the MIP after CPLEX pre-solve. "nodes" and "time" are the number of nodes explored and total computation time, respectively, of CPLEX. "DKvsLB", "DKvsUB", and "LBvsUB" are the relative gaps between the objective value of Dobson-Kalish heuristic versus the best feasible objective value found by CPLEX, heuristic objective value versus the best upper-bound found by CPLEX, and the best feasible objective value versus the best upper-bound found by CPLEX, respectively.

n	m	ncols	nrows	nnz	nodes	time	DKvsLB	DKvsUB	LBvsUB
40	2	160	2,565	5,280	268	5.69	0.55	0.55	0.00
40	5	393	5,438	12,336	842,353	10,674.75	38.46	38.47	0.01
40	10	756	9,917	25,442	532,301	10,800.00	33.02	49.16	24.09
40	20	1,367	16,753	49,665	354,501	10,800.00	30.45	52.40	31.56
40	40	2,338	24,543	81,474	248,901	10,800.00	18.40	39.47	25.82
40	60	3,243	29,429	101,771	246,801	10,800.00	13.21	22.70	10.93
40	80	4,093	31,669	114,097	197,279	10,800.00	6.12	16.74	11.31
40	100	4,915	32,475	121,033	207,301	10,800.00	9.40	18.04	9.53
60	2	240	5,383	10,994	1,252	29.58	0.28	0.28	0.00
60	5	586	11,638	25,417	244,401	10,800.00	20.26	34.28	17.58
60	10	1,076	20,396	47,866	162,301	10,800.00	28.42	50.21	30.44
60	20	1,957	34,618	89,235	108,701	10,800.00	18.26	44.62	32.25
60	40	3,379	49,708	140,533	66,701	10,800.00	25.62	43.97	24.66
60	60	4,703	57,387	175,213	84,131	10,800.00	25.84	45.46	26.46
60	80	5,978	62,880	190,984	74,101	10,800.00	20.41	32.57	15.28
60	100	7,220	65,693	207,378	103,882	10,800.00	2.91	20.29	17.90
80	2	320	9,252	18,790	1,789	73.08	0.32	0.33	0.01
80	5	762	20,783	44,249	141,987	10,800.00	0.00	19.45	19.45
80	10	1,389	35,571	79,705	70,508	10,800.00	29.27	46.90	24.93
80	20	2,520	57,748	138,506	37,501	10,800.00	31.62	55.90	35.51
80	40	4,389	82,533	213,557	35,001	10,800.00	28.82	49.51	29.06
80	60	6,128	95,325	254,820	32,101	10,800.00	31.17	47.63	23.92
80	80	7,825	102,420	299,197	45,271	10,800.00	18.55	41.53	28.21
80	100	9,479	107,935	310,861	32,199	10,800.00	16.71	30.34	16.37
100	2	400	14,249	28,852	9,055	408.33	0.58	0.59	0.01
100	5	946	31,649	66,667	64,437	10,800.00	21.48	36.76	19.46
100	10	1,693	53,664	116,907	35,195	10,800.00	28.97	52.45	33.05
100	20	3,065	86,886	199,515	22,229	10,800.00	31.38	55.74	35.50
100	40	5,375	121,525	295,681	16,224	10,800.00	27.85	50.01	30.72
100	60	7,542	140,487	357,700	14,001	10,800.00	23.34	42.42	24.89
100	80	9,643	151,340	397,789	18,402	10,800.00	15.47	32.46	20.11
100	100	11,707	156,940	442,429	25,513	10,800.00	16.12	34.62	22.05

Table 10: Computational results on **real data** for $n \geq 40$ on the MIP formulation (8) with (13), (18) and (17) with a three hour time limit. n is the number of customer segments and m is the number of products. "ncols", "nrows", "nnz" are the resulting number of columns, rows and non-zeros, respectively, of the MIP after CPLEX pre-solve. "nodes" and "time" are the number of nodes explored and total computation time, respectively, of CPLEX. "DKvsLB", "DKvsUB", and "LBvsUB" are the relative gaps between the objective value of Dobson-Kalish heuristic versus the best feasible objective value found by CPLEX, heuristic objective value versus the best upper-bound found by CPLEX, and the best feasible objective value versus the best upper-bound found by CPLEX, respectively.

		From Formulation			After CPLEX Presolve		
n	**m**	**ncols**	**nrows**	**nnz**	**ncols**	**nrows**	**nnz**
2	2	10	26	60	8	8	18
2	5	25	62	210	20	42	143
2	10	50	122	620	40	82	432
2	20	100	242	2,040	64	118	810
2	40	200	482	7,280	100	133	1,376
2	60	300	722	15,720	100	133	1,441
2	80	400	962	27,360	96	131	1,217
2	100	500	1,202	42,200	104	135	1,252
5	2	22	95	210	20	67	146
5	5	55	230	675	50	160	487
5	10	110	455	1,850	100	319	1,429
5	20	220	905	5,700	172	379	1,914
5	40	440	1,805	19,400	315	755	6,588
5	60	660	2,705	41,100	430	749	6,864
5	80	880	3,605	70,800	520	773	8,508
5	100	1,100	4,505	108,500	575	939	11,163
10	2	42	290	620	40	192	414
10	5	105	710	1,850	100	470	1,310
10	10	210	1,410	4,700	200	930	3,617
10	20	420	2,810	13,400	388	1,690	9,488
10	40	840	5,610	42,800	650	2,250	15,170
10	60	1,260	8,410	88,200	870	2,455	17,180
10	80	1,680	11,210	149,600	1,090	2,516	23,008
10	100	2,100	14,010	227,000	1,280	2,798	21,929
20	2	82	980	2,040	80	590	1,256
20	5	205	2,420	5,700	199	1,450	3,686
20	10	410	4,820	13,400	397	2,770	8,883
20	20	820	9,620	34,800	746	5,010	20,700
20	40	1,640	19,220	101,600	1,271	7,754	36,479
20	60	2,460	28,820	200,400	1,691	8,492	43,651
20	80	3,280	38,420	331,200	2,111	8,874	50,705
20	100	4,100	48,020	494,000	2,531	9,144	52,809

Table 11: Size of the mixed-integer programming problem (8) with (13) and (18) on **real data**. "From Formulation" is data-independent the size calculated from the formulation (i.e., ncols=$2nm + m$, nrows=$5nm + n(n-1)m + n$, nnz= $2m^2n + 9nm + 2n(n-1)m$) and "After CPLEX Presolve" is the size *after* CPLEX preprocess.

		From Formulation			After CPLEX Presolve		
n	**m**	**ncols**	**nrows**	**nnz**	**ncols**	**nrows**	**nnz**
40	2	162	3,560	7,280	160	1,989	4,128
40	5	405	8,840	19,400	393	4,678	10,816
40	10	810	17,640	42,800	756	9,070	23,748
40	20	1,620	35,240	101,600	1,367	16,286	48,731
40	40	3,240	70,440	267,200	2,338	24,131	80,650
40	60	4,860	105,640	496,800	3,243	28,781	100,475
40	80	6,480	140,840	790,400	4,093	31,151	113,061
40	100	8,100	176,040	1,148,000	4,915	31,923	119,929
60	2	242	7,740	15,720	240	4,186	8,600
60	5	605	19,260	41,100	586	9,995	22,131
60	10	1,210	38,460	88,200	1,076	18,659	44,392
60	20	2,420	76,860	200,400	1,957	33,091	86,181
60	40	4,840	153,660	496,800	3,379	48,434	137,985
60	60	7,260	230,460	889,200	4,703	56,739	173,917
60	80	9,680	307,260	1,377,600	5,978	61,871	188,966
60	100	12,100	384,060	1,962,000	7,220	64,504	205,000
80	2	322	13,520	27,360	320	7,181	14,648
80	5	805	33,680	70,800	762	17,125	36,933
80	10	1,610	67,280	149,600	1,389	31,612	71,787
80	20	3,220	134,480	331,200	2,520	54,879	132,768
80	40	6,440	268,880	790,400	4,389	79,314	207,119
80	60	9,660	403,280	1,377,600	6,128	92,181	248,532
80	80	12,880	537,680	2,092,800	7,825	100,460	295,277
80	100	16,100	672,080	2,936,000	9,479	105,373	305,737
100	2	402	20,900	42,200	400	10,977	22,308
100	5	1,005	52,100	108,500	946	26,253	55,875
100	10	2,010	104,100	227,000	1,693	47,417	104,413
100	20	4,020	208,100	494,000	3,065	80,834	187,411
100	40	8,040	416,100	1,148,000	5,375	115,770	284,171
100	60	12,060	624,100	1,962,000	7,542	134,793	346,312
100	80	16,080	832,100	2,936,000	9,643	146,145	387,399
100	100	20,100	1,040,100	4,070,000	11,707	152,674	433,897

Table 12: Size of the mixed-integer programming problem (8) with (13) and (18) on **real data**. "From Formulation" is the data-independent size calculated from the formulation (i.e., ncols=$2nm + m$, nrows=$5nm + n(n-1)m + n$, nnz= $2m^2n + 9nm + 2n(n-1)m$) and "After CPLEX Presolve" is the size *after* CPLEX preprocess.

Keeping all other prices the same, we started the heuristic with the corresponding segment-product assignment. The heuristic required 13 reassignments and terminated in 120.4 CPU seconds.

For scenario (b), we tested four of the following variations. Note that when the reservation prices are changed by a constant across all products for a given set of segments, the previous segment-product assignment remains feasible.

- Competitor surplus decreases by \$200 for 10 customer segments. This required 6 iterations of the Dobson-Kalish Reassignment Heuristic and terminated in 55.252 CPU seconds.
- Competitor surplus increases by \$200 for 10 customer segments. This required 2 iterations of the Dobson-Kalish Reassignment Heuristic and terminated in 18.304 CPU seconds.
- Competitor surplus decreases by \$200 for 500 customer segments. This required 36 iterations of the Dobson-Kalish Reassignment Heuristic and terminated in 329.284 CPU seconds.
- Competitor surplus increases by \$200 for 500 customer segments. This required 80 iterations of the Dobson-Kalish Reassignment Heuristic and terminated in 737.092 CPU seconds.

We see that the approach returns near optimal prices within a matter of minutes for small changes in the data. This indicates that our approach can be used not only in strategic optimal pricing decisions but also in day-to-day operations and in the analysis of what-if scenarios.

8 Conclusion

This paper explored mixed-integer programming formulations of the maximum utility model, implemented heuristic algorithms and developed valid inequalities to optimize computation time. As our results illustrate, our Dobson-Kalish heuristic is very effective and efficient in practice. We also showed that our approaches can be used to solve very large scale instances arising in the tourism sector. One of the reasons for the large problems sizes is the large number of possible bundles (the products in our models are actually bundles of the individual products offered by the company). Thus, the large data instance in the computational experiments section solved optimal bundling and optimal pricing problems together.

There are clearly many potential extensions and improvements we need to consider. As mentioned earlier, we have yet to determine the worst-case running time of the Dobson-Kalish Reassignment Heuristic and hammer out the details of the Generalized Reassignment Heuristic. In addition, the computational experiments have shown that the valid inequalities are effective in cutting off fractional solutions, but they slowed down the total solution time. Thus, we need to improve our dynamic separation procedure, possibly removing redundant inequalities.

Semidefinite relaxations, as discussed in Section 6, may also yield strong bounds and interesting mathematical insights.

Although, our motive in this paper was to explore efficient methods to solve the maximum utility model, we conclude by showing some preliminary evidence of the advantages of optimization-based pricing strategies. To gauge the practical impact of our approach, we tested our solution against the actual historical sales. Under very conservative assumptions that imposed disadvantages to our model, optimal prices given by our model led to a 23% increase in total revenue. It was also interesting to note that our optimal prices "converted" consumers from low-end products to high-end products. We clustered the thousands of individual products into 18 product groups of similar properties and illustrate the results in the table below (for confidentiality purposes, the dollar figures were scaled). The product groups 1 to 10 correspond to the mid to high-end products and the product groups 11 to 18 correspond to the low-end products. In this particular company, managers of the lower-end products tended to "dump" prices when sales were under expectation, thus "cannibalizing" other products and negatively impacting the total revenue. Our initial results seem to reflect the benefit of a more global approach to product line pricing.

Product Group	**Estimated Sales Using Our Prices**	**Actual Sales**	**Difference**
Group 1	$43,427.67	$29,947.18	$13,480.49
Group 2	$41,281.08	$28,594.49	$12,686.60
Group 3	$54,214.30	$30,667.04	$23,547.26
Group 4	$123,820.55	$67,193.30	$56,627.26
Group 5	$111,165.09	$35,125.78	$76,039.31
Group 6	$179.74	$175.21	$4.53
Group 7	$1,454.74	$422.95	$1,031.79
Group 8	$143,830.59	$79,968.27	$63,862.32
Group 9	$5,697.14	$6,606.03	-$908.90
Group 10	$116.05	$304.05	-$188.00
Group 11	$199,696.63	$191,379.68	$8,316.96
Group 12	$454.94	$226.51	$228.43
Group 13	$442.91	$1,771.83	-$1,328.92
Group 14	$1,896.87	$2,974.16	-$1,077.28
Group 15	$57,865.96	$75,279.91	-$17,413.96
Group 16	$25,489.63	$55,477.98	-$29,988.35
Group 17	$27,924.41	$41,310.23	-$13,385.82
Group 18	$50,578.27	$75,601.37	-$25,023.10
Total	$889,536.58	$723,025.95	**$166,510.63**

A Additional Proofs

Properties 2.1

Proof. (a) This follows directly from network flow theory. Also, (8) is clearly feasible (e.g. set all variables to 0), thus the resulting network in the optimal solution will never have a negative cost cycle.

(b) For $j \in B$, π_j is the dual variable of the corresponding constraint in (11). Thus, the optimal π_j equals the shortest path from node j to node 0 in the digraph of Figure 1. Since all of the arc costs are integral, the shortest path and thus the optimal price must also be integral. For $j \notin B$, we can set $\pi_j = \max_i R_{ij}$ which is integral.

(c) In the optimal shortest-path tree, if node/product k is the predecessor of node 0, then clearly, $\pi_k = \gamma_k = \min_{iinC_k} R_{ik}$.

(d) The optimal π_j can be found from the Bellman equation $\pi_j = \min\{\gamma_j, \min_{k\neq j}\{r_{jk} + \pi_k\}\}$. Given $i^* \in C_1$ and $R_{i^*1} \leq R_{i1}, \forall i$, $\pi_1 = \min\{R_{i^*1} - \delta_{i^*}, \quad \min_{k\neq 1}\{r_{ik} + \pi_k\}\}$. Let $k_0 = \arg\min_{k\neq 1}\{r_{ik} + \pi_k\}$ and the corresponding shortest path from k_0 to 0 follow the nodes $k_0, k_1, \ldots, k_\ell, 0$ in that order. Also, let $i' = \arg\min_{i\in C_{k_\ell}}\{R_{ik_\ell} - \delta_i\}$.

Suppose for the sake of contradiction that $R_{i^*1} - \delta_{i^*} > r_{1k_0} + \pi_{k_0}$. Subtracting $R_{i'1}$ from both sides gives us

$$\begin{aligned} R_{i^*1} - \delta_{i^*} - R_{i'1} > r_{1k_0} + \pi_{k_0} - R_{i'1} &= r_{1k_0} + \sum_{j=1}^{\ell} r_{k_{j-1}k_j} + (R_{i'k_\ell} - \delta_{i'}) - R_{i'1} \\ &\geq r_{1k_0} + \sum_{j=1}^{\ell} r_{k_{j-1}k_j} + r_{k_\ell 1} \geq 0. \end{aligned}$$

We get the first inequality on the second line from the definition of $r_{k_\ell 1}$ and since $i' \in C_{k_\ell}$. The last inequality is due to the lack of negative cost cycles. This leads to $R_{i^*1} > R_{i'1} + \delta_{i^*}$, which is a contradiction.

□

Lemma 2.1

Proof. We show that the following is an optimal solution for the LP relaxation of (8): $\pi^*_{j_i} = R_{ij_i}, \theta^*_{ij_i} = 1, \theta^*_{ik} = 0, p^*_{ij_i} = \pi_{j_i}, p^*_{ik} = 0, \quad \forall i \in \{1, \ldots, n\}, \forall k \neq j_i, \quad \pi^*_k = \max_i R_{ik}, \quad \forall k \neq j_i, \forall i \in \{1, \ldots, n\}$.

In fact the above solution is feasible for (8) and, for any feasible solution $p_{ij}, \theta_{ij}, \pi_j$ for the LP relaxation of (8),

$$\sum_i N_i \sum_j p_{ij} \leq \sum_i N_i \sum_j R_{ij}\theta_{ij} \leq \sum_i N_i R_{ij_i} \left(\sum_j \theta_{ij}\right) \leq \sum_i N_i R_{ij_i} = \sum_i N_i \sum_j p^*_{ij},$$

where the first inequality follows from the constraints $R_{ij}\theta_{ij} - p_{ij} \geq 0$, $\forall i, \forall j$, in (8), the second inequality follows from the definition of j_i, while the third inequality follows from $\sum_j \theta_{ij} \leq 1$ for every j. □

Lemma 2.2

Proof. Given C_j^*, let $\theta_{ij}^* = 1$ if $i \in C_j^*$ and $\theta_{ij}^* = 0$ otherwise, let $\pi^* = \min_{i \in C_j^*} R_{ij}$ and $p_{ij}^* = \pi_j^* \theta_{ij}^*$. We will show that θ_{ij}^*, p_{ij}^* and θ_{ij}^* is feasible for (8) for δ sufficiently small since

$$R_{ij} - \pi_j^* \geq R_{ik} - \pi_k^* + \delta_i, \quad \text{for } i \in C_j^*, \forall k \neq j.$$

$R_{ij} - \pi_j^*$ is nonnegative since $i \in C_j^*$, and $R_{ik} - \pi_k^* = R_{ik} - \min_{l \in C_k^*} R_{lk} < 0$ since $i \notin C_k^*$ from the no overlap condition. We need to make sure that $\delta_i \leq \min_{l \in C_k^*} R_{lk} - R_{ik}$, $\forall k$ so that $R_{ik} - \pi_k^* + \delta_i \leq 0$. Thus, $\delta_i \leq \min_{j=1,\ldots,m}\{\min_{l \in C_j^*} R_{lj} - R_{ij}\}$. The other constraints of (8) are satisfied trivially.

Let $p_{ij}, \theta_{ij}, \pi_j, \quad i, j$, be any feasible solution to the MIP (8). Then

$$\sum_j \left(\sum_i N_i p_{ij} \right) \leq \sum_j \left(\sum_i N_i p_{ij}^* \right)$$

since, for every j, p_{ij}^*, θ_{ij}^*, $\forall i$, π_j^* define an optimal solution for the problem with only product j. Therefore $\pi_j^*, p_{ij}^*, \theta_{ij}^*$ is optimal for (8). □

Acknowledgments: We thank Maurice Cheung for a preliminary Java implementation for CPLEX usage and Zhengzheng Zhou for her work during the very early stages of this research project.

References

[1] F. Barahona and É. Tardos. Note on weintraub's minimum cost circulation algorithm. *SIAM J. Comput.*, 18:579–583, 1989.

[2] G. Dobson and S. Kalish. Positioning and pricing a product line. *Marketing Science*, 7:107–125, 1988.

[3] G. Dobson and S. Kalish. Heuristics for pricing and positioning a product-line using conjoint and cost data. *Management Science*, 39:160–175, 1993.

[4] Z Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1999.

[5] V. Guruswami, J. Hartline, A. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. *SODA*, 2005.

[6] W. Hanson and R.K. Martin. Optimal bundle pricing. *Management Science*, 36:155–174, 1990.

[7] ILOG. *CPLEX 9.1 User Manual*.

[8] S. Kalish and P. Nelson. A comparison of ranking, rating and reservation price measurement in conjoint analysis. *Marketing Letters*, pages 327–335, 1991.

[9] U.G. Kraus and C.A. Yano. Product line selection and pricing under a share-of-surplus choice model. *European Journal of Operational Research*, 150:653–671, 2003.

[10] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optim.*, 1:166–190, 1991.

[11] M.W. Padberg and L.A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 1985.

[12] R. Shioda, L. Tunçel, and B. Hui. Probabilistic choice models for product pricing based on reservation prices. *Research Report, Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo*, CORR 2007-02, 2007.

[13] G. Van Ryzin and K.T. Talluri. *The Theory and Practice of Revenue Management*. Kluwer Academic Publishers, 2004.

[14] A. Weintraub. A primal algorithm to solve network flow problems with convex costs. *Management Science*, 21:87–97, 1974.