Efficient Heuristic Algorithms
for
Maximum Utility Product Pricing Problems

T. G. J. Myklebust[*]   M. A. Sharpe[†]   L. Tunçel[‡]

November 2012, revised: November 2015

Abstract

We propose improvements to some of the best heuristic algorithms for optimal product pricing problem originally designed by Dobson and Kalish in the late 1980's and in the early 1990's. Our improvements are based on a detailed study of a fundamental decoupling structure of the underlying mixed integer programming (MIP) problem and on incorporating more recent ideas, some from the theoretical computer science literature, for closely related problems. We provide very efficient implementations of the algorithms of Dobson and Kalish as well as our new algorithms. We show that our improvements lead to algorithms which generate solutions with better objective values and that are more robust in overall performance. Our computational experiments indicate that our implementation of Dobson-Kalish heuristics and our new algorithms can provide good solutions for the underlying MIP problems where the typical LP relaxations would have more than a trillion variables and more than three trillion constraints.

Keywords: mixed integer programming, network flow problems, total unimodularity, optimal product pricing, algorithms

AMS Subject Classification: 90C11, 91B25, 90C27, 90C35, 68C05, 05C85

[*]Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada (e-mail: *tmyklebu@csclub.uwaterloo.ca*). Research of this author was supported in part by a Tutte Scholarship, and Discovery Grants from NSERC.

[†]Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada (e-mail: *masharpe@uwaterloo.ca*). Research of this author was supported in part by a Summer Undergraduate Research Assistantship Award from NSERC (Summer 2009) and a Discovery grant from NSERC.

[‡](**Corresponding Author**):Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada (e-mail: *ltuncel@uwaterloo.ca*). **Tel: 1 (519) 888-4567 ext.35598, Fax: 1 (519) 725-5441**. Research of this author was supported in part by Discovery Grants from NSERC and by research grants from University of Waterloo.

1   Introduction

We study a class of optimal product pricing problems in which the customer demands, budgets and preferences are encoded by *reservation prices* — the highest price that a customer is willing and able to pay for a given product. See, for instance, [18, 19, 20]. Essentially every company operating in a free market environment faces a variant of this problem. Therefore, the optimal pricing problem in revenue management is widespread. A very fundamental version of the problem is based on the assumption that the customers will buy the product that maximizes their individual surplus (or utility), where the surplus is defined as the difference between the reservation price of the customer for the product and the price of the product. In some similar contexts, this model can also be used for *envy-free pricing*; see the related models in [1, 3, 11]. Maximum utility pricing models are also related to bilevel pricing problems. For the latter, see, for instance, [15] and the references therein. The problem considered there is equivalent to maximum utility product pricing problem, see [16]

From a computational complexity viewpoint, the problem is $\mathcal{NP}$-hard, see [9, 3]; it is $\mathcal{NP}$-hard even to approximate within a reasonable worst-case ratio, see [11] (even though some special cases of related optimal pricing problems admit efficient algorithms, see for instance [12]). Currently, the best heuristic algorithms for solving these problems are the ones proposed by Dobson and Kalish, see [8, 9]. While these algorithms are successful on many instances of the problem, quite often they are not.

We present very efficient implementations of Dobson-Kalish heuristic algorithms for optimal pricing problems [8, 9]. Then, we consider further improvements based on a decoupling property of the problem with respect to continuous and binary variables. Our improvements are based on a detailed study of a fundamental decoupling structure of the underlying mixed integer programming (MIP) problem and on incorporating some more recent ideas for closely related problems from Computer Science, Operations Research and Optimization. We provide very efficient implementations of our new algorithms; efficiency, in particular the speed of the algorithms is critical in practical applications of our framework. A main reason is our stationary price assumption for the competitors' products. Indeed, in practice the competitors may adjust their prices, as a result, we should be able to re-optimize our prices and respond very quickly. Another main reason for the required speed comes from the sizes of the optimization problems in nontrivial applications. For the sake of presentation, we make simplifying assumptions about the optimal pricing problem while constructing the mathematical model. Indeed, in applications, we can relax most of the assumptions required by our mathematical models. However, to do so, we introduce new variables to the problem. For example, the mathematical model requires that every customer segment pay the same price for the same product. Suppose that in some application, we would like to offer different prices to customers who buy their plane tickets four to six weeks in advance of their trip than others. In this case, we simply create a new product in our mathematical model to represent these plane tickets. Another instance arises in product bundling. For each bundle of products, we create a new variable which represents the bundle. These transformations when properly applied can lead to tens of thousands of "product variables" in the mathematical model (which in turn lead to LP relaxations with a huge number of variables and constraints).

We show that our improvements yield algorithms which generate solutions with better objective values and that are more robust in overall performance. Our computational experiments indicate that our implementation of Dobson-Kalish heuristics and our new algorithms can pro-

vide good solutions for the underlying MIP problems where the typical LP relaxations would have more than a trillion variables and more than three trillion constraints. In the next section, we begin the development of the necessary notation and the description of some of the fundamental mathematical models we utilize. Then, at the end of next section, we describe the material in the remaining sections.

2   Notation and Fundamental Ingredients of the Heuristics

Suppose we have n customer segments, with each segment homogeneous (i.e., customers within a segment behave similarly), and m products. R_{ij} denotes the reservation price of customer segment i for product j. Let our decision variables be as follows:

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

$$\pi_j := \text{price of product } j.$$

Customers strive to maximize their utility which in our case corresponds to *consumer surplus* given by $(R_{ij} - \pi_j)$ for customer i and product j. For a fixed customer segment i, if the consumer surplus is negative for every product then customer segment i would buy nothing. Therefore, customers buy among all product with nonnegative consumer surplus, the one with the largest surplus. We are modeling the problem from the viewpoint of one fixed company with products $1, 2, \ldots, m$. To represent the competing companies' products, we assume that the prices of their competing products are known to us. We denote by CS_i (which is part of the data for our optimization problem), the maximum surplus for customer segment i across all competitor products. Then the constraints that model the buying behaviour are

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

and

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

respectively. Note that we may replace R_{ij} by $\max\{R_{ij} - CS_i, 0\}$ for every i, j and assume without loss of generality that $CS_i = 0$ for every i.

Assuming that each customer segment buys at most one type of product, and each customer buys at most one unit of a product, and denoting by N_i the number of customers in segment i, the problem can be expressed as the following nonlinear mixed-integer programming problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} N_i \pi_j \theta_{ij}, \tag{1}$$

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

Note that the second group of constraints (arising from the elimination of CS_i) are very similar to the first group of constraints. Indeed, the second group of constraints can be removed from the formulation by introduction of a dummy product indexed zero, with price set equal to zero. We will see this again shortly when we expose the underlying network structure in the above formulation.

The assumptions stipulated above are not very restrictive in practice as there are ways of relaxing them by modifying the interpretation of the variables etc.; for example, production costs can be incorporated in the objective function without destroying the important mathematical structures of (1). Some of the classical work in the area explicitly included fixed costs. Adding production costs (fixed and/or variable unit costs) to our model does not change the applicability of our algorithms. This is due to the fact that in making assignment decisions (assigning a customer segment to a product), our algorithms evaluate the whole objective function for that assignment/reassignment. As a result, feasible solutions of the problem (1) will be correctly evaluated if we change the objective function of the problem allowing fixed costs. Moreover, the transformations we made in relation to CS_i are indeed still valid in this case.

The assumption about the competitors' prices being stationary and known to us, can also be handled in practice, as long as we are able to reoptimize our prices very quickly (for details, see [20], [19] and [21]). This provides more motivation for having very fast and effective heuristic algorithms with good re-optimization properties. As we mentioned in the Introduction, our model is closely related to *envy-free pricing* models. Considering Walrasian Equilibrium Models (going back at least to Walras in 1874, see [22]), equilibrium prices should be chosen so that

(i) customers have no incentive to change their decision;

(ii) the company has no incentive to change its prices.

At an optimal solution of our problem (1), item (i) is satisfied due to the fact that every customer who buys a product, buys a product for which the customer has the maximum possible positive surplus. Item (ii) is satisfied, due to the fact that the total revenue/profit for the company is maximized and no further change to the prices can increase the current revenue/proift.

For a fixed $\theta \in \{0,1\}^{n \times m}$ such that $\sum_{j=1}^{m} \theta_{ij} \leq 1$, $\forall i$, the problem (1) becomes an LP problem (on variables π) with a *totally unimodular* (TUM) coefficient matrix. The dual of this LP is equivalent to a shortest path problem, where there are $(m+1)$ nodes (a node for each product and a dummy product). Let 0 denote the dummy product node. Assign arc lengths:

$$r_{kj} := \min_{i \in C_j} \{R_{ij} - R_{ik}\},$$

$$r_{0j} := \min_{i \in C_j} \{R_{ij}\},$$

where $C_j := \{i : \theta_{ij} = 1\}$ and $B := \{j : C_j \neq \emptyset\}$. Note that r_{kj} for $k \in \{1, 2, \ldots, m\}$ may be negative and we utilize the usual convention that the minimum over an empty set is $+\infty$. Then, this dual LP problem can be solved by finding shortest paths from node 0 to every node $j \in B$. The lengths of these shortest paths provide optimal π_j values. For the details, see [8, 20].

Dobson and Kalish gave two heuristic algorithms for the above problem that have stood the test of time. The first one is a reassignment heuristic which we refer to as DK88. DK88 takes as input a feasible assignment θ and constructs a shortest path tree, as described above, rooted at the dummy node 0. Then for each customer segment that determines the cost on an arc of the

shortest path tree, the algorithm considers assigning that customer segment to the parent of the current node in the shortest path tree. Among all such reassignments, DK88 picks the one with the largest improvement. If the improvement is positive, the algorithm modifies the assignment variables θ, updates the arc lengths and the shortest path tree, and repeats; otherwise, the algorithm terminates and reports the current θ and π.

Another heuristic algorithm by Dobson and Kalish introduces products one at a time with initial prices at $+\infty$. For each product, the algorithm computes the maximum possible price that a customer segment can be charged while keeping the prices of all other products fixed at their current levels. Then, the algorithm computes the total profit for setting the price of the current product to one of the n possible values just computed. The algorithm loops through these steps until no improvement is obtained. We refer to this algorithm as DK93.

We provide a state-of-the-art implementation of the algorithm DK88 that we call Fast-DK. Then, considering the ideas of DK93, we design a new version that we call Global-DK. We further exploit the structure of the mixed integer nonlinear programming problem and the basic decoupling as exposed by Dobson and Kalish [8], as well as the ideas offered by Dobson and Kalish [9], Shioda et al. [20] and Guruswami et al. [11]. In addition to providing very efficient implementations of DK88 and Global-DK, our work leads to five fundamental improvements. Three of the improvements are directly related to our generalization (Generalized Reassignment Heuristic GRH, as described in the next section) of the algorithms of Dobson and Kalish. By decoupling the problem into its combinatorial and continuous parts, we are able to apply continuous optimization techniques in the price space. In many continuous optimization algorithms, in each iteration, we pick a *search direction*, a direction along which to change the current solution; and once the search direction is chosen, we then decide on the *step size* (how far to move along the search direction). Then, we have the next iterate.

- Once a search direction (a direction along which the prices are being changed) is picked, instead of looking for a first change in the customers' choices for products to buy, in some of our new algorithms, we do a full line search, allowing the *step size* to take any value in $(-\infty, +\infty)$ and picking the price that *globally* maximizes the objective function along this line.

- In an extension of the algorithm DK88, using the fact that when a customer segment is assigned to its parent, only the children of the new node may change in the new shortest path tree, provided no price is decreased, we consider increasing the prices of all products for each of the subtrees of the node (product) being considered. We call this new algorithm GRH-SubTree.

- Through our study of geometric, algebraic and combinatorial properties of the problem, we find that the search directions corresponding to picking two products and increasing the price of one while decreasing the price of the other at the same rate may yield to better solutions. We utilize these search directions in addition to the conventional search directions. We use these ideas in one of our featured algorithms called Cell-Pierce.

The remaining fundamental improvements are on initialization of the reassignment algorithms mentioned above. The most recent computational work reported in [20] used a greedy initialization heuristic called MaxR, described in the next section. However, Guruswami et al. [11] showed that setting all prices equal to a common value and optimizing this common value

gives another simple algorithm which has a provable performance guarantee (even though a very weak one from an applications viewpoint).

- We utilize the basic idea of the Guruswami et al. heuristic and call this algorithm `Guru`. We observe that `Guru` does perform better than `MaxR` under many circumstances.

- Exploiting the decoupling nature of the "assignment" part and "given an assignment, finding the optimal prices for that assignment" part of the problem, we define a simple iterated optimization procedure. Finding a fixed point of these iterated operators is computationally easy. We apply the iterated operators to the output of `Guru`. The resulting algorithm is called $\overline{\text{Guru}}$ which provides improvements over `Guru`, as verified by our computational experiments.

In addition to the above-mentioned contributions, we also present some theoretical foundations for our algorithmic ideas and some theoretical connections to related problems. In particular, after providing more detailed descriptions of our new algorithms in Section 3, in Section 4 we derive properties of the ingredients of our algorithms based on the decoupling structure in the MIP. This decoupling structure and the underlying heuristic algorithms, motivate the need for better understanding of the polyhedral subdivisions induced by them. We conclude Section 4 by analyzing the polyhedral subdivision induced by the MIP in the price space. The exchange results established for these polyhedra further motivate a connection to another optimal pricing problem called the *Stackelberg Network Pricing Game* in Subsection 4.3. In Section 5, this time motivated by Guruswami et al. [11] results, we explore theoretical worst-case approximation guarantees. We report the results of our computational experiments as well as a new lifted LP relaxation of the MIP in Section 6. We conclude with brief final remarks in Section 7.

3 Generalized Reassignment Heuristic and Related Algorithms

As in Shioda et al. [20], we will utilize the *Maximum Reservation Price Heuristic* (`MaxR`) as an initialization heuristic for `Fast-DK` as well as `Global-DK`:

Maximum Reservation Price Heuristic (`MaxR`)

1: Set $C_j = \{i : j = \arg\max_k\{R_{ik}\}\}, \forall j$. I.e., assign each customer to her/his maximum-utility product.

Based on our assumptions, every $\pi \in \mathbb{R}_+^m$ defines a corresponding product-customer assignment $\theta \in \{0,1\}^{n \times m}$. As we did above, we use two equivalent representations of such product-customer assignments, the other being the collection of subsets $\mathcal{C} := \{C_1, C_2, \ldots, C_m\}$ (with C_j denoting the set of customers who buy product j under the current assignment). Depending on the context, we will use θ and $\mathcal{C}$ interchangeably, to refer to the same product-customer assignment. We call $\theta \in \{0,1\}^{n \times m}$ a *feasible assignment* if there exists $\pi \in \mathbb{R}_+^m$ such that (θ, π) is a feasible solution of the nonlinear MIP (1). Note that even though every nonnegative price vector π yields a feasible assignment, not every $\{0,1\}^{n \times m}$ is a feasible assignment. We will also use the notation $\mathcal{C}$ which represents an assignment as a function which maps a given price vector π to the corresponding product-customer assignment. It may be that there are many optimal assignments for given prices. To disambiguate, we require that $\mathcal{C}$ always returns the

assignment where each customer, when indifferent between two products and both products have the same surplus, chooses the product with least index. This can be phrased as taking the lexicographically-least optimal assignment. Even though lexico.-least approach seems arbitrary, there is a notion called a *price ladder structure* in the area of revenue management; see, for example, [1] and [18]. Indexing the products with respect to some loose notion of price ladder may make sense in practice too. Let us denote such an assignment by $\mathcal{C}(\pi)$. Then, by construction, the assignment $\mathcal{C}(\pi)$ is feasible. Therefore, our algorithms based on this function never encounter infeasible assignments θ. Also note that in addition to the `MaxR` heuristic, *any* heuristic which delivers some $\pi \in \mathbb{R}_+^m$ can be used to initialize our algorithms via the computation of $\mathcal{C}(\pi)$.

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 auxiliary graph of products. 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 a high enough price that does not change $\mathcal{C}$. That is, we need

$$\pi_u > \min_i \left\{ R_{iu} - R_{ij_i} + \pi_{j_i} \right\},$$

where j_i denotes the product j such that $i \in C_j$. Note that even if $C_i \neq \emptyset$, for every i, it is rarely the case that for an arbitrary $\pi \in \mathbb{R}_+^m$, $\Pi(\mathcal{C}(\pi)) = \pi$.

3.1 `Fast-DK`: An Efficient Implementation of the Dobson-Kalish 1988 Heuristic

Given an assignment θ of customers to products, let the "pricing graph" be the directed graph on all products together with the null product where there is an arc from product k to product j with weight

$$\min_{i:\theta_{ij}=1} \left\{ R_{ij} - R_{ik} \right\}.$$

The weights on the arcs emanating from the null product are also computed by the same formula (with the convention that $R_{i0} := 0$). Applying operator Π to the assignment θ simply amounts to building this graph and computing single-source shortest paths taking the null product as the root.

In a `DK88` iteration, we have a shortest-path tree for the current assignment of customers to products and we try, for each product, reassigning the customer who constrains the price the most to the parent product. This reassignment produces, in each case, a very closely related pricing graph — only the arcs incident with the parent and child products can be changed. Thus, the adjacency matrices of the pricing graphs of the original and reassigned problems differ in at most two rows and two columns.

This leads to a trick for computing the pricing graphs of the reassigned subproblems quickly. We store the original pricing graph in adjacency matrix form as an array of length m of pointers to arrays of length m. (The kth entry of the array pointed to by the jth pointer is the smallest value of $R_{ij} - R_{ik}$ among all customer segments assigned to product j.) Note that, if customer segment i is reassigned from product j to product k, then all edge weights in this graph are unchanged except those involving products j and k.

Thus, we can represent a reassigned subproblem as a copy of the original graph with two pointers changed together with a list of two columns whose contents have been replaced. The original matrix is not mutated; the identities and contents of these columns are stored separately

from the original matrix. Looking up the weight of the arc from j to k is done by checking whether k is one of the affected columns, using its replacement if so, or using the kth entry of the array pointed to by the jth pointer if not.

When recomputing shortest paths after reassigning customer i from product j to product k, we set shortest path lengths for all descendants of k to infinity and then run Bellman-Ford-Moore algorithm starting at k, taking care when visiting a node first to relax along all *inbound* arcs.

3.2 Generalized Framework for Heuristic Algorithms

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 the `MaxR` 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 the underlying shortest path problems, including prices π and the corresponding objective function value z.

1: **repeat**
2:    For each $d \in \mathcal{D}_\pi$ compute $\Pi\left(\mathcal{C}(\pi + \alpha_d d)\right)$, and the corresponding objective value z_d.
3:    **if** $\max_{d \in \mathcal{D}_\pi} \{z_d\} \leq z$, **then**
4:       STOP and return π and the corresponding assignment $\mathcal{C}(\pi)$.
5:    **end if**
6:    Let $z := \max_{d \in \mathcal{D}_\pi} \{z_d\}$, $\bar{d} := \mathrm{argmax}_{d \in \mathcal{D}_\pi} \{z_d\}$, $\pi := \Pi\left(\mathcal{C}(\pi + \alpha_{\bar{d}}\bar{d})\right)$.
7: **until** Maximum number of iterations.

Note that the above algorithm can increase the prices of an arbitrary number of products at arbitrary relative rates and simultaneously decrease the prices of an arbitrary number of products at arbitrary, relative rates. Moreover, the set of search directions may varied based on the current price vector π. Next remark highlights how general our proposed framework is compared to the Dobson-Kalish reassignment heuristic, `DK88`.

Remark 1. *The Dobson-Kalish reassignment heuristic,* `DK88`*, is a special case of the above algorithm, where*

$$\mathcal{D}_\pi := \{e_j : j \in B\} \bigcup \{0\},$$

independent of π, with e_i denoting the ith unit vector and 0 is included in $\mathcal{D}_\pi$ to represent the status quo. α_d would be determined by the first change in $\mathcal{C}$ along this price change direction d.

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 and/or $M_j := \sum_{i \in C_j} N_i$, total number of customers who are buying product j under the current prices) 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 `DK88` 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 GRH over the `DK88` heuristic is that GRH can handle some non-trivial modifications to the model. For example, in practice, one of the drawbacks of our current model is that the model effectively assumes *if there is a tie for the largest positive surplus, then the customer segment will buy the most expensive product.* Shioda et al. [20] proposed the usage of a *utility tolerance* which forces the model to only assign the customer segment i to the product j when the corresponding surplus is at least a predetermined amount δ larger than for all other products. Indeed, this tolerance may differ for every pair j, k of products, in addition to varying over the customer segments (we may denote the tolerance by δ_{ijk}). Note that GRH is driven by the prices π_j rather than the assignments θ_{ij}. Consequently, GRH also easily handles the various models based on nonzero utility tolerances δ, δ_i, δ_{ijk}.

3.2.1   Line Search and the algorithm `Global-DK`

Given a direction d, we may in $O(mn \log n)$ time perform a full line search to compute the best objective function value for potential prices

$$\pi + \alpha d, \text{ where } \alpha \in \mathbb{R}.$$

Note that we allow negative as well as positive values for α.

The key idea here is that each customer's utility function (when considered as a function of α), is convex and piecewise-linear, being defined by the maximum of m affine functions of α. This is depicted in Figure 1. Thus, one can represent the epigraph of a customer's utility function as the intersection of m halfspaces; this intersection may be computed using duality and a convex hull algorithm. If we use Graham's scan (see for example Chapter 3 of [17]) to compute this convex hull, notice that the same ordering, namely by component of d, works for all customers; sorting the products in order once, suffices for all customers. So, we can compute each customer's utility function in $O(m \log m + mn)$ time.

The revenue function may be considered as the sum over all customers of the revenue gained from each customer. The revenue gained from a customer is quite strongly related to the customer's utility function; both are piecewise-linear functions and they "break" at the same abscissae. Compare Figures 1 and 2.

In $O(mn \log n)$ time, we can compute the maximum value of a sum of n discontinuous but piecewise-linear functions and its abscissa (which is closely related to the fundamental problem of computing the convex hull, in computational geometry). We do this with a sweep-line algorithm, sweeping from left to right and adjusting the value and its slope at each "break" point. Care

must be taken when programming this line search procedure to handle degenerate cases, such as when three lines meet at a point.
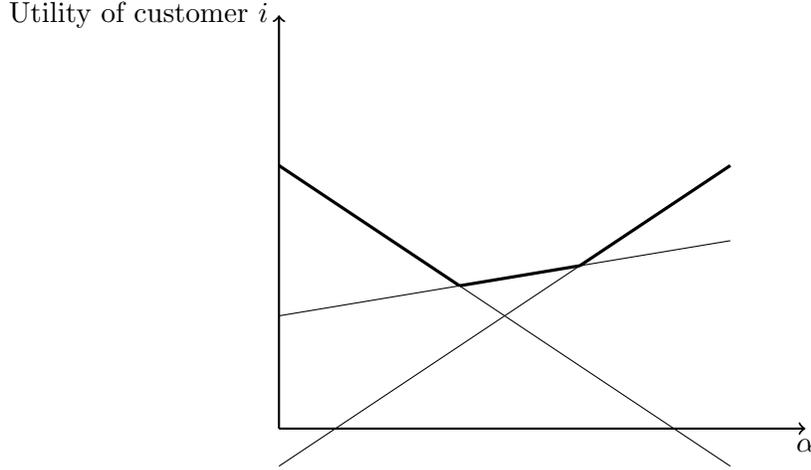
Figure 1: A customer's utility as a function of α (as the price vector varies with $\pi + \alpha d$).

The heuristic `Global-DK` lies in the framework of GRH and uses directions in the π-space that only increase or decrease prices one at a time:

$$\mathcal{D}_\pi := \{\pm e_j : j \in \{1, 2, \ldots, m\}\} \bigcup \{0\}.$$

Next, we turn to the problem of finding promising search directions other than $\pm e_i$ (i.e., other than increasing or decreasing the price of a single product at a time).

3.2.2 Algorithm `Cell-Pierce`

Every facet of every cell of the decomposition of π-space by customer assignment (θ) is defined either by an inequality of the form $\pi_j - \pi_k \leq R_{ij} - R_{ik}$, or $\pi_j \leq R_{ij}$ or by an inequality of the form $\pi_j \geq 0$. Therefore, when we focus on inequalities generated from a fixed customer, we would see a rectangular prism in $\mathbb{R}_+^m$ which is cut by half spaces $\pi_j - \pi_k \leq R_{ij} - R_{ik}$, some of which will correspond to parallel halfspaces. See Figure 3, where we took two products ($m := 2$) and three customer segments ($n := 3$), and demonstrated the rectangular prisms and some cuts of the type $\pi_j - \pi_k \leq R_{ij} - R_{ik}$. Note that while Figure 3 is informative, it can be quite misleading in higher dimensions (e.g., no two of the hyperplanes $\{\pi \in \mathbb{R}^5 : \pi_1 - \pi_2 = 10\}$, $\{\pi \in \mathbb{R}^5 : \pi_2 - \pi_3 = 20\}$, $\{\pi \in \mathbb{R}^5 : \pi_3 - \pi_4 = 30\}$, $\{\pi \in \mathbb{R}^5 : \pi_4 - \pi_5 = -10\}$ are parallel in $\mathbb{R}^5$). However, we know the normals to all hyperplanes of all facets of all cells. Thus, if one was so inclined, one could find a point in the relative interior of each facet of the current cell and "see what is on the other side." This observation motivates the search directions $\{e_j \pm e_k : 1 \leq j \leq m, 1 \leq k \leq m\}$ in the context of GRH.

A subset of these search directions with the full line search as described above are used in our algorithm called `Cell-Pierce` (we named the algorithm based on the hope that if we move along the directions that are normals to the facets of the cells and do a full line search $\alpha \in (-\infty, +\infty)$, we have a higher chance of "piercing" through the interiors of full dimensional cells an not get

Revenue from customer i
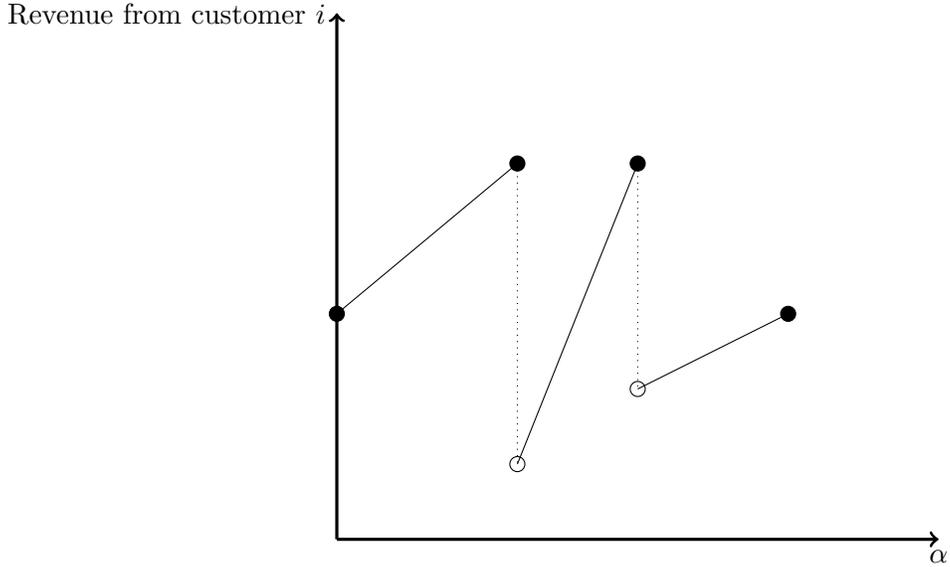
Figure 2: Revenue from the customer of Figure 1 as a function of α (as the price vector varies with $\pi + \alpha d$).

stuck on low dimensional faces of cells prematurely). We chose only the directions $e_j - e_k$ such that $\pi_j - \pi_k = R_{ij} - R_{ik}$ at the current point for some i and made no attempt at finding the relative interior of each appropriate face.

4   Algorithm `GRH-SubTree`, and Structural Results

The feasible assignments θ partition π-space so that each part is a TUM polyhedron. We call such a polyhedron defined by a such a θ the *polyhedron induced by θ*. Note that moving along the edges of such polyhedra correspond to moving along $d \in \{-1, 0, +1\}^m$. (I.e., prices are increased and/or decreased at the same rate.)

First, we note that when the price of a product is increased just enough that it causes a change in the assignment θ, say a customer segment gets assigned to product j, the optimal prices for this new assignment can be computed by simply updating the shortest path tree for the children of the node j. Motivated by this observation (which is founded on Lemma 3.3 of [20]), we propose an extension of `DK88` which we call `GRH-SubTree`. The new algorithm `GRH-SubTree` is a special case of `GRH` which in addition to considering directions e_j, considers increasing the prices for a product and all its children (in the current shortest path tree) at the same rate.

Next, we analyze repeated applications of operators $\mathcal{C}$ and Π which map prices to corresponding feasible assignments, and feasible assignments to optimal prices, respectively.

4.1   Fixed Points of $\Pi(\mathcal{C})$

Now, it is useful to consider the properties of fixed points of $\Pi(\mathcal{C})$, i.e., those π such that $\Pi(\mathcal{C}(\pi)) = \pi$. Later in Section 6, we present computational experiments with an algorithm of
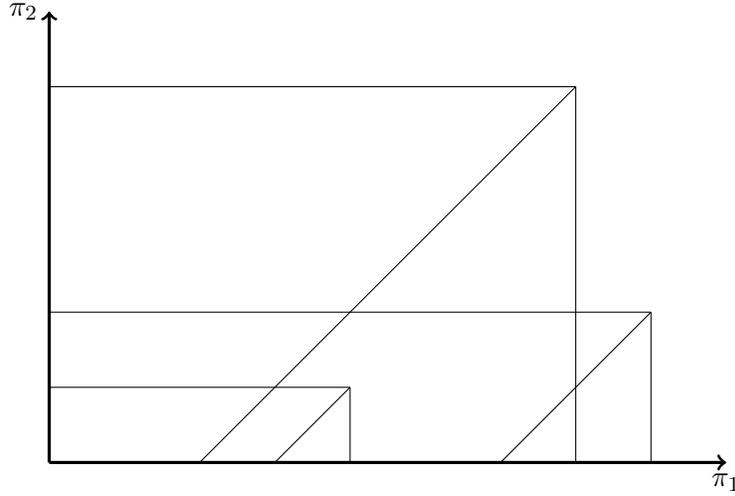
Figure 3: An illustration of some of the constraints in the subdivision of the price space $\mathbb{R}^m$ by feasible product-customer assignments

Guruswami et al. [11] which we call `Guru` and our enhancement of it based on finding fixed points of $\Pi(\mathcal{C})$. Our algorithm $\overline{\texttt{Guru}}$ is simply Algorithm `Guru` followed by iterating the operators Π and $\mathcal{C}$ until a fixed point is found. We use our enhancement in other heuristic algorithms as well.

Definition 4.1. A price vector π is called a *fixed point* if $\Pi(\mathcal{C}(\pi)) = \pi$. An assignment θ is called a *fixed point* if $\mathcal{C}(\Pi(\theta)) = \theta$. A pair (π, θ) is called a *fixed point* if $\Pi(\theta) = \pi$ and $\mathcal{C}(\pi) = \theta$.

The following proposition justifies the terminology.

Proposition 2. *The following are equivalent:*

1. (π, θ) *is a fixed point;*

2. π *is a fixed point and θ is defined by $\theta := \mathcal{C}(\pi)$;*

3. θ *is a fixed point and π is defined by $\pi := \Pi(\theta)$.*

Recall that once we fix θ, our MIP formulation has only price variable π and in these variables, the problem has a feasible region (which is empty iff θ is infeasible iff its dual which is a shortest path problem has a negative-cost cycle in its network) every π in this feasible region is called *feasible* for that given θ.

Proposition 3. *Given π feasible for θ, the following are equivalent:*

1. $\theta = \mathcal{C}(\pi)$;

2. *in the network defined by θ, all arcs having zero reduced cost with respect to π have non-negative cost, and θ is lexicographically least under this restriction.*

Note that our statement and claim are algorithm dependent. I.e., they depend on the detailed definitions of the maps $\mathcal{C}$ (e.g., lexico.-least rule) and Π. In the upcoming material we will use terminology of Linear Programming duality, e.g., *reduced cost* (see, for instance, [10]). Further note that in many of the proofs, we do not need to treat the null product separately (it suffices to recall that for every feasible price π, $\pi_0 = 0$, moreover, by definition, $R_{i0} := 0$, for every $i \in \{1, 2, \ldots, n\}$).

Proof. Let π be feasible for some assignment θ. First, suppose $\theta = \mathcal{C}(\pi)$. Then by definition, the arcs (j, k) that have zero reduced cost (in the network defined by θ) satisfy:

$$0 = r_{jk} - \pi_j + \pi_k = \min_{i \in C_j} \{R_{ij} - R_{ik}\} - \pi_j + \pi_k = (R_{i^*j} - \pi_j) - (R_{i^*k} - \pi_k),$$

where i^* is a customer segment that attains the minimum above. Since customer i^* bought product j rather than k (under θ') and both products have the same surplus,

- either $\pi_j > \pi_k$, or

- $\pi_j = \pi_k$ and $j < k$.

In either case, $r_{jk} = \pi_j - \pi_k \geq 0$. By definition, θ is the lexicographically least assignment.

Next, suppose in the network defined by θ (where π is feasible for θ), all arcs having zero reduced cost with respect to π have nonnegative cost, and θ is lexicographically least under this restriction. Then, for every arc (j, k),

$$0 = r_{jk} - \pi_j + \pi_k \text{ implies } r_{jk} \geq 0.$$

Hence, for every arc (j, k) with zero reduced cost,

$$\min_{i \in C_j} \{R_{ij} - R_{ik}\} \geq 0.$$

Putting all of this together, we deduce that customer i buys product j iff product j has the largest nonnegative surplus (if there is a tie, product j has the largest price among all products with maximum surplus, and there is yet another tie, then product j is the one with the least index), as desired. Therefore, $\theta = \mathcal{C}(\pi)$. $\square$

Next, we prove that we can compute a fixed point in strongly polynomial time. The hypothesis of the next proposition requires a feasible assignment θ; however, notice that *any* $\pi \in \mathbb{R}_+^m$ can be used to find a feasible θ via $\theta := \mathcal{C}(\pi)$, in strongly polynomial time, very efficiently.

Proposition 4. *Given any feasible assignment θ, a fixed point can be obtained in strongly polynomial time with objective value at least as large. In particular, there is an optimal solution that is a fixed point.*

Proof. Repeatedly apply $\mathcal{C}(\Pi)$ until a fixed point is obtained. Observe that whenever applying $\mathcal{C}(\Pi)$ causes a segment i to switch from a product j to a product k, we have

$$R_{ij} - \pi_j = R_{ik} - \pi_k$$

and

$$\pi_k \geq \pi_j \Rightarrow R_{ik} \geq R_{ij},$$

as well as either

$$\pi_k > \pi_j \Rightarrow R_{ik} > R_{ij}$$

or

$$k < j.$$

Thus, if we consider a single segment i and order its products j in a list L by descending order of R_{ij} and breaking ties by ascending order of j, we see that $\mathcal{C}(\Pi)$ only ever reassigns i from j to k if j precedes k in L.

Hence, the number of applications of $\mathcal{C}(\Pi)$ needed to reach a fixed point is bounded by $O(nm)$. Since $\mathcal{C}(\Pi)$ can be computed in strongly polynomial time, we conclude that a fixed point can be computed in strongly polynomial time. $\qquad\square$

Example 5. *It is possible that if an assignment is optimal and some customers are deleted from the problem, the assignment is no longer a fixed point. This can matter in some search strategies. Take $m := 2$, $n := 3$ and*

$$N := [10, 10, 1], R := \begin{bmatrix} 3 & 0 \\ 0 & 1 \\ 3 & 2 \end{bmatrix}.$$

Then,

$$\theta := \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \pi := [3, 1]$$

give a fixed point. Next, consider deleting customer 2, so our assignment becomes $\tilde{\theta} := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, with $C_1 = \{1\}$ and $C_2 = \{3\}$. This leads to the optimal price vector $\tilde{\pi} := [3, 2]$. However, $\tilde{\theta}$ is not a fixed point with $\tilde{\pi}$, since customer 3 will be assigned to product 1 when applying $\mathcal{C}(\tilde{\pi})$, due to the lexico.-least rule.

Proposition 6. *If a given feasible assignment θ is a fixed point, then $\Pi(\theta)$ can be computed using Dijkstra's algorithm.*

Proof. The shortest path tree consists of equality arcs with respect to the computed shortest path distances (by *equality arcs* we mean those arcs (j, k) for which the reduced cost $(r_{jk} - \pi_j + \pi_k)$ is equal to zero, consistent with the terminology of for instance [6]), so all the arcs of the shortest path tree will be nonnegative. Thus, we may remove all negative-cost arcs from the network, apply Dijkstra's algorithm, and use the resulting shortest path tree and shortest path distances. $\qquad\square$

Proposition 7. *If (π, θ) is a fixed point, there is no directed cycle of equality arcs with respect to π.*

Proof. Along any directed path of equality arcs, the product indices must decrease strictly since $\theta = \mathcal{C}(\pi)$. Therefore, a directed cycle cannot exist. $\qquad\square$

Proposition 8. *If a given feasible assignment θ is a fixed point, then the polyhedron induced by θ has dimension m.*

Proof. Let $\pi = \Pi(\theta)$ and let P be the polyhedron induced by θ. By the previous proposition, the equality arcs form a directed acyclic graph G'. Topologically order the non-null products with respect to G' as $j_1, j_2, \ldots, j_m$. Let $\tilde{\pi} = \pi - \epsilon \sum_{k=1}^{m} k e_{j_k}$, where $\epsilon > 0$ is small enough that $\tilde{\pi}$ is still feasible and no new equality arcs form, which exists because of the topological ordering. Observe that no inequalities hold at equality for $\tilde{\pi}$, so the only equality that holds for P is $\pi_0 = 0$. Therefore, $\dim(P) = m$. $\qquad\square$

Example 9. *In general, many applications of the operator $\Pi(\mathcal{C})$ may be needed to reach a fixed point. Consider an instance with n customer segments with $N_i := 1$ for every i, and two products, $m := 2$. Let*

$$R := \begin{bmatrix} n & 0 \\ n & 1 \\ n & 2 \\ \vdots & \vdots \\ n & (n-1) \end{bmatrix}.$$

Consider the initial feasible assignment

$$\theta^{(0)} := \left[e_1, \ \sum_{j=2}^{n} e_j \right].$$

Then,

$$\pi^{(0)} = \Pi(\theta^{(0)}) \quad = \quad [n, 1],$$

$$\theta^{(1)} = \mathcal{C}(\pi^{(0)}) \quad = \quad \left[e_1 + e_2, \ \sum_{j=3}^{n} e_j \right],$$

$$\pi^{(1)} = \Pi(\theta^{(1)}) \quad = \quad [n, 2],$$

$$\theta^{(2)} = \mathcal{C}(\pi^{(1)}) \quad = \quad \left[e_1 + e_2 + e_3, \ \sum_{j=4}^{n} e_j \right],$$

$$\pi^{(2)} = \Pi(\theta^{(2)}) \quad = \quad [n, 3],$$

$$\vdots$$

$$\theta^{(n-1)} = \mathcal{C}(\pi^{(n-2)}) \quad = \quad \left[\sum_{j=1}^{n} e_j, \ 0 \right],$$

$$\pi^{(n-1)} \quad = \quad \Pi(\theta^{(n-1)}) = [n, +\infty].$$

In this example, where $m = 2$, the operator must be applied $\Theta(n)$ times to reach a fixed point.

4.2 Adjacent Polyhedra

As we noted earlier, for each $\theta \in \{0, 1\}^{n \times m}$, we obtain a polyhedron in the π-space from our MIP. Moreover, under this map, the union of all feasible assignments partitions the π-space into polyhedra. Motivated by the fact that the algorithms DK88 and Fast-DK move in the assignment space by a single customer segment being reassigned at a time, in this subsection, we analyze the related structure of this polyhedral subdivision.

Proposition 10 (Simultaneous Feasibility)**.** *Suppose θ is a feasible assignment and that a single reassignment produces another feasible assignment θ'. Then there exists π that is feasible for both θ and θ'.*

Proof. Let G and G' be the networks corresponding to θ and θ' respectively, and take the disjoint union of their arc-sets to obtain a new network H. We need to check that there is no negative-cost cycle in H.

Let i be the customer that switched from product j to product k, between θ and θ'. Suppose there is a negative-cost cycle C in H. Without loss of generality, C is simple (i.e., no node in C is repeated). Since there is no negative-cost cycle in either G or G', C must use an arc that is not in G and an arc that is not in G'. Clearly, both these arcs must be induced by customer i. Indeed, the cycle cannot be made up from just two arcs (j, k) and (k, j), since the cost of the cycle, then would be zero. Without loss of generality, the first arc is from some product j' to j, and the second arc is from some product k' to k. Then we can write $C := j'j, P_1, k'k, P_2$ for some paths P_1 and P_2 that appear in both G and G'.

Let $C_1 := k'j, P_1$ be a cycle in G and $C_2 := j'k, P_2$ be a cycle in G'. Then

$$
\begin{aligned}
c(C) &= c(j'j) + c(P_1) + c(k'k) + c(P_2) \\
&= R_{ij} - R_{ij'} + c(P_1) + R_{ik} - R_{ik'} + c(P_2) \\
&= R_{ij} - R_{ik'} + c(P_1) + R_{ik} - R_{ij'} + c(P_2) \\
&\geq c(k'j) + c(P_1) + c(j'k) + c(P_2) \\
&= c(C_1) + c(C_2).
\end{aligned}
$$

Therefore, either $c(C_1) < 0$ or $c(C_2) < 0$, a contradiction. (Note that in the above, if $k = 0$, i.e., customer i is assigned to the null product, the argument applies the same). $\quad\square$

Proposition 11. *Given feasible assignments θ and θ', θ' can be obtained from θ by performing at most $2n$ reassignments while maintaining feasibility at all times.*

Proof. Since reassignment is symmetric, it is enough to show how to get from θ to the null assignment (all customers assigned to the null product) with at most n reassignments. To do this, repeatedly compute optimal prices for the current assignment, identify a segment that is indifferent between its choice and the null product (note that such a segment always exists, recall the arcs with zero reduced cost that are adjacent to node zero), and assign that segment to the null product. $\quad\square$

The above proposition justifies a search by making single reassignments.

Proposition 12. *All feasible single reassignments can be found in $O(nm^2)$ time.*

Proof. First, compute all-pairs shortest paths in $O(nm^2)$ time. Now, for each customer i and each product j, determine whether it is possible to reassign i to j. To do this, note that by simultaneous feasibility, it is enough to check that there is no negative-cost cycle when the new in-arcs of product j are introduced. Therefore, the reassignment is feasible if and only if for all products k,

$$
\mathrm{dist}(j, k) + R_{ij} - R_{ik} \geq 0.
$$

The total time to make this check for every customer-product combination is $O(nm^2)$. $\quad\square$

Example 13. *Given a feasible assignment θ, there might not exist a* monotone path *(where each arc is a single reassignment which does not decrease the objective value) connecting θ to an optimal assignment θ^*.*

Consider $m := 1$, $n := 3$, $R := [2,1,1]^\top$, $N := [3,2,2]$, $\theta := [1,0,0]^\top$. The initial objective value is 6. The unique optimal solution is $\theta^* = [1,1,1]^\top$, which has objective value 7, and is more than a single reassignment different from θ. A single reassignment applied to θ will produce objective value 0, if customer 1 is reassigned to product 0, or objective value 5, if either of customer 2 or 3 is reassigned to product 1. Therefore, there is no monotone path from θ to θ^*.

Note that `DK88` as well as `Fast-DK` move only along strictly monotone paths with single reassignments in the θ-space. These algorithms may get stuck in "local" maximizers in this sense of neighbourhood. However, many of our other algorithms, including `Global-DK` and `Cell-Pierce` allow multiple reassignments in a single iteration and have a chance of breaking free of such local maximizers.

Example 14. *In the previous subsection, we observed that there is an optimal θ that induces a shortest-path polyhedron of dimension m. Unfortunately, although we can get this solution using only single reassignments while maintaining feasibility, we might not be able to maintain full-dimensionality.*

Consider $m := 1$, $n := 2$, $N := [1,1]$ and $R := [1,1]^\top$. The unique optimal solution is to assign both customers to product 1, but it is clear that this cannot be done with single reassignments while maintaining a shortest-path polyhedron of dimension m.

Searching just dimension-m polyhedra may be desirable for some algorithms, since moving into a lower-dimensional polyhedron is in some ways a degenerate move and will produce less change in the pricing structure. This example motivates finding more sophisticated reassignments that allow us to move between any two assignments that induce dimension-m polyhedra, while maintaining that dimension.

Proposition 15 (Characterization of Feasible Single Reassignments)**.** *Given a feasible assignment, reassigning a customer i from j to $k \neq j$ produces a feasible assignment if and only if $R_{ij} - R_{ik}$ is the cost of the shortest path from k to j.*

Proof. Suppose that the reassignment is feasible. We know from an earlier proposition that the new assignment must be simultaneously feasible with the previous one. I.e., there exists a price vector π which is feasible with respect to both assignments. From a geometrical viewpoint, in the π-space, the polytopes which correspond to these two assignments "touch." From a combinatorial optimization viewpoint, we can check feasibility in the combined graph.

Suppose that in the old graph, there is a path P from k to j with cost $c(P) < R_{ij} - R_{ik}$. Then in the combined graph, consider the cycle P, jk, which has cost at most $c(P) + (R_{ik} - R_{ij}) < 0$, contradicting simultaneous feasibility.

Conversely, suppose that $R_{ij} - R_{ik}$ is the cost of the shortest path from k to j. Then find a price vector π by computing shortest paths rooted at k, and translating so that $\pi_0 = 0$; this is possible since the old assignment is feasible. By assumption, we have $\pi_j - \pi_k = R_{ij} - R_{ik}$, so $R_{ij} - \pi_j = R_{ik} - \pi_k$, i.e. customer i is indifferent between j and k at the price vector π. Thus π is still a feasible price vector after customer i is reassigned from j to k, proving feasibility of the new assignment. □

Lemma 16 (Weak Exchange Property). *Let θ and θ' be distinct, feasible assignments. Let $(j_i : i \in \{1, 2, \ldots, n\})$ and $(k_i : i \in \{1, 2, \ldots, n\})$ be their respective vectors of purchases. Then there is some i^* such that $j_{i^*} \neq k_{i^*}$ and setting $j_{i^*} := k_{i^*}$ in θ produces a feasible assignment.*

First, we illustrate the proof idea on a concrete example. Suppose that for every i where $j_i \neq k_i$, it is infeasible to set $j_i := k_i$. Then for each of the networks corresponding to these reassignments, there is a negative-cost cycle. For this example, there are three segments, i_1, i_2, i_3 that have different purchases. Thus, we get three negative-cost cycles. Each arc is labeled with the segment that induces it. Gray arcs represent paths that exist in all the considered networks.
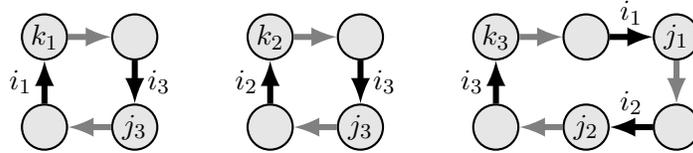
Figure 4: An illustration of the proof idea for the weak exchange property: step 1

Now, create a network where the vertices are the segments, and each gray path with a k node at its tail is an arc.
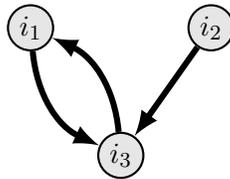
Figure 5: An illustration of the proof idea for the weak exchange property: step 2

Since every node in this new network has an out-arc (this follows from the fact that as in the proof of Proposition 10, the negative-cost cycles must be induced by the customer segments who switched products), there is a cycle. In this case, the cycle uses the out-arcs from i_1 and i_3. Keep only the negative-cost cycles for those segments.
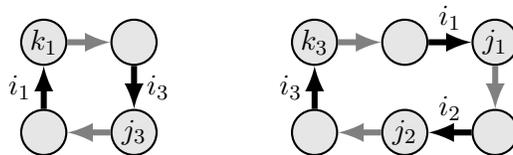
Figure 6: An illustration of the proof idea for the weak exchange property: step 3

Observe that we can rearrange, preserving costs, into a single cycle having the k-nodes and zero or more cycles having the j-nodes. This is always possible.

However, cycles having only k-nodes are cycles in the network of θ', and cycles having only j-nodes are cycles in the network of θ. By feasibility, all those cycles have nonnegative cost, a contradiction.
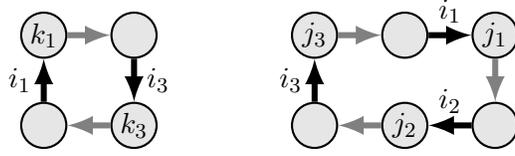
Figure 7: An illustration of the proof idea for the weak exchange property: step 4

Proof. Suppose that for every i such that $j_i \neq k_i$, it is infeasible to set $j_i := k_i$. Then, for each of the networks corresponding to these reassignments, there is a negative-cost cycle. Label each arc with the segment that induces it. Replace the paths that exist in all considered networks by gray arcs (by contracting if necessary). Now, create a network (call this the *auxiliary network*) where the nodes are the segments, and each gray arc (path) with a k node at its tail is an arc.

Since every node in the auxiliary network has an out-arc (again, as in the proof of Proposition 10, the negative-cost cycles must be induced by the customer segments who switched products), there is a cycle in the auxiliary network. Go back to the original negative cost cycles which certified that for every i such that $j_i \neq k_i$, it is infeasible to set $j_i := k_i$. Only keep the negative cost cycles that were involved in the cycle picked from the auxiliary network. Note that we can rearrange, by preserving costs, into a single cycle having the k-nodes and zero or more cycles having the j-nodes (we can do this by cutting the cycles into paths and then solving matching problems, one for the paths involving k-nodes and the other for the paths involving j-nodes). However, cycles having only k-nodes are cycles in the network corresponding to θ', and cycles having only j-nodes are cycles in the network corresponding to θ. Since both θ and θ' are feasible assignments, all those cycles have nonnegative cost, a contradiction. $\square$

Example 17. *We present a simple example where not all such steps are feasible: Let $n := 2, m := 1, R := [1, 2]^{\top}$ and $j_1 := j_2 := 1, k_1 := k_2 := 0$.*

In this case, segment 1 must be reassigned to 0 before segment 2 is, because if segment 1 buys product 1 at some price, then segment 2 will get positive surplus with that price.

The following result immediately follows from the last lemma.

Theorem 18. *Suppose θ and θ' are both feasible assignments that differ in the purchases of ℓ segments. Then there is a sequence of ℓ feasible single reassignments connecting θ to θ'.*

So far, we have studied the fundamental properties of the polyhedral subdivisions induced by the decoupling structure of our MIP as well as some of the fundamental combinatorial structures as they relate to the algorithms. This study also led us to investigating local properties of these subdivisions based on single reassignment of a single customer segment (as used in some of the algorithms). Indeed, in moving from one feasible assignment to a neighboring feasible assignment, exchange properties of the underlying structures are critical. Next, we investigate the possibility of extending some of these exchange properties to a very closely related optimal pricing problem which also has a similar decoupling property. We prove that the fundamental property of *simultaneous feasibility* goes through; however, various versions of the *weak exchange property* does not.

4.3   Analogous results for StackMST

A related optimal pricing problem arises in the area of Stackelberg network pricing games. In this general framework, there is a leader who sets prices on a subset of edges of a given graph (these edges are called *priceable*). The other edges may have some fixed costs. Then, the followers solve a combinatorial optimization problem to choose a minimum cost solution for themselves. Finally, the leader collects these revenues from the followers. The objective for the leader is to find a maximum revenue solution. Note that the arcs with fixed costs are irrelevant to the leader's revenue but are very relevant to the followers' cost. Moreover, these fixed costs and the priced arcs are both expenses to the followers who use them.

In this subsection, we consider the version of this problem where the combinatorial optimization problem solved by the followers is the Minimum Spanning Tree (MST) problem (see [5]). Motivation for our choice is at least two fold:

- MST is one of the most fundamental problems (that could be considered as a building block in many more sophisticated models),

- and it is the prime, matroidal, example for consideration of exchange properties.

We call the above problem StackMST.

A tree T is called *feasible*, if there exists a feasible price vector for which T is an MST.

Example 19. *Between two feasible trees T_1, T_2 for a StackMST problem, it can happen that there is $e \in T_2 \setminus T_1$ such that for all $f \in T_1 \setminus T_2$ such that $T' := T_1 \cup \{e\} \setminus \{f\}$ is a tree, T' is not feasible.*

Observe that both of the following trees, given in Figure 8, are feasible, where the tree edges are double-dashed and priceable edges are indicated by asterisks.
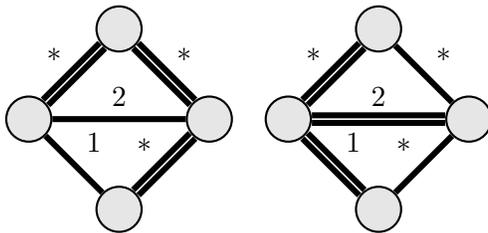
Figure 8: An illustration of the feasible trees for Example 19

However, if the edge with cost 2 is added to the tree on the left, then the top right edge must be deleted, and the resulting tree is infeasible since $2 > 1$, i.e., the tree in Figure 9 is not feasible, since it is not an MST.

The above example is analogous to the fact that in our original pricing problem, given two feasible assignments and a segment i that buys differently in each of the assignments, it is not always feasible to reassign i from its choice in the first assignment to its choice in the second assignment.

Theorem 20 (Simultaneous Feasibility for StackMST). *Suppose T, T' are feasible trees for a StackMST problem, and suppose $|T \setminus T'| = 1$. Then there exists a price vector π for the edges such that T, T' are both minimum-cost spanning trees with respect to π.*
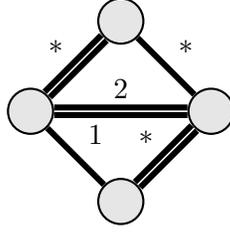
Figure 9: An illustration of the infeasible tree for Example 19

Given a tree T and an edge $e \notin T$, the unique cycle created by adding e to T is called the *fundamental cycle of e in T* and is denoted by cyc_e.

Proof. The idea is to explicitly specify π and check MST optimality conditions for it. (Namely, a tree T is a MST if and only if for every edge $e \notin T$ and every $f \in \mathrm{cyc}_e$, $\pi_e \geq \pi_f$.) Write $\{e\} = T' \setminus T$ and $\{f\} = T \setminus T'$. Let $F = T \cup T'$. Observe that e and f lie on the only cycle in F, call it G; that e lies on cyc'_f, the fundamental cycle of f in T'; and that f lies on cyc_e, the fundamental cycle of e in T. A first observation is that an edge $g \notin T \cup T'$ has $f \in \mathrm{cyc}_g$ if and only if $e \in \mathrm{cyc}'_g$. Moreover, if g has $f \in \mathrm{cyc}_g$, then $\mathrm{cyc}_g \triangle \mathrm{cyc}'_g = G$, and otherwise $\mathrm{cyc}_g = \mathrm{cyc}'_g$. We will always set all prices of priceable edges in $T \cap T'$ to 0 and all prices of priceable edges in $E \setminus (T \cup T')$ to ∞. Together with feasibility of T and T', this ensures that the only optimality conditions we need to check are:

1. $\pi_e \geq \pi_g \quad \forall g \in \mathrm{cyc}_e, \pi_g$ fixed;

2. $\pi_f \geq \pi_g \quad \forall g \in \mathrm{cyc}'_f, \pi_g$ fixed;

3. $\pi_g \geq \pi_e \quad \forall g \notin (T \cup T'), e \in \mathrm{cyc}'_g, \pi_g$ fixed;

4. $\pi_g \geq \pi_f \quad \forall g \notin (T \cup T'), f \in \mathrm{cyc}_g, \pi_g$ fixed.

There are four cases, depending on whether π_e, π_f, or both are fixed. If both π_e and π_f are fixed, then all the four conditions above are obvious. If both π_e and π_f are variable, we set them to be equal to the maximum of 0 and the fixed prices of the edges in the cycle G. Then the first two conditions are clearly satisfied. The second group of conditions are satisfied because if $\pi_e = \pi_f = 0$, then by non-negativity, and otherwise, there is an edge $h \in G$ with fixed π_h such that $\pi_e = \pi_f = \pi_h$, so $h \in \mathrm{cyc}_g \triangle \mathrm{cyc}'_g$, and hence $\pi_h \leq \pi_g$ by feasibility of T and T'. If exactly one of π_e and π_f is variable, without loss of generality, π_e is variable. Set $\pi_f = \pi_e$. Then, the first condition follows from feasibility of T, which then implies the second condition since $\pi_f = \pi_e$. The third condition follows from feasibility of T', and then implies the fourth condition since $\pi_f = \pi_e$. $\square$

Example 21. *We now prove that the weak exchange property does not hold for StackMST, in general. For ease of exposition, the example is presented where neither the fixed-cost edges nor the priceable edges form connected graphs. Clearly these conditions can be achieved by inserting new fixed-cost edges with large cost and inserting new priceable edges, so as to connect the graphs. Consider the following assignment of costs to K_4. The trees are given by double edges. In Figure 10, T is given first, then in Figure 11, we illustrate T''.*
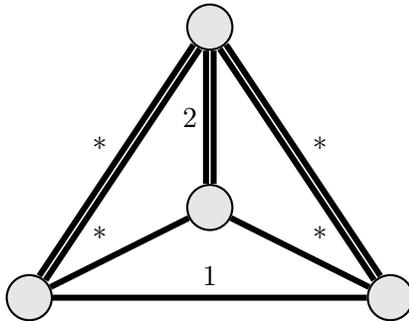
Figure 10: Tree T for Example 21
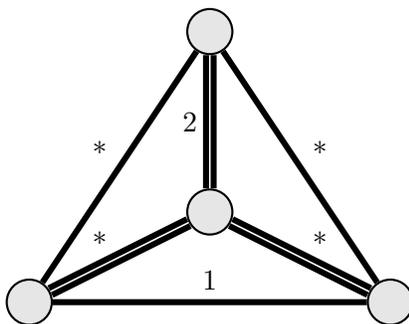
Figure 11: Tree T'' for Example 21

Note that $|T \setminus T''| = 2$. It is clear that adding either edge in $T'' \setminus T$ to T determines exactly one edge from $T \setminus T''$ that must be deleted to maintain the tree. However, after either of these two steps, the resulting tree is infeasible, since the edge of cost 2 lies on the fundamental cycle of the edge of cost 1.

5   Approximation Results

Guruswami et al. [11] give an $O(\log n)$ approximation algorithm for the case where $N_i = 1$ $\forall i$ (also see the related follow=up results in [2]). Their analysis extends to the case where the ratio between the largest N_i and the smallest N_i is bounded by a constant, but not to the fully general problem. Their approximation algorithm is based on selling all products at the same price.

We can easily adopt their heuristic to the general case. Define

$$\bar{R}_i := \max_j \{R_{ij}\}, \quad \forall i \in \{1, 2, \ldots, n\}.$$

Let the customer segments be indexed such that

$$\bar{R}_1 \geq \bar{R}_2 \geq \cdots \geq \bar{R}_n.$$

Then Guruswami et al. type heuristic finds

$$\ell := \operatorname{argmax}_k \left\{ \bar{R}_k \sum_{i=1}^{k} N_i \right\}$$

and chooses

$$\pi_j := \bar{R}_\ell, \quad \forall j \in \{1, 2, \ldots, m\}.$$

We call this algorithm `Guru`. Guruswami et al. [11] gave an example to show that their heuristic algorithm does not yield an approximation ratio better than $\Omega(\log n)$. In the following, we present a closely related family of instances which prove that `Global-DK` as well as `MaxR` followed by `DK88` both can perform as poorly as the approximation ratio $\Omega(\log n)$.

Example 22. *Let $m := n$, $N_i := 1, \forall i \in \{1, 2, \ldots, n\}$. Let ϵ be positive but sufficiently small.*

$$R_{ij} := \begin{cases} 1 + \epsilon, & \text{if } i = j = 1; \\ 1, & \text{if } i = 1, j \geq 2; \\ \frac{1}{i}, & \text{if } i = j \geq 2; \\ 0, & \text{otherwise.} \end{cases}$$

Our variant of `DK93`, `Global-DK`, when run on the above family of instances, would assign a price of $(1+\epsilon)$ to every product and achieve the total profit $1+\epsilon$ out of the possible $\left(\sum_{i=2}^{n} \frac{1}{i} \right) + \frac{1}{n}$. The latter can be achieved by setting $\pi_i := 1/i$, for $i \in \{1, 2, \ldots, n\}$. Therefore, `Global-DK` cannot have better than $\Omega(\log n)$ approximation ratio. Similarly, we can show that in a related family of instances, `MaxR` followed by `DK88` achieves a profit of approximately 1, and therefore this algorithm cannot hope for better than $\Omega(\log n)$ approximation ratio in the worst-case either.

Using the above idea and the structures, we can provide a family of instances on which `Global-DK` and related algorithms perform worse.

Example 23. *Let $m := n$, $N_i := 1, \forall i \in \{1, 2, \ldots, n\}$. Let ϵ be positive but sufficiently small.*

$$R_{ij} := \begin{cases} 1 + \epsilon, & \text{if } i = j = 1; \\ 1, & \text{if } i = 1, j \geq 2; \\ \frac{1}{2}, & \text{if } i = j \geq 2; \\ 0, & \text{otherwise.} \end{cases}$$

In the above family of instances, setting $\pi_i := \frac{1}{2}$, for $i \in \{1, 2, \ldots, n\}$ yields a total profit of $n/2$, while `Global-DK` achieves the total profit $1 + \epsilon$. Therefore, `Global-DK` cannot achieve an approximation ratio better than $\Omega(n)$ (or $\Omega(m)$). Next, we show that when the population sizes, N_i, are not necessarily all the same, the worst-case behaviour of the algorithm `Guru` gets worse as well.

Example 24. *Let $m := n$, and set the reservation prices as follows*

$$R_{ij} := \begin{cases} 2^{(i-1)}, & \text{if } i = j; \\ 0, & \text{otherwise.} \end{cases}$$

Finally, set the population sizes N_i to $2^{n-1}, 2^{n-2}, \ldots, 2^0$.

Then selling all products at the same price always gives profit at most $2^k(2^{n-k} - 1) \leq 2^n$, but the optimal profit is $n2^{n-1}$. Thus, the algorithm has a ratio of $\Omega(n)$ in this example.

Remark 25. *Guruswami et al.'s algorithm has the property that every segment that buys some product buys its most desired product. In this sense, it shares something in common with the* `MaxR` *initialization heuristic.*

Remark 26. *Guruswami et al.'s solution can be found with Dobson-Kalish-style reassignments in the following way. Initialize with MaxR heuristic. Then repeatedly compute optimal prices, choose a segment that is inducing an edge from* 0 *that is tight, and reassign that segment to* 0. *At some point during this procedure, the Guruswami et al. solution will arise.*

We conclude this section with a performance analysis of `Guru` which is stated in terms of population sizes and reservation prices. Our proof technique is based on treating the possible input of the algorithm as variables in an optimization problem set up for computing the worst-case performance ratio.

Proposition 27. *The above heuristic,* `Guru`, *has the worst-case approximation ratio at most:*

$$\min\left\{\sum_{i=1}^n \frac{N_i}{\sum_{\ell=1}^i N_\ell}, n - \sum_{i=1}^{n-1} \frac{\bar{R}_{i+1}}{\bar{R}_i}\right\}.$$

Proof. The ratio is determined by the optimal objective value of the following optimization problem (the variables are N_i and $\bar{R}_i$):

$$
\begin{aligned}
\max \quad & \sum_{i=1}^n N_i \bar{R}_i \\
\left(\sum_{k=1}^i N_k\right) \bar{R}_i \;\leq\;& 1, \quad \forall i \in \{1, 2, \ldots, n\} \\
\bar{R}_{i+1} \;\leq\;& \bar{R}_i, \quad \forall i \in \{1, 2, \ldots, n-1\} \\
\bar{R}_n \;\geq\;& 0, \\
N_i \;\geq\;& 0, \quad \forall i \in \{1, 2, \ldots, n\}.
\end{aligned}
$$

To obtain the second bound, take optimal $\bar{R}_i$, compute the optimal N_i in terms of $\bar{R}_i$:

$$N_1 = \frac{1}{\bar{R}_1}, \quad N_i = \frac{1}{\bar{R}_i} - \frac{1}{\bar{R}_{i-1}}, \quad \forall i \in \{2, \ldots, n\},$$

and evaluate the objective function value. The first bound is very elementary but it can also be obtained from the above problem, by taking optimal N_i, computing the corresponding optimal $\bar{R}_i$, in terms of N_i and evaluating the objective function value. $\qquad\square$

This proposition and the last example prove that the worst-case approximation ratio is $\Theta(n)$. However, for many interesting special cases, the above proposition may yield better approximation ratios. For instance, (N_i may be arbitrary but) if

$$\frac{\bar{R}_{i+1}}{\bar{R}_i} \geq 1 - \frac{k}{n-1}, \quad \forall i \in \{1, 2, \ldots, n-1\},$$

then

$$\text{the approx. ratio } \leq (k+1).$$

6   Computational Experiments

All experiments were run using our `pricedown` software on a 48-core AMD Opteron 6176 machine with 256GB of memory running Red Hat Enterprise Linux 5.8. We compiled `pricedown` using `gcc-4.7.0`. `pricedown` is single-threaded and all times reported are "user time."

6.1   Basic Experiments

We compare the following heuristics:

- `Guru`: The algorithm described in Section 5 based on the heuristic by Guruswami et al. [11].

- $\overline{\text{Guru}}$: Algorithm `Guru` followed by iterating the operators Π and $\mathcal{C}$ until a fixed point is found.

- `MaxR Fast-DK`: `Fast-DK` initialized with the `MaxR` heuristic.

- `MaxR Global-DK`: full line search algorithm `Global-DK`, initialized with the `MaxR` heuristic.

- $\overline{\text{Guru}}$ `GRH-SubTree`: Initialize with $\overline{\text{Guru}}$ and then use `GRH-SubTree` (the version of `GRH` where the direction set is the set of incidence vectors of bottom-up subtrees of the shortest path tree).

- `Cell-Pierce`:

    1. Initialize using `MaxR`.
    2. If a `GRH-SubTree` move improves the objective value, do it and go to 2.
    3. If a `Fast-DK` move improves the objective value, do it and go to 3.
    4. If a `Global-DK` move improves the objective value, do it and go to 8.
    5. If a `GRH-SubTree` move improves the objective value, do it and go to 8.
    6. If a `GRH` move using direction set $\{e_j \pm e_k : 1 \leq j \leq n, 1 \leq k \leq n\}$ with $R_{ij} - R_{ik} = \pi_j - \pi_k$ for some i improves the objective value, do it and go to 8.
    7. Return.
    8. Apply operator $\mathcal{C}$ then operator Π then go to 3.

We used two families of test data. For each m and n in $\{10, 20, 40, 60, 80, 100\}$, we generated ten $m \times n$ problems by selecting each reservation price, segment size, and competitor surplus uniformly among the integers between 0 and 1000. For these problems, we report the *average* CPU time and the ratio of the heuristic objective value to the LP relaxation objective value. These results are recorded in Table 6.1. We used the optimal objective value of the following (lifted) LP relaxation:

$$\begin{array}{rlr}
\max & \sum_{i=1}^{n} \sum_{j=1}^{m} N_i p_{ij} & \\
\text{s.t.} & \pi_k - \sum_{j=1}^{m} v_{ijk} \geq 0 & \forall i, k \\
& v_{ijk} \geq 0 & \forall i, j, k \\
& v_{ijk} \geq (R_{ij} - R_{ik})\theta_{ij} - p_{ij} & \forall i, j, k \\
& p_{ij} \leq \pi_j & \forall i, j \\
& p_{ij} \leq \bar{R}_j \theta_{ij} & \forall i, j \\
& p_{ij} \geq \pi_j - \bar{R}_j(1 - \theta_{ij}) & \forall i, j \\
& p_{ij} \geq 0 & \forall i, j \\
& \sum_j \theta_{ij} \leq 1 & \forall i \\
& \theta_{ij} \geq 0 & \forall i, j \\
& \pi_j \geq 0 & \forall j,
\end{array} \qquad (2)$$

where $\bar{R}_j = \max_{i} \{R_{ij}\}$ and the null product, $j = 0$, is included above with the usual conventions of this paper ($\pi_0 := 0$, $p_{i0} := 0$, $R_{i0} := 0$, $\forall i$).

The above LP relaxation uses the auxiliary variables p_{ij} and v_{ijk}. Some version of p_{ij} variables seem necessary in formulating the bilinear terms $\pi_j \theta_{ij}$ in a MIP. The v_{ijk} variables allow an efficient lifting of various aggregation strategies for the large number of constraints resulting from this linearization. For some examples of such constraint aggregations, see [20]. Our lifted formulation above unifies such strategies and it is at least as strong as any LP relaxation resulting from such aggregation strategies.

To analyze the formulation (2), rearrange all the constraints so that they are of the form

$$Ax + By \leq b,$$

where the x vector is the concatenation of vectors θ, π and p_{ij}, and the vector y has the entries v_{ijk}. Then, the feasible region of the effective formulation in the x-space is

$$P := \{x : Ax + By \leq b, \text{ for some } y\}.$$

The underlying linear inequalities representing the polyhedron P are defined by

$$u^\top A x \leq u^\top b,$$

where u is an extreme ray of the polyhedral convex cone

$$K := \{z : Bz = 0, z \geq 0\}.$$

This association also indicates how one might design matrices A and B to include various classes of integration strategies.

Note that Heilporn, Labbé, Marcotte, and Savard [14] proved that the formulation of [15] is stronger than the formulation used by [20]. Our lifted formulation (2) given above is stronger than all of these formulations. However, our formulation comes with a significant increase in the number of variables and constraints.

We see from the results in the Table 6.1 that the largest instance in the table only took 0.078 seconds to solve with a provable guarantee of being at least 66.78% of the optimal value. When the number of products, m, is much smaller than the number of customer segments and the test data is generated as above, `Global-DK` produced better solutions than `Fast-DK`; however,

n	m	Guru	Guru	MaxR Global-DK	MaxR Fast-DK	Guru GRH-SubTree	Cell-Pierce
10	10	69.130% (0.000 s)	75.409% (0.000 s)	79.962% (0.000 s)	86.049% (0.000 s)	88.933% (0.000 s)	89.188% (0.000 s)
10	20	74.930% (0.000 s)	76.564% (0.000 s)	61.065% (0.000 s)	90.919% (0.000 s)	89.998% (0.000 s)	91.092% (0.000 s)
10	40	66.080% (0.000 s)	70.251% (0.000 s)	63.910% (0.000 s)	82.618% (0.000 s)	83.991% (0.000 s)	87.104% (0.001 s)
10	60	69.685% (0.000 s)	77.547% (0.000 s)	59.608% (0.001 s)	87.579% (0.001 s)	89.298% (0.001 s)	91.351% (0.001 s)
10	80	65.684% (0.000 s)	74.368% (0.000 s)	63.277% (0.001 s)	84.711% (0.001 s)	90.189% (0.000 s)	92.174% (0.001 s)
10	100	64.546% (0.000 s)	78.004% (0.000 s)	49.956% (0.001 s)	82.327% (0.001 s)	88.237% (0.000 s)	90.012% (0.001 s)
20	10	70.456% (0.000 s)	72.608% (0.000 s)	76.489% (0.001 s)	81.842% (0.001 s)	83.750% (0.000 s)	83.800% (0.000 s)
20	20	64.485% (0.000 s)	67.321% (0.000 s)	68.359% (0.002 s)	80.280% (0.001 s)	80.584% (0.000 s)	82.235% (0.001 s)
20	40	63.533% (0.000 s)	65.658% (0.000 s)	58.871% (0.004 s)	78.272% (0.001 s)	80.712% (0.001 s)	83.123% (0.001 s)
20	60	61.396% (0.000 s)	66.012% (0.000 s)	46.543% (0.003 s)	78.667% (0.002 s)	77.917% (0.001 s)	80.767% (0.002 s)
20	80	63.662% (0.001 s)	68.833% (0.001 s)	48.210% (0.006 s)	79.514% (0.003 s)	81.819% (0.001 s)	83.584% (0.003 s)
20	100	59.868% (0.000 s)	64.535% (0.000 s)	50.968% (0.006 s)	75.693% (0.003 s)	75.474% (0.002 s)	78.598% (0.005 s)
40	10	65.317% (0.000 s)	65.317% (0.000 s)	73.445% (0.003 s)	72.180% (0.001 s)	76.200% (0.001 s)	76.704% (0.002 s)
40	20	63.916% (0.000 s)	65.599% (0.000 s)	63.358% (0.010 s)	64.552% (0.002 s)	75.123% (0.002 s)	76.469% (0.003 s)
40	40	60.102% (0.000 s)	62.594% (0.001 s)	58.986% (0.030 s)	68.892% (0.005 s)	72.680% (0.002 s)	74.488% (0.005 s)
40	60	58.325% (0.000 s)	61.299% (0.001 s)	45.652% (0.040 s)	66.466% (0.007 s)	70.343% (0.004 s)	72.024% (0.008 s)
40	80	58.345% (0.001 s)	60.393% (0.001 s)	43.981% (0.045 s)	66.726% (0.012 s)	70.519% (0.005 s)	72.960% (0.011 s)
40	100	59.494% (0.001 s)	62.783% (0.001 s)	36.065% (0.061 s)	69.997% (0.015 s)	72.515% (0.006 s)	75.408% (0.015 s)
60	10	64.073% (0.000 s)	64.073% (0.000 s)	68.942% (0.004 s)	60.254% (0.002 s)	73.196% (0.002 s)	73.605% (0.003 s)
60	20	60.656% (0.000 s)	61.079% (0.001 s)	66.073% (0.019 s)	61.360% (0.005 s)	70.283% (0.004 s)	71.452% (0.007 s)
60	40	58.713% (0.001 s)	59.448% (0.001 s)	55.250% (0.061 s)	60.784% (0.010 s)	68.344% (0.006 s)	70.197% (0.012 s)
60	60	59.280% (0.001 s)	61.233% (0.002 s)	49.666% (0.115 s)	62.309% (0.016 s)	68.427% (0.008 s)	70.365% (0.016 s)
60	80	59.456% (0.001 s)	60.854% (0.002 s)	47.303% (0.162 s)	63.211% (0.022 s)	69.350% (0.011 s)	71.030% (0.023 s)
60	100	58.760% (0.001 s)	59.996% (0.003 s)	45.503% (0.238 s)	63.927% (0.031 s)	68.315% (0.015 s)	70.284% (0.032 s)
80	10	62.991% (0.000 s)	62.991% (0.001 s)	69.375% (0.006 s)	56.432% (0.002 s)	69.461% (0.003 s)	69.953% (0.004 s)
80	20	61.280% (0.001 s)	61.280% (0.001 s)	64.923% (0.030 s)	54.925% (0.007 s)	69.341% (0.008 s)	70.013% (0.010 s)
80	40	58.974% (0.001 s)	59.329% (0.002 s)	60.058% (0.124 s)	58.266% (0.016 s)	68.186% (0.014 s)	69.475% (0.022 s)
80	60	57.319% (0.001 s)	58.271% (0.003 s)	48.642% (0.224 s)	54.569% (0.026 s)	66.577% (0.016 s)	68.000% (0.027 s)
80	80	56.380% (0.002 s)	57.584% (0.004 s)	48.265% (0.343 s)	58.765% (0.044 s)	65.521% (0.023 s)	67.307% (0.044 s)
80	100	56.651% (0.002 s)	57.707% (0.005 s)	47.633% (0.495 s)	58.340% (0.054 s)	65.701% (0.023 s)	67.760% (0.053 s)
100	10	61.661% (0.001 s)	61.661% (0.001 s)	67.040% (0.008 s)	44.893% (0.004 s)	67.621% (0.004 s)	67.806% (0.005 s)
100	20	60.965% (0.001 s)	61.102% (0.002 s)	62.773% (0.039 s)	54.504% (0.011 s)	68.385% (0.010 s)	68.815% (0.013 s)
100	40	56.551% (0.001 s)	57.031% (0.003 s)	60.081% (0.185 s)	56.923% (0.025 s)	64.317% (0.022 s)	65.854% (0.035 s)
100	60	57.387% (0.003 s)	57.880% (0.004 s)	54.449% (0.374 s)	57.812% (0.047 s)	65.817% (0.035 s)	66.826% (0.050 s)
100	80	56.354% (0.003 s)	56.871% (0.005 s)	50.954% (0.605 s)	55.234% (0.072 s)	63.820% (0.044 s)	65.496% (0.074 s)
100	100	56.852% (0.004 s)	57.633% (0.007 s)	45.916% (0.846 s)	56.352% (0.080 s)	64.805% (0.042 s)	66.784% (0.078 s)

Table 1: Average gaps (ratio of the heuristic objective value to the LP relaxation objective value) and computation times for the tested heuristics on 10 test cases of the specified sizes.

in other cases, `Fast-DK` did very well including its speed. In the worst-case, the number of iterations of `Fast-DK` is at least quadratic in n [7]; however, in our experiments (with randomly generated data as well as with the real data, see Table 6.3) `Fast-DK` was indeed much faster than its worst-case lower bound. Overall, our algorithms based on Generalized Reassignment Heuristic, $\overline{\text{Guru}}$ GRH-SubTree and `Cell-Pierce`, performed in a more robust manner, generating feasible solutions with objective value at least as good as the best of `Fast-DK` and `Global-DK`. Moreover, in some cases, these good solutions were attained in a fraction of the time spent by `MaxR Global-DK`. Based on the computational results, we also suggest that $\overline{\text{Guru}}$ can serve as a robust initialization heuristic for optimal pricing problems.

6.2 Large-Scale Experiments

Our second data set is generated as follows. For each m in $\{200, 400, 600, 1000\}$ and each n in $\{5000, 10000, 20000, 40000\}$, we generated one $m \times n$ problem. To do this, we generated an $(m + 5) \times 20$ matrix V and a $20 \times n$ matrix W choosing each entry uniformly from $[-32, 32]$. We formed $W = UV$, added normally distributed noise with mean zero and standard deviation 20, rounded to integers, and clamped negative entries at zero. The reservation price matrix is the first m rows of W, the competitor surplus vector is the elementwise maximum of the last five rows, and the segment sizes are drawn uniformly from the integers in $[512, 1023]$. We also generated these instances with $m \in \{5000, 10000, 20000, 40000\}$ and $n \in \{200, 400, 600, 1000\}$. For these problems, we report the CPU time taken and the improvement over the `Guru` heuristic. We put an 86400-second CPU time limit and an 32GB memory limit on each run. These results are recorded in Table 6.2.

One does not market tens of thousands of distinct but interchangeable products to thousands of distinct, identifiable customer segments in practice. However, being able to solve, even heuristically, large-scale pricing problems allows one flexibility in modelling.

One might use a large number of products in order to price product bundles optimally (for a classical MIP approach see [13]). In many applications, a company may offer only a small number of products but sell the products in various bundles whose prices can be individually set. (Indeed, declaring the price of a bundle as the sum of the prices of its ingredients rarely leads to an optimal solution.) Within our framework, one can simply model each bundle as a separate product; as the number of bundles increase so does the number of products, m, in our formulation.

The large values of n can be explained by the large amount of data available to the companies about their current and potential customers. Due to very detailed information gathered by the companies about their customers (e.g., zip code) including the information gathered through customer loyalty programs (what and when each customer is buying, their exact address, income level, etc.) and the data mining tools allowing the analysis of these data, the companies are able to divide their current and potential customers into very many customer segments. Moreover, in our mathematical model, we had a unit demand assumption (each customer in a customer segment buys at most one product), an obvious way to relax this assumption in practice is to simulate the behaviour of a given customer by many customers in the mathematical model. These tricks can increase the number of customer segments very significantly.

In addition, the readers may wonder whether it is reasonable to generate these large-scale instances from small rank, e.g., rank 20, matrices. Indeed, our discussion above of why the large-scale instances may be necessary to solve in practice explains that even though m and

n themselves may be very large, usually there is a small number (like 20) of most important driving/determining main factors for m as well as n. Clearly, the addition of random noise is justified by the difficult nature of quantifying customer preferences.

Many other metrics could be considered here; we also performed experiments with many different combinations of our algorithms during 2009–2012. In this section, we only reported the results for the very few versions that we thought were most interesting. However, we provide the C++11 source to `pricedown`, our input files, and our computation logs at `http://csclub.uwaterloo.ca/~tmyklebu/pricing/` interested researchers may freely construct and try out their favourite variants.

Our most sophisticated and resource consuming algorithm is `Cell-Pierce`. During our preliminary experiments, for each cell, the algorithm considered $\Theta(m^2)$ directions. This was extremely time-consuming for the large instances. We then tried doing some preprocessing in every iteration to determine first which constraints are tight for the current cell at the current price vector and then use only the directions $e_i \pm e_j$ that arise from those inequalities. This latter idea worked very well computationally and our reported results use this version.

The results in Table 6.2 suggest that the iterated operator part of the heuristic $\overline{\text{Guru}}$ does not improve the objective value much over the results of the basic heuristic `Guru` when n is much larger then m (even though the former can be very time consuming). On the other hand, when m is much larger then n, $\overline{\text{Guru}}$ can improve the objective value as much as 23% over that of `Guru` objective value. We see that the behaviours of the algorithms that we noticed during our basic experiments (see Table 6.1) are still present in these large-scale experiments and they are only magnified. A very promising result is that our most sophisticated heuristics $\overline{\text{Guru}}$ GRH-SubTree and `Cell-Pierce` also continued their good behaviour observed in the basic experiments, generating good feasible solutions within generally reasonable computation times.

6.3   Experiments using historical vacation package sales data

We ran the heuristics on data derived from historical vacation package sales. This is the same data as in [20] and [19] and the procedure for deriving the needed reservation prices is described in an appendix of [19]. We unfortunately cannot make these datasets available to other researchers due to a confidentiality agreement with the data provider.

The methods in [19], when applied to the historical sales data that we have, produced one dataset with 7117 customers and 234 products and another dataset with 9148 customers and 234 products. We generated "reduced" versions of these two datasets by clustering the customer segments into various numbers of clusters and taking a cluster's reservation price list to be the weighted mean of its constituents' reservation price lists. The clustering is unsophisticated; we randomly assign the customers to clusters then hill-climb to minimise the sum of Euclidean distances across clusters. In no case was the set of products changed.

One will note from the results in Table 6.3 that the heuristics `Cell-Pierce`, $\overline{\text{Guru}}$, GRH-Subtree, and `MaxR Global-DK` produce solutions with rather similar objective values on these cases. Even though most of the patterns we noted in the experiments with randomly generated data are present in Table 6.3, compared to the performance with randomly generated data, in the real data case, `Guru`, and as a result $\overline{\text{Guru}}$ did worse, `MaxR Global-DK` did significantly better. `MaxR Global-DK` also became more robust with the real data (partly due to the fact that ranges of m and n seem to favor `MaxR Global-DK` in this case). `Cell-Pierce` was still the most robust; however, due to the fact that the real data was *easy* for `MaxR Global-DK`, `Cell-Pierce` did not

n	m	Guru	MaxR Global-DK	MaxR Fast-DK	Guru GRH-SubTree	Cell-Pierce
5000	200	100.0467 (.009 h)	108.8830 (.096 h)	75.5171 (.058 h)	108.6972 (.102 h)	109.1345 (.111 h)
5000	400	100.2450 (.022 h)	111.8544 (.426 h)	86.5640 (.124 h)	111.2061 (.611 h)	112.3437 (.564 h)
5000	600	100.3346 (.036 h)	112.7227 (1.339 h)	96.3856 (.249 h)	112.4902 (1.045 h)	113.7898 (1.115 h)
5000	1000	100.8233 (.068 h)	114.1830 (2.904 h)	102.4372 (.752 h)	113.6746 (3.234 h)	115.2064 (3.107 h)
10000	200	100.0000 (.030 h)	106.3786 (.254 h)	54.2994 (.228 h)	106.3399 (.246 h)	106.6803 (.369 h)
10000	400	100.1346 (.109 h)	109.1887 (1.127 h)	75.1328 (.519 h)	109.1368 (1.598 h)	109.6671 (1.415 h)
10000	600	100.0365 (.106 h)	109.8326 (4.907 h)	79.3633 (.739 h)	109.4865 (3.145 h)	110.2880 (3.412 h)
10000	1000	100.3057 (.245 h)	111.4575 (12.736 h)	83.8101 (2.017 h)	111.3775 (9.760 h)	112.4647 (15.479 h)
20000	200	100.0060 (.170 h)	105.4225 (.719 h)	47.3979 (.704 h)	105.2912 (.710 h)	105.4328 (.761 h)
20000	400	100.0410 (.316 h)	107.3023 (3.008 h)	53.5044 (1.581 h)	107.1143 (5.327 h)	107.4160 (4.147 h)
20000	600	100.0377 (.603 h)	106.5057 (6.495 h)	69.8042 (3.327 h)	106.4201 (10.476 h)	106.7048 (9.766 h)
20000	1000	100.0984 (1.077 h)	108.0280 (20.442 h)	80.8387 (6.184 h)	107.5701 (23.993 h)	107.5012 (23.995 h)
40000	200	100.0085 (3.227 h)	104.6436 (1.965 h)	23.9715 (1.309 h)	104.6112 (3.758 h)	104.6882 (1.755 h)
40000	400	100.0106 (2.146 h)	105.8200 (7.715 h)	41.6453 (4.888 h)	105.6322 (10.800 h)	105.8977 (9.772 h)
40000	600	100.0000 (.597 h)	104.4826 (23.994 h)	47.8095 (21.661 h)	105.0925 (23.993 h)	104.6178 (23.994 h)
40000	1000	100.0274 (7.235 h)	90.3893 (23.994 h)	77.8084 (23.993 h)	104.1509 (23.994 h)	104.1252 (23.994 h)
200	5000	123.6134 (0 h)	135.5827 (0 h)	155.7910 (.012 h)	155.1074 (.004 h)	157.0861 (.011 h)
200	10000	115.0773 (0 h)	152.0927 (.003 h)	157.8988 (.008 h)	157.4648 (.007 h)	160.4208 (.024 h)
200	20000	113.2464 (.001 h)	145.2699 (.007 h)	147.8242 (.025 h)	147.5414 (.050 h)	148.3224 (.137 h)
200	40000	117.4900 (.002 h)	146.2617 (.048 h)	150.4751 (.114 h)	153.0789 (.134 h)	153.7721 (.589 h)
400	5000	117.4507 (.001 h)	136.3736 (.050 h)	144.5134 (.020 h)	142.3170 (.037 h)	146.0311 (.055 h)
400	10000	119.3917 (.002 h)	130.0758 (.042 h)	143.9515 (.045 h)	144.9777 (.036 h)	146.0984 (.073 h)
400	20000	117.7649 (.005 h)	135.5036 (.043 h)	148.2505 (.197 h)	148.1672 (.142 h)	150.1365 (.213 h)
400	40000	110.1589 (.008 h)	142.6810 (.031 h)	149.1894 (.526 h)	149.7208 (1.002 h)	150.7921 (1.610 h)
600	5000	114.3141 (.003 h)	126.6203 (.114 h)	141.0762 (.082 h)	140.5073 (.074 h)	142.7010 (.116 h)
600	10000	113.1518 (.007 h)	131.4402 (.070 h)	142.7323 (.108 h)	142.7636 (.118 h)	145.7095 (.237 h)
600	20000	111.4827 (.010 h)	137.9467 (.732 h)	143.8965 (.265 h)	142.1926 (.317 h)	144.7289 (.703 h)
600	40000	110.8461 (.021 h)	135.5177 (.472 h)	140.6506 (.646 h)	140.3603 (.246 h)	142.5931 (1.987 h)
1000	5000	112.5696 (.010 h)	125.9860 (.873 h)	132.0921 (.152 h)	132.3481 (.264 h)	135.3372 (.398 h)
1000	10000	115.5102 (.021 h)	128.0345 (1.270 h)	136.4630 (.482 h)	136.3058 (.495 h)	139.2389 (.758 h)
1000	20000	112.7683 (.032 h)	122.4778 (1.159 h)	138.9816 (1.351 h)	138.2458 (4.716 h)	141.5158 (2.937 h)
1000	40000	115.0332 (.077 h)	133.2337 (1.520 h)	143.7260 (4.039 h)	142.8655 (1.741 h)	146.1982 (6.442 h)

Table 2: Quotient of solution value to Guru solution value and computation time.

provide the same level of additional improvement.

7   Conclusions

We provided very efficient implementations of `DK88` for optimal pricing problem and new heuristic algorithms that perform significantly better under various conditions. Our ideas may also be useful for other mixed integer programming problems where a similar decoupling of the continuous and integer variables is present.

References

[1] G. Aggarwal, T. Feder, R. Motwani and A. Zhu, Algorithms for multi-product pricing. *Automata, Languages and Programming*, pp. 72–83, Lecture Notes in Comput. Sci., 3142, Springer, Berlin, 2004.

[2] M. Bouhtou, A. Grigoriev, S. van Hoesel, A. F. van der Kraaij, F. C. R. Spieksma and M. Uetz. Pricing bridges to cross a river. *Naval Res. Logist.* 54 (2007) 411–420.

[3] P. Briest. Towards hardness of envy-free pricing. *Electronic Colloqium on Computational Complexity*, Report No. 150, 2006.

[4] P. Briest, M. Hoefer and P. Krysta, Stackelberg network pricing games, *Symposium on Theoretical Aspects of Computer Science* 2008, pp. 132-144.

[5] J. Cardinal, E. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman, and O. Weimann, The Stackelberg Minimum Spanning Tree Game, In *Proc. of 10th WADS*, 2007.

[6] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Inc., New York, 1998.

[7] D. Demirtaş. *Worst-Case Complexity Analysis for the Dobson-Kalish Optimal Pricing Algorithm and its Relatives*. M.Math. Essay, Department of Combinatorics and Optimization, University of Waterloo, Canada, 2010.

[8] G. Dobson and S. Kalish. Positioning and pricing a product line. *Marketing Science* 7 (1988) 107–125.

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

[10] B. Guenin, J. Könemann and L. Tunçel, *A Gentle Introduction to Optimization*, Cambridge University Press, 2014.

[11] V. Guruswami, J. Hartline, A. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, 2005, pp. 1164–1173.

[12] O. Günlük. A pricing problem under Monge property. *Disc. Optim.* 5 (2008) 328–336.

[13] W. Hanson and R.K. Martin. Optimal bundle pricing. *Management Science* 36 (1990) 155–174.

[14] G. Heilporn, M. Labbé, P. Marcotte and G. Savard, Valid inequalities and branch-and-cut for the clique pricing problem, *Discrete Optimization* 8 (2011) 393–410.

[15] G. Heilporn, M. Labbé, P. Marcotte and G. Savard, A polyhedral study of the network pricing problem with connected toll arcs, *Networks* 55 (2010) 234-246.

n_{orig}	n	$\overline{\text{Guru}}$	MaxR Global-DK	MaxR Fast-DK	$\overline{\text{Guru}}$ GRH-SubTree	Cell-Pierce
7117	20	1.109846 (0.000 h)	1.340405 (0.000 h)	1.332328 (0.000 h)	1.326056 (0.000 h)	1.379898 (0.000 h)
7117	40	1.199965 (0.000 h)	1.372700 (0.000 h)	1.258547 (0.000 h)	1.399828 (0.000 h)	1.420796 (0.000 h)
7117	60	1.170783 (0.000 h)	1.404354 (0.000 h)	1.411266 (0.000 h)	1.408558 (0.000 h)	1.436655 (0.000 h)
7117	80	1.110223 (0.000 h)	1.406168 (0.000 h)	1.400128 (0.000 h)	1.340770 (0.000 h)	1.388602 (0.000 h)
7117	100	1.109573 (0.000 h)	1.402892 (0.000 h)	1.378750 (0.000 h)	1.389410 (0.000 h)	1.414016 (0.000 h)
7117	200	1.099071 (0.000 h)	1.279759 (0.000 h)	1.172690 (0.000 h)	1.286883 (0.000 h)	1.293138 (0.000 h)
7117	500	1.099211 (0.000 h)	1.322974 (0.002 h)	0.969098 (0.000 h)	1.290349 (0.001 h)	1.343019 (0.002 h)
7117	1000	1.054702 (0.001 h)	1.304169 (0.005 h)	0.837517 (0.003 h)	1.281609 (0.005 h)	1.324190 (0.007 h)
7117	2000	1.036667 (0.006 h)	1.312206 (0.015 h)	0.733839 (0.012 h)	1.302699 (0.018 h)	1.336229 (0.023 h)
7117	5000	1.076994 (0.060 h)	1.244568 (0.053 h)	0.479482 (0.050 h)	1.277673 (0.113 h)	1.319164 (0.093 h)
7117	7117	1.007253 (0.039 h)	1.457783 (0.083 h)	1.166382 (0.030 h)	1.389568 (0.056 h)	1.463782 (0.101 h)
9148	20	1.153943 (0.000 h)	1.698111 (0.000 h)	1.693019 (0.000 h)	1.693019 (0.000 h)	1.698111 (0.000 h)
9148	40	1.303224 (0.000 h)	1.670624 (0.000 h)	1.660871 (0.000 h)	1.669130 (0.000 h)	1.670624 (0.000 h)
9148	60	1.259112 (0.000 h)	1.637445 (0.000 h)	1.595959 (0.000 h)	1.638282 (0.000 h)	1.638606 (0.000 h)
9148	80	1.218465 (0.000 h)	1.702579 (0.000 h)	1.518892 (0.000 h)	1.701917 (0.000 h)	1.702579 (0.000 h)
9148	100	1.169126 (0.000 h)	1.712444 (0.000 h)	1.622756 (0.000 h)	1.710712 (0.000 h)	1.711692 (0.000 h)
9148	200	1.137791 (0.000 h)	1.681346 (0.000 h)	1.506625 (0.000 h)	1.682116 (0.000 h)	1.683095 (0.000 h)
9148	500	1.065883 (0.000 h)	1.573282 (0.002 h)	1.175608 (0.001 h)	1.572475 (0.002 h)	1.573751 (0.002 h)
9148	1000	1.162142 (0.002 h)	1.564558 (0.005 h)	0.935925 (0.004 h)	1.563288 (0.006 h)	1.565473 (0.007 h)
9148	2000	1.051191 (0.009 h)	1.502041 (0.012 h)	0.710806 (0.016 h)	1.498817 (0.021 h)	1.502264 (0.024 h)
9148	5000	1.080402 (0.071 h)	1.436485 (0.033 h)	0.527796 (0.064 h)	1.428032 (0.103 h)	1.437561 (0.119 h)
9148	9148	1.010496 (0.058 h)	1.416377 (0.072 h)	1.041056 (0.039 h)	1.368147 (0.115 h)	1.414822 (0.151 h)

Table 3: Quotient of solution value to Guru solution value and computation time.

[16] G. Heilporn, M. Labbé, P. Marcotte and G. Savard, A parallel between two classes of pricing problems in transportation and marketing, *Journal of Revenue and Pricing Management* 9 (2010) 110-125.

[17] F. P. Preparata and M. I. Shamos, *Computational geometry. An introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1985.

[18] P. Rusmeivhichientong, B. Van Roy, P. W. and Glynn, A non-parametric approach to multi-product pricing: theory and application, *Oper. Res.* 54 (2006) 82–98.

[19] R. Shioda, L. Tunçel and B. Hui. Applications of deterministic optimization techniques to some probabilistic choice models for product pricing using reservation prices, *Pacific Journal of Optimization* 10 (2014) 767–808.

[20] R. Shioda, L. Tunçel and T. G. J. Myklebust. Maximum utility product pricing models and algorithms based on reservation prices. *Computational Optimization and Applications* 48 (2011) 157–198.

[21] L. Tunçel, Optimization based approaches to product pricing, *Selected Proceedings of ICBME'2008* vol. II, O. İçöz, C. Pinar (eds.), Yaşar University, İzmir, Turkey, pp. 93–102, 2008.

[22] L. Walras, *Elements of Pure Economics*, Translated by W. Jaffé, Homewood, Illinois: Richard D. Irwin, Inc. 1954.