Efficient Heuristic Algorithms for Maximum Utility Product Pricing Problems

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

November 19, 2012

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
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@math.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.

‡Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada (e-mail: *ltuncel@math.uwaterloo.ca*). 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 one of 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, [9, 10, 11]. 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 assumptions 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*. Maximum utility pricing models are also related to bilevel pricing problems. For the latter, see, for instance, [8] and the references therein.

From a computational complexity viewpoint, the problem is $\mathcal{NP}$-hard, see [4, 1]; it is $\mathcal{NP}$-hard even to approximate within a reasonable worst-case ratio, see [5] (even though some special cases of related optimal pricing problems admit efficient algorithms, see for instance [6]). Currently, the best heuristic algorithms for solving these problems are the ones proposed by Dobson and Kalish, see [3, 4]. While these algorithms are successful on many instances of the problem, quite often they are not.

We present very efficient implementations of Dobson-Kalish optimal pricing algorithms [3, 4]. 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 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.

2 Notation and Fundamental Ingredients of the Heuristics

Suppose we have n homogeneous customer segments and m products. R_{ij} denotes the reservation price of customer segment i for product j. Let our decision variables be as follows:

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

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

We denote by CS_i 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:

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

The assumptions stipulated above are not very restrictive in practice as there are always 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) (for details, see [11] and [12]).

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 $B := \{j : C_j \neq \emptyset\}$. 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 [3, 11].

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 [3], as well as the ideas offered by Dobson and Kalish [4], Shioda et al. [11] and Guruswami et al. [5]. 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.

- 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 [11] used a greedy initialization heuristic called `MaxR`, described in the next section. However, Guruswami et al. [5] 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 $\overline{\texttt{Guru}}$ 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. We conclude Section 4 by analyzing the polyhedral subdivision induced by the MIP in the price space. The exchange results established for these polyhedra motivate a connection to another optimal pricing problem called the Stackelberg Network Pricing Game in Section 5. In Section 6, 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 7. We conclude with brief final remarks in Section 8.

3 Generalized Reassignment Heuristic and Related Algorithms

As in Shioda et al. [11], 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 $C_j = \{i : j = \arg\max_k\{R_{ik}\}\}, \forall j$.
2: Assign each customer to his maximum-utility product.

Based on our assumptions, every $\pi \in \mathbb{R}^m_+$ defines a corresponding product-segment assignment $\mathcal{C} := \{C_1, C_2, \ldots, C_m\}$. 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 price, 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 [9]. 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.

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

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

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 j to product

k with weight

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

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 as an array of length m of pointers to arrays of length m. We store pricing graphs of reassigned subproblems as a copy of the array of m pointers and a (short) array of columns that have been replaced. Replacing row j is done by simply overwriting the jth pointer in the array of pointers to rows, while replacing column k is done by adding k to the array of indices of replaced columns and adding the replacement column to the array of replaced columns. Looking up the weight of the arc from j to k is done by checking whether column k has a replacement, using it if so, and using the kth entry of the jth row otherwise.

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 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 $d \in \mathcal{D}_\pi$.

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

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 $z := \max_{d\in\mathcal{D}_\pi}\{z_d\}$, $\bar{d} := \operatorname{argmax}_{d\in\mathcal{D}_\pi}\{z_d\}$, $\pi := \Pi\left(\mathcal{C}(\pi + \alpha_{\bar{d}}\bar{d})\right)$.
7: **until** Maximum number of iterations.

current prices, or proportional to some "target" R_{ij} (which may be determined by N_i, M_j or N_iM_j) for each j, or proportional to the distance to such a target $(R_{ij} - \pi_j)$.

GRH also has the ability to "revive" products or customers that were eliminated. In the `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. [11] 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. 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 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.

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. 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.

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 by an inequality of the form $\pi_j \geq 0$. 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 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 [11]), 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, it 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 $\Pi(\mathcal{C}(\pi)) = \pi$.

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 $\pi := \Pi(\theta)$.*

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 w.r.t. π have non-negative cost, and θ is lexicographically least under this restriction.*

Proposition 4. *Given any 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 $O(nm)$. Therefore, $\mathcal{C}(\Pi)$ can be computed in strongly polynomial time. □

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*

$$N := (+\infty, +\infty, 1), R := \begin{bmatrix} 3 & 0 \\ 0 & 1 \\ 3 & 2 \end{bmatrix}, \theta := \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \pi := (3, 1).$$

Consider deleting customer 1, so $\tilde{\theta} := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\tilde{\pi} := (3, 2)$. However, $\tilde{\theta}$ is not optimal for $\tilde{\pi}$, since customer 2 can be reassigned to product 1 to obtain more profit.

Proposition 6. *If 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, so all the arcs of the shortest path tree will be non-negative. 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. □

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. □

Proposition 8. *If θ is a fixed point, 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, $\dim(P) = m$. □

Example 9. *$\Pi(\mathcal{C})$ is not idempotent in general.*

Consider

$$
\begin{aligned}
R &:= \begin{bmatrix} n+1 & 0 \\ n+1 & 1 \\ n+1 & 2 \\ \vdots & \vdots \\ n+1 & n \end{bmatrix}, \\
\theta^{(0)} &:= \begin{bmatrix} e_1, & \sum_{j=2}^{n} e_j \end{bmatrix}, \\
\pi^{(0)} = \Pi(\theta^{(0)}) &= (n+1, 1), \\
\theta^{(1)} = \mathcal{C}(\pi^{(0)}) &= \begin{bmatrix} e_1 + e_2, & \sum_{j=3}^{n} e_j \end{bmatrix}, \\
\pi^{(1)} = \Pi(\theta^{(1)}) &= (n+1, 2), \\
\theta^{(2)} = \mathcal{C}(\pi^{(1)}) &= \begin{bmatrix} e_1 + e_2 + e_3, & \sum_{j=4}^{n} e_j \end{bmatrix}, \\
\pi^{(2)} = \Pi(\theta^{(2)}) &= (n+1, 3), \\
&\vdots \\
\theta^{(n)} = \mathcal{C}(\pi^{(n-1)}) &= \begin{bmatrix} \sum_{j=1}^{n} e_j, & 0 \end{bmatrix}, \\
\pi^{(n)} &= \Pi(\theta^{(n)}) = (n+1, +\infty).
\end{aligned}
$$

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 $\theta \in \{0,1\}^{mn}$, we obtain a polyhedron in the π-space from our MIP. Moreover, under this map, union of all feasible assignments partitions a subset of 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 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 products between θ and θ'. Let $j = \theta_i$ and $k = \theta'_i$. Suppose there is a negative-cost cycle C in H. Without loss of generality, C is simple. 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. 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_1 and $C_2 := j'k, P_2$ be a cycle in G_2. 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. □

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, and assign that segment to the null product. □

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,

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

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

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 $R = (2; 1; 1)$, $N = (3, 2, 2)$, $\theta = (1, 0, 0)$. The initial objective value is 6. The unique optimal solution is $\theta^* = (1, 1, 1)$, 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 $R = (1; 1)$. 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 shortest-path 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.*

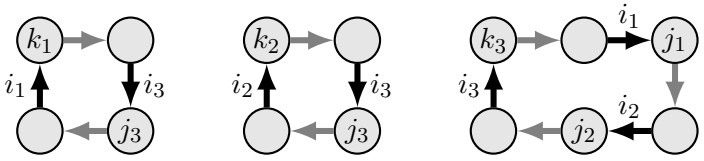
Proof. Suppose that the reassignment is feasible. We know from an earlier proposition that the new assignment must be simultaneously feasible with the previous one. i.e. We can check feasibility in the combined graph.

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.

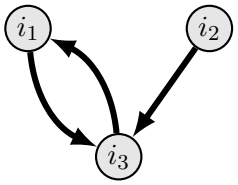
Conversely, suppose that $R_{ij} - R_{ik}$ is the cost of the shortest path from k to j. Then find a price vector π by computing shortest paths rooted at k, and translating so that $\pi_0 = 0$; this is possible since the old assignment is feasible. By assumption, we have $\pi_j - \pi_k = R_{ij} - R_{ik}$, so $R_{ij} - \pi_j = R_{ik} - \pi_k$, i.e. customer i is different to 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 4.1 (Weak Exchange Property)**.** *Let θ and θ' be 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.*

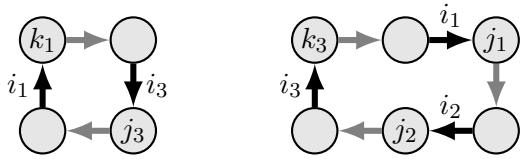
First, we illustrate the proof idea on a concrete example. Suppose that for every i where $j_i \neq k_i$, it is infeasible to set $j_i := k_i$. Then for each of the networks corresponding to these reassignments, there is a negative cycle. For this example, there are three segments, i_1, i_2, i_3 that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



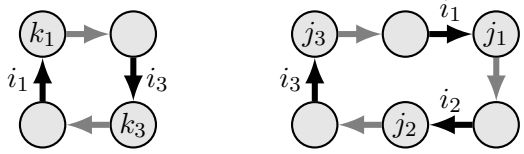
Now, create a network where the vertices are the segments, and each gray path with a k node at its tail is an arc.



Since every node in this new network has an out-arc, 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.



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 non-negative cost, a contradiction.

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 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, 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. 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 non-negative cost, a contradiction. □

Example 16. *We present a simple example where not all such steps are feasible: Let* $n := 2, m := 1, R_{11} := 1, R_{21} := 2, j_1 := j_2 := 1, k_1 := k_2 := 0$.

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 is immediate.

Theorem 17. *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'.*

5 Analogous results for StackMST

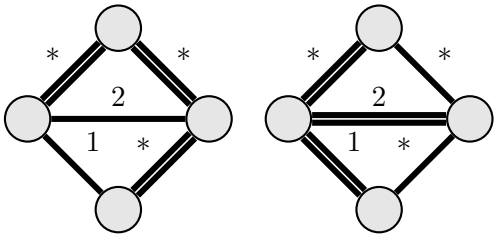
A related optimal pricing problem arises in the area of Stackelberg network pricing games. In this general framework, there is a leader who sets prices on a subset of edges of a given graph (these edges are called *priceable*). The other edges may have some fixed costs. Then, the followers solve a combinatorial 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. In this section, we consider the version of this problem where the combinatorial optimization problem solved by the followers is the Minimum Spanning Tree (MST) problem. 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.

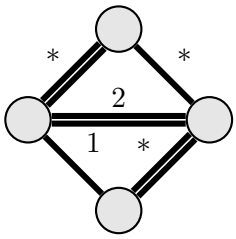
We call the above problem StackMST.

Example 18. *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 the following trees are feasible, where the tree edges are double-dashed and priceable edges are indicated by asterisks.



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 $2 > 1$.



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.

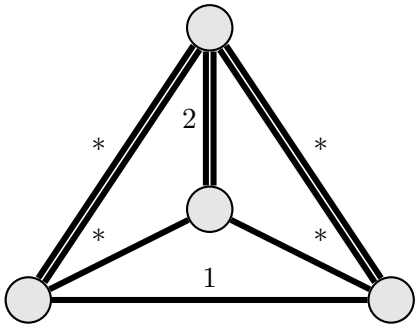
Theorem 19 (Simultaneous Feasibility for StackMST)**.** *Suppose T, T' are feasible trees for a StackMST problem, and suppose $|T \setminus T'| = 1$. Then there exists a price vector π for the edges such that T, T' are both minimum-cost spanning trees with respect to π.*

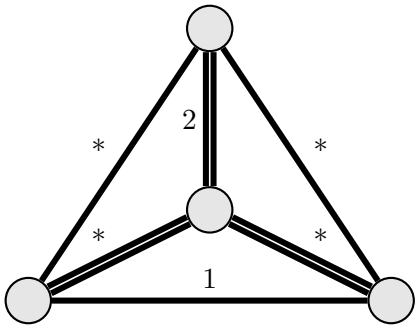
Proof. The idea is to explicitly specify π and check MST optimality conditions for it. (Namely, a tree T is a MST if and only if for every edge $e \notin T$ and every $f \in C_e$, $\pi_e \geq \pi_f$.) Write $\{e\} = T' \setminus T$ and $\{f\} = T \setminus T'$. Let $F = T \cup T'$. Observe that e and f lie on the only cycle in F, call it G; that e lies on C'_f, the fundamental cycle of f in T'; and that f lies on C_e, the fundamental cycle of e in T. A first observation is that an edge $g \notin T \cup T'$ has $f \in C_g$ if and only if $e \in C'_g$. Moreover, if g has $f \in C_g$, then $C_g \triangle C'_g = G$, and otherwise $C_g = C'_g$. 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. $\pi_e \geq \pi_g \quad \forall g \in C_e, \pi_g$ fixed;
2. $\pi_f \geq \pi_g \quad \forall g \in C'_f, \pi_g$ fixed;
3. $\pi_g \geq \pi_e \quad \forall g \notin (T \cup T'), e \in C'_g, \pi_g$ fixed;
4. $\pi_g \geq \pi_f \quad \forall g \notin (T \cup T'), f \in C_g, \pi_g$ 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 $h \in C_g \triangle C'_g$, 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 $\pi_f = \pi_e$. □

Example 20. *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. T is given first, then T''.*





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.

6 Approximation Results

Guruswami et al. [5] give a $O(\log n)$ approximation algorithm for the case where $N_i = 1\ \forall i$. 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

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

and chooses

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

We call this algorithm `Guru`. Guruswami et al. [5] gave the following example to show that their heuristic algorithm does not yield an approximation ratio better than $O(\log n)$.

Example 21. *[5] Let $m := n$, $N_i := 1, \forall i \in \{1, 2, \ldots, n\}$.*

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

Our variant of `DK93`, `Global-DK`, when run on the above example, achieves profit $2 - \frac{1}{n}$ out of the possible $\sum_{i=1}^{n} \frac{1}{i}$. Therefore, it cannot have better than $O(\log n)$ approximation ratio. We can modify Guruswami et al.'s example by increasing all the reservation prices for product n by

a small $\epsilon > 0$. Then `MaxR` followed by `DK88` achieves a profit of $1 + n\epsilon$, so this algorithm cannot hope for better than $O(\log n)$ approximation ratio in the worst-case either.

Next, we show that when the population sizes, N_i, are not necessarily all the same, the worst-case behaviour of the algorithm `Guru` get worse.

Example 22. *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 populations N_i to $2^{n-1}, 2^{n-2}, \ldots$.

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 23. *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 24. *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 25. *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 $\bar{R}_i$):

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

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. □

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

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

7 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."

7.1 Basic Experiments

We compare the following heuristics:

- `Guru`: The algorithm described in Section 6 based on the heuristic by Guruswami et al. [5].
- $\overline{\texttt{Guru}}$: 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.
- $\overline{\texttt{Guru}}$ `GRH-SubTree`: Initialize with $\overline{\texttt{Guru}}$ 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 7.1. We used the optimal objective value of the following (lifted) LP relaxation:

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

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

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 [11]. Our lifted formulation above unifies such strategies and it is at least as strong as any LP relaxation resulting from such aggregation strategies.

We see from the results in the Table 7.1 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, in other cases, `Fast-DK` did very well. Overall, our algorithms based on Generalized Reassignment Heuristic, $\overline{\texttt{Guru}}$ `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 $\overline{\texttt{Guru}}$ can serve as a robust initialization heuristic for optimal pricing problems.

7.2 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 $n \in \{200, 400, 600, 1000\}$.

n	m	Guru	$\overline{\text{Guru}}$	MaxR Global-DK	MaxR Fast-DK	$\overline{\text{Guru}}$ 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% (0.000 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% (0.001 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.

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 7.2.

It may seem implausible that, in practice, one would try to hawk tens of thousands of distinct but interchangeable products at a large number of distinct customer segments whose preferences can be determined to such precision that high confidence could be placed in a reservation price matrix. This is indeed implausible, but being able to solve (even heuristically) large pricing problems can make various modelling tricks practical.

The large values of m can be explained by a need to treat product bundling decisions optimally (for a classical MIP approach see [7]). In many applications, a company may offer only 20-100 products; however, the company may also offer bundles of these products. (Indeed, declaring the price of a bundle as the sum of the prices of its ingredients rarely leads to an optimal solution.) Our framework can handle bundling by treating each bundle as a new 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. Clearly, 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 7.2 suggest that the iterated operator part of the heuristic $\overline{\texttt{Guru}}$ 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, $\overline{\texttt{Guru}}$ can improve the objective value as much as 23% over

n	m	$\overline{\texttt{Guru}}$		MaxR Global-DK		MaxR Fast-DK		$\overline{\texttt{Guru}}$ 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	100.0000	(.597 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	100.0085	(3.227 h)	104.4826	(23.994 h)	47.8095	(21.661 h)	105.0925	(23.994 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	111.4507	(.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. Table: October 3, 2012

that of `Guru` objective value. We see that the behaviours of the algorithms that we noticed during our basic experiments (see Table 7.1) are still present in these large-scale experiments and they are only magnified. A very promising result is that our most sophisticated heuristics `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.

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

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

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

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

[5] 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.

[6] O. Günlük. A pricing problem under Monge property. *Disc. Optim.*, 5:328–336, 2008.

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

[8] 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.

[9] 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.

[10] R. Shioda, L. Tunçel and B. Hui. Applications of deterministic optimization techniques to some probabilistic choice models for product pricing using reservation prices, *Research Report* CORR 2007-02, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, February 2007 (revised: February 2009)

[11] 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.

[12] 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.