**MANAGEMENT SCIENCE**
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0025-1909 | EISSN 1526-5501 | 00 | 0000 | 0001

**INFORMS**
DOI 10.1287/xxxx.0000.0000
© 0000 INFORMS

# Data-driven assortment optimization

Dimitris Bertsimas
Sloan School of Management and Operations Research Center, E40-147, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139, USA; dbertsim@mit.edu

Velibor V. Mišić
Operations Research Center, E40-149, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139, USA; vvmisic@mit.edu

Assortment optimization refers to the problem of selecting a set of products to offer to a group of customers so as to maximize the revenue that is realized when customers make purchases according to their preferences. Assortment optimization is essential to a wide variety of application domains that includes retail, online advertising and social security; however, it is challenging in practice because one typically has limited data on customer choices on which to base the decision. In this paper, we present a two-step approach to making effective assortment decisions from transaction data. In the first step, we use the data to estimate a generic ranking-based model of choice that is able to represent any choice model based on random utility maximization. In the second step, using the estimated model, we find the optimal assortment by solving a mixed-integer optimization (MIO) problem that is scalable and that is flexible, in that it can easily accommodate constraints. We show through computational experiments with synthetic data that (1) our MIO model is practically tractable and can be solved to full optimality for large numbers of products in operationally feasible times; (2) our MIO model is able to accommodate realistic constraints with little impact to solution time; (3) our estimation procedure is computationally efficient and produces accurate out-of-sample predictions of the true choice probabilities; and (4) by combining our estimation and optimization procedures, we are able to find assortments that achieve near-optimal revenues that outperform alternative parametric and non-parametric approaches.

*Key words*: choice modeling; assortment optimization; large-scale linear optimization; mixed-integer optimization.
*History*: This paper was first submitted on June 5, 2015.

## 1. Introduction

A ubiquitous element of business is that of making an assortment decision, which can be described most generally as follows. A firm offers a set of products (*an assortment*) to a group of individuals. The individuals possess preferences over the different products, which may vary from individual to individual, and proceed to select the product that they most prefer; the firm then garners some revenue from the choices of the individuals. The problem of *assortment optimization* is to decide what set of products, taken from a larger set of possible products, should be offered so as to maximize the firm's expected revenue when customers exercise their preferences.

Assortment optimization problems arise in many contexts:

• **Retail**. As a concrete example, consider a grocery store. A grocery store has many different product categories (e.g., coffee, soft drinks, breakfast cereals and so on). Within each category, the grocery store has many different individual products that could be stocked on its shelves (e.g., for coffee, the grocery store might have different roasts of different brands in different sizes). The grocery store must decide which ones to stock on its shelves so as to maximize its expected revenue per customer.

• **Online advertising**. In online advertising, part of a webpage is devoted to displaying ads that are managed by an advertising platform (such as Google's AdSense system). The advertising platform has limited display space and must decide which ads to display so as to maximize the expected revenue (paid by the advertiser to the platform) per page view.

• **Social security**. As part of its social security system, a government may offer its citizens a number of funds that they can invest in for retirement. These funds may vary with respect to the type of securities, the industries represented in those securities and their overall risk-return performance. The government must then decide which funds to allow its citizens to invest in so as to ensure the maximum average return for its population while limiting the risk to the total investment of its citizens. A similar problem is faced by private employers who must decide on the assortment of 401(k) retirement savings plan to offer to their employees.

Such decisions have immense impact. In the retail domain, Fisher and Vaidyanathan (2012) describe how the grocery store Super Fresh stopped carrying certain dry grocery items in order to make room for fresh offerings, which lead to customers taking their business to other stores and ultimately, to the store declaring bankruptcy. In social security, Thaler and Sunstein (2008) describe how the Swedish government in 2000 undertook an effort to privatize and reform the country's social security system. The government allowed private funds meeting certain criteria to be offered and encouraged its citizens to choose from among the resulting 456 different funds. The authors compare those who actively selected their portfolios with those who simply invested in a carefully designed default fund. The authors write:

> [Those] who selected portfolios for themselves selected a higher equity exposure, more active management, much more local concentration, and higher fees. ... Because of the decline in the market that followed the launch of this plan, investors did not do well for the first three years (from October 31, 2000, through October 31, 2003), but those who invested in the default fund suffered less. The default fund lost 29.9 percent in those three years, while the average portfolio of those participants who picked their funds actively lost 39.6 percent. ... Through

July 2007 the default fund is up 21.5 percent while the average actively managed portfolio is up only 5.1 percent. (Thaler and Sunstein 2008)

The importance of assortment decisions is also underscored by the numerous commercial planning tools that are available or under development such as IBM's DemandTec® Assortment Optimization tool (IBM 2015), JDA Assortment Optimization (JDA Software Group, Inc. 2015), Oracle Retail (Oracle Corporation 2015) and Celect's Choice Engine (Celect, Inc. 2014).

Assortment decisions in practice are difficult for a number of reasons:

1. **Choice modeling**. Most assortment decisions are affected by *demand substitution*. Each customer has preferences for the different products, and would ideally want to be able to choose the product from the universe of products that he prefers the most. However, if this product is not present in the assortment, he will select the product in the assortment that is most preferable to him. To model demand with substitution, there exists a wide variety of discrete choice models that may be used, such as the multinomial logit (MNL), nested logit (NL) and mixed multinomial logit (MMNL) models (Ben-Akiva and Lerman 1985). This poses a challenge because the decision maker is faced with the question of which model to use. If the model is too simple, it may underfit the available data. If it is too complex, it may overfit the available data, leading to inaccurate predictions.

2. **Tractability**. Although in some cases the underlying choice model leads to an assortment optimization problem that can be easily solved, typically the resulting problem is difficult from a computational complexity standpoint. There is also often an undesirable tradeoff between predictive accuracy and tractability: simple but inaccurate choice models lead to easy optimization problems, while complex choice models lead to intractable problems.

3. **Constraints**. In many assortment contexts, the firm may have various business rules that limit the possible assortments. A basic constraint is that of capacity; for instance, a retailer may have limited shelf space, and an advertiser may have limited space on a webpage to fit ads. However, there may be many other constraints arising from the firm's internal rules that could affect assortment decisions. For example, a retailer may require that some products be offered together (e.g., if brand X dark roast coffee is offered, then brand Y dark roast coffee must also be offered), that only some number of products within a certain subclass of products is offered (e.g., of all dark roast coffee brands, offer at most three) or require at least some number of products within a given subset of products to be offered (e.g., offer at least two medium roast coffee products).

In this paper, we propose a new approach for making assortment decisions that directly addresses the above challenges. The approach is based on modeling the choice behavior of the market through a generic ranking-based model of choice, and then using this ranking-based choice model to formulate and solve a mixed-integer optimization (MIO) problem that directly yields the optimal assortment decision. We make the following specific contributions:

1. We present a new approach for making assortment decisions based on modeling and solving the problem as an MIO problem. The optimization problem assumes a generic non-parametric choice model where the market is represented by a probability distribution over a finite, fixed collection of rankings over the products; such a model is able to capture any choice model based on random utility maximization, and can be readily formulated in an MIO framework. The optimization problem is efficient from a modeling perspective, as the number of binary variables scales with the number of products and the problem possesses desirable structure that facilitates its solution by standard solution techniques (specifically branch-and-bound). Using standard MIO modeling techniques, the problem also readily accommodates constraints.
2. We present an associated procedure for estimating the ranking-based choice model required by our model. The approach is based on solving a large-scale linear optimization (LO) problem that aims to minimize the $\ell_1$ error between the choice probabilities predicted by the model and the actual choice probabilities that are given by the data. We propose an efficient solution procedure based on column generation to find a finite collection of rankings and an associated probability distribution over them.
3. We demonstrate the effectiveness of our approach computationally:

    (a) We show that our assortment optimization model is practically efficient. Using commercial solvers, we show that we are able to solve very large instances of the basic assortment optimization problem with large numbers of products and large numbers of rankings comprising the underlying choice model *to full optimality.* Moreover, we show that in a large number of cases, the solution of the LO relaxation is integral, and when it is not, the relaxation gap is very small.

    (b) We show that constraints are easily modeled in our framework and have minimal impact on the practical efficiency of our model. To evaluate the benefit of our MIO framework, we adapt a local search procedure that delivers excellent practical performance on "simple" problems, namely unconstrained and cardinality constrained assortment problems, for more complex assortment optimization cases. We show that our MIO approach outperforms this local search procedure in these more complex cases.

    (c) We show that our estimation approach is tractable and yields accurate ranking-based models. To demonstrate this, we generate transaction data according to a known choice model, estimate our ranking-based model using a portion of the data and evaluate the accuracy of the estimated model on the remainder of the data. We show that our approach becomes more accurate with more data, is relatively resistant to overfitting and makes more accurate revenue predictions than a simple parametric model and a state-of-the-art non-parametric approach.

    (d) We show that combining our estimation and optimization approaches leads to effective decisions. To demonstrate this, we apply our estimation procedure to a set of training data, solve our MIO problem to obtain an assortment and then compare the *true* revenue of this assortment, i.e., under the same model that generated the data, to the best possible revenue under the same model. We show that our combined approach leads to revenues that are only a few percent from the optimal, that improve with more data and that are higher than revenues obtained from optimizing the aforementioned alternative parametric and non-parametric models.

The rest of this paper is organized as follows. In Section 2, we review the relevant literature in choice modeling and assortment optimization. In Section 3, we present our modeling framework by describing the underlying ranking-based choice model and our MIO model for making assortment decisions given such a choice model. In Section 4, we present our estimation procedure for producing a ranking-based choice model from transaction data. In Section 5, we report on the results of our computational experiments. Lastly, in Section 6, we summarize our findings and propose directions for future research.

## 2. Literature review

Assortment optimization is a central topic in the operations management research literature; for a comprehensive overview of the topic, we refer the reader to the book chapter of Kök et al. (2009). We divide our discussion of how this work fits into the existing research landscape according to two areas: choice modeling and optimization.

**Choice modeling**. The primary prerequisite for assortment optimization is a choice model that specifies, for a given set of products, how frequently customers will select each product in the set. There exists a variety of choice models one could apply; the reader is referred to the excellent literature review of Farias et al. (2013) for a concise overview and the books of Ben-Akiva and Lerman (1985) and Train (2009) for a more detailed treatment. One of the most basic parametric choice models is the multinomial logit model (MNL) model. Although the MNL model is widely used, it exhibits certain undesirable properties, such as the independence of irrelevant alternatives (IIA) property (Ben-Akiva and Lerman 1985). As a result, more complex parametric models such as the mixture of multinomial logits (MMNL) model and the nested logit (NL) model have been proposed.

While parametric models have proven to be accurate in many applications, it is often not easy to decide on which parametric model to use. A simple model such as the MNL model may underfit the data and ignore important choice phenomena in the data, potentially leading to poor predictions of choice frequencies and suboptimal assortment decisions. At the same time, a more complex model such as the NL model may overfit the data and also lead to poor predictions and suboptimal assortment decisions.

As an alternative to parametric models, a number of generic choice models have been proposed that make minimal structural assumptions and that are capable of representing a wide variety of choice models. In particular, Farias et al. (2013) proposed a general model of choice where one represents choice behavior by a probability distribution over all of the possible rankings of the products. For a given assortment, Farias et al. (2013) then propose predicting the expected revenue of the assortment by computing the worst-case expected revenue, where the worst-case is taken over all probability distributions that are consistent with the available transaction data. They show, using both synthetic and real data, that their revenue predictions are more accurate than those produced by parametric models such as MNL and MMNL. Another general model that has recently been proposed is the Markov chain model of customer choice (Blanchet et al. 2013). In this model, products are modeled as states in a Markov chain and substitution behavior is modeled by transitions in the same Markov chain; the choice probabilities for a given offer set can then be computed as the absorption probabilities of the products in the offer set. They show that such a model provides a good approximation to any choice model based on random utility maximization. Note, however, that the Markov chain choice model as presented in Blanchet et al. (2013) can only be estimated when one has historical data corresponding to a specific set of $n+1$ assortments, where $n$ is the number of products; the estimation of the model when one has an arbitrary collection of historical assortments remains an open problem.

The assortment optimization approach that we present in this paper builds upon the framework of Farias et al. (2013) in that we also consider a probability distribution over rankings, but there is a critical difference. As mentioned above, Farias et al. (2013) is concerned with predicting revenues by evaluating the worst-case revenue over all probability distributions that reconcile the available data. In contrast, our approach does not consider the worst case. Rather, we fix a small set of rankings over the products and a probability distribution over this small set of rankings. This is important for three reasons. The first reason is that revenue predictions under our framework are considerably simpler to compute than in the worst-case approach. In our framework, for a given set of products, one simply computes the most preferred product under each ranking and sums the corresponding revenues weighted by the probability distribution. On the other hand, evaluating the worst-case revenue when the support of the probability distribution is the set of *all* possible ranking is more challenging, as one has to solve an LO problem with an intractable number of decision variables (one per ranking of products; for $n$ products and a single no-purchase alternative, this results in $(n+1)!$ decision variables) (Farias et al. 2013). The second reason, which is related to the first, is that in our framework, the corresponding assortment optimization problem can be compactly formulated as an MIO whose size scales gracefully in the number of products and the number of rankings. In contrast, in the worst-case framework, it is not clear how one may formulate the problem of optimizing the worst-case revenue as an efficiently solvable mathematical optimization formulation. The most promising proposal for solving this problem is the ADXOpt algorithm, proposed by Jagabathula (2014), which is a local search algorithm and cannot guarantee global optimality. The third and final reason is that there are substantial differences in predictive accuracy. As we will show later, fixing a set of rankings and a probability distribution over them leads to more accurate predictions of revenue than the worst-case approach (Section 5.4), and ultimately leads to better decisions (Section 5.6).

This conceptual difference between our work and the framework of Farias et al. (2013) leads to the question of estimation: given some transaction data, what is a set of rankings and an associated probability distribution that would give rise to this data? In response, we propose a simple estimation procedure for identifying such a probability distribution and a set of rankings. Our procedure is based on solving an $\ell_1$ approximation problem using column generation. To the best of our knowledge, the only other procedure for fitting this kind of choice model – a probability distribution over the set of rankings – is the approach of van Ryzin and Vulcano (2015). There are some similarities between our procedure and that of van Ryzin and Vulcano (2015); most notably, both we and van Ryzin and Vulcano (2015) propose a column generation step that involves solving essentially the same subproblem. However, the main difference between our proposal and that of van Ryzin and Vulcano (2015) is that we approach the model from an $\ell_1$ approximation perspective whereas van Ryzin and Vulcano (2015) use maximum likelihood estimation. This turns out to be important, because one can find optimal solutions to our problem that are basic feasible solutions, where only a small number of rankings have non-zero probability; this limits the complexity of the model that our procedure identifies and guards against overfitting. On the other hand, in the framework of van Ryzin and Vulcano (2015), one must perform some kind of model selection to prevent overfitting.

**Optimization**. Alongside the body of research devoted specifically to modeling choice, there is a rich literature that considers assortment optimization under these models – some examples include the MNL model (Talluri and van Ryzin 2004), the robust MNL model (Rusmevichientong and Topaloglu 2012), the NL model (Davis et al. 2014, Li et al. 2013), the MMNL model (Bront et al. 2009, Rusmevichientong et al. 2014) and the Markov chain choice model (Blanchet et al. 2013, Feldman and Topaloglu 2014). There is also research that considers assortment optimization under constraints such as cardinality constraints (Rusmevichientong et al. 2010, Gallego and Topaloglu 2014), capacity constraints (Désir and Goyal 2014, Gallego and Topaloglu 2014, Feldman and Topaloglu 2015) and other constraints such as precedence and quality consistency (Davis et al. 2013).

The majority of prior research on assortment optimization fits what can be summarized as a "fix-then-exploit" approach: one *fixes* a given parametric choice model and then *exploits* the structure of the resulting optimization problem to develop exact solution procedures, if possible, or otherwise heuristics or approximation algorithms. On the other hand, the approach we propose and advocate for is to estimate a ranking-based model of choice from the data and to formulate the assortment optimization problem as an MIO problem. We believe that our approach carries a number of advantages. First, in our approach, there is no "problem-specific" effort that is required: MIO problems can always be solved exactly via branch-and-bound to obtain a solution that is provably optimal or has a guarantee on its suboptimality (if the process is terminated early). Second, MIO as a methodology is highly flexible, in that (1) we can easily model a variety of business rules as linear constraints in the MIO problem, and (2) we can then simply "declare" these constraints to the solver, and let the solver do the work of accounting for them. In contrast, constraints are highly problematic in the fix-then-exploit approach, in that they necessitate entirely different algorithms relative to the unconstrained case. For example, the unconstrained MNL problem can be solved by enumerating the $n$ revenue-ordered subsets (Talluri and van Ryzin 2004); however, with capacity constraints, the problem becomes *NP-Hard* and one must consider more complex approximation algorithms (Désir and Goyal 2014).

Of special note is the ADXOpt algorithm, proposed by Jagabathula (2014). ADXOpt is a local search procedure that starts with the empty set and in each iteration, moves to a new assortment that most improves the expected revenue by considering additions of products outside of the assortment, deletions of products inside of the assortment and exchanges of products inside the assortment with ones outside. The procedure is general in that the only prerequisite for it is a function that maps an assortment to its expected revenue. Moreover, the procedure identifies the exact optimal assortment for certain choice models and delivers excellent performance in numerical experiments with other choice models (such as the model of Farias et al. 2013). Although this approach is simple, it is not designed to handle complex constraints, nor is it clear how to extend it to handle them. To demonstrate the impact of this, in Section 5.2 we consider a reasonable modification to ADXOpt to accommodate constrained problems – namely, to only consider local moves that maintain feasibility – and we show that for moderately constrained problems, ADXOpt can be significantly suboptimal compared to our MIO approach.

Outside of operations management, assortment optimization is connected to the problem of product line design (PLD), which is a central problem in marketing science. In the PLD problem, a firm seeks to introduce a product that has certain attributes to a market of heterogeneous customers. The firm must decide on which versions of the product, as defined by the attributes, to offer so as to maximize the revenue garnered from the customer's choices. This problem has been studied extensively and there exist many IO formulations of it (see the literature review of Bertsimas and Mišić 2014). The MIO formulation that we describe is most similar to the formulation of the PLD problem used in Belloni et al. (2008), which is described in the electronic companion of that paper. Our formulation refines the formulation of Belloni et al. (2008) by modeling a large number of variables as continuous rather than binary by exploiting a property of the constraints. However, the scales of the problems that we are solving are different and as such, our papers generate substantially different insights. The paper of Belloni et al. (2008) solved the PLD for a real problem involving 3584 candidate products and over 300 preference rankings of these products; even after implementing various sophisticated MIO enhancements, such as Lagrangian relaxation and valid inequalities, the problem required *over a week* to solve to full optimality and as such, it is not practical for product line design problems. However, in our computational experiments, the corresponding assortment optimization problems we are faced with are considerably smaller and as such, can be solved to full optimality within *seconds* and *without* additional enhancements of the kind used in Belloni et al. (2008).

## 3. Assortment optimization

In this section, we describe our assortment optimization method. We begin in Section 3.1 by describing our ranking-based choice model; then, in Section 3.2 we present our MIO formulation for finding the optimal assortment under this choice model.

### 3.1. Choice model

We begin by defining the non-parametric choice model that describes the choice behavior of the market. The model we will consider is conceptually similar to the model used in Farias et al. (2013); the reader is referred to Farias et al. (2013) for further details.

We assume that there are $n$ products, indexed from 1 to $n$. We use the index 0 to denote the no-purchase alternative (the possibility that the customer does not purchase any of the products that we offer). Together, we refer to the set $\{0, 1, 2, \dots, n\}$ – the set of products together with the no-purchase alternative – as the *options* that are available. We assume that a customer will select exactly one of the available options. We also assume that the no-purchase alternative 0 is always available to the customer.

We assume that we have $K$ rankings or permutations $\sigma^1, \dots, \sigma^K$ over the options $\{0, 1, 2, \dots, n\}$. Each permutation $\sigma^k : \{0, 1, 2, \dots, n\} \to \{0, 1, 2, \dots, n\}$ is a bijection that assigns each option to a rank in $\{0, 1, 2, \dots, n\}$. The value $\sigma^k(i)$ indicates the rank of option $i$; $\sigma^k(i) < \sigma^k(j)$ indicates that $i$ is more preferred to $j$ under the permutation $\sigma^k$. We can think of each $\sigma^k$ as a "mode" or "type" of choice behavior that a customer may follow. We assume that given a set of products

$S \subseteq \{1, 2, \ldots, n\}$, a customer that follows the permutation $\sigma^k$ will select the option $i$ from the set $S \cup \{0\}$ with the lowest rank, i.e., the option $\arg\min_{i \in S \cup \{0\}} \sigma^k(i)$.

We use $\lambda^k$ to denote the probability that a random customer makes a choice according to the permutation $\sigma^k$; we use $\boldsymbol{\lambda}$ to denote the probability mass function (PMF) over the set of permutations $\{\sigma^1, \ldots, \sigma^K\}$. For a given assortment of products $S \subseteq \{1, 2, \ldots, n\}$, the probability $\mathbb{P}(i \,|\, S)$ that a random customer selects option $i \in \{0, 1, 2, \ldots, n\}$ given that the available set of products was $S$ is given by

$$\mathbb{P}(i \,|\, S) = \sum_{k=1}^{K} \lambda^k \cdot \mathbb{I}\{i = \arg\min_{i' \in S \cup \{0\}} \sigma^k(i')\},$$

where $\mathbb{I}\{\cdot\}$ is the indicator function ($\mathbb{I}\{A\}$ is 1 if $A$ is true and 0 otherwise).

Let $r_i$ be the revenue of option $i$; we assume that the revenue $r_0$ of the no-purchase alternative is exactly zero. Then the expected revenue from offering the set of products $S$ is denoted by $R(S)$ and is given by

$$\begin{aligned} R(S) &= \sum_{i \in S} r_i \cdot \mathbb{P}(i \,|\, S) \\ &= \sum_{i \in S} r_i \cdot \left( \sum_{k=1}^{K} \lambda^k \cdot \mathbb{I}\{i = \arg\min_{i' \in S \cup \{0\}} \sigma^k(i')\} \right). \end{aligned}$$

## 3.2. Mixed-integer optimization model

Having defined the non-parametric choice model we will be using, we now turn our attention to how to make assortment decisions. The problem we wish to solve is to find the set of products $S^*$ that maximizes the expected revenue:

$$S^* = \arg\max_{S \subseteq \{1, \ldots, n\}} R(S).$$

We will formulate the problem as an MIO problem. For each product $i \in \{1, \ldots, n\}$, let $x_i$ be a binary decision variable that is 1 if product $i$ is included in the assortment, and 0 otherwise. For each option $i \in \{0, 1, \ldots, n\}$, let $y_i^k$ be a decision variable that is 1 if option $i$ is chosen under the $k$th permutation, and 0 otherwise.

The problem, in its most basic form, can then be formulated as follows.

$$\begin{aligned} \underset{\mathbf{x}, \mathbf{y}}{\text{maximize}} \quad & \sum_{k=1}^{K} \sum_{i=1}^{n} r_i \cdot \lambda^k \cdot y_i^k && \text{(1a)} \\ \text{subject to} \quad & \sum_{i=0}^{n} y_i^k = 1, \quad \forall\, k \in \{1, \ldots, K\}, && \text{(1b)} \\ & y_i^k \leq x_i, \quad \forall\, k \in \{1, \ldots, K\},\ i \in \{1, \ldots, n\}, && \text{(1c)} \end{aligned}$$

$$\sum_{j:\sigma^k(j)>\sigma^k(i)} y_j^k \leq 1 - x_i, \quad \forall\, k \in \{1,\ldots,K\},\; i \in \{1,\ldots,n\}, \tag{1d}$$

$$\sum_{j:\sigma^k(j)>\sigma^k(0)} y_j^k = 0, \quad \forall\, k \in \{1,\ldots,K\}, \tag{1e}$$

$$x_i \in \{0,1\}, \quad \forall\, i \in \{1,\ldots,n\}, \tag{1f}$$

$$y_i^k \geq 0, \quad \forall\, k \in \{1,\ldots,K\},\; i \in \{0,1,\ldots,n\}. \tag{1g}$$

In order of appearance, the constraints have the following meaning. Constraint (1b) ensures that under each ranking, exactly one choice is made. Constraint (1c) ensures that under ranking $k$, product $i$ can only be chosen if product $i$ is included in the assortment. Constraint (1d) ensures that, if product $i$ is included in the assortment, then none of the options that are less preferred to $i$ under ranking $\sigma^k$ may be chosen under ranking $k$ (the $y_j^k$ variables for all $j$ that are less preferred are forced to zero). Constraint (1e) is a similar constraint, but pertaining to the no-purchase option: those options that are less preferred to the no-purchase option 0 may not be selected and are forced to zero. The penultimate constraint ensures that the $x_i$'s are binary. The last constraint ensures that each $y_i^k$ may be nonnegative.

It is worth commenting on several important aspects of this formulation. First, note that the formulation does not require the $y_i^k$ variables to be binary. This is because for fixed binary values of $x_i$, constraints (1b)–(1e) ensure that the $y_i^k$ values are forced to the correct binary values. Thus, as a result, the formulation has only $n$ binary variables (the $x_i$ variables).

Second, note that the structure of the formulation allows for relatively efficient optimization via branch-and-bound. In particular, suppose that at some point in the branch-and-bound tree, we branch on variable $x_i$ and set it to 1. Then, constraint (1d) forces $y_j^k$ for all $j$ that are less preferred to $i$ to be zero. It is plausible that for many rankings $k$, there may be many options $j$ that are less preferred to $i$; as a result, branching up on the variable $x_i$ changes the revenue significantly, allowing for other nodes in the branch-and-bound tree to be pruned.

Finally, note that although we have formulated the problem in a rather basic way, we can expand it to incorporate different kinds of business requirements by adding constraints to the formulation. We provide some examples below.

1. **Lower and upper bounds on the size of the assortment.** If the set of possible products is large, the firm may not want to offer too many products; similarly, the firm may also want to avoid situations where it offers too few products. If $U$ and $L$ are upper and lower bounds on the number of products in the assortment, then this requirement can be modeled as follows:

$$L \leq \sum_{i=1}^{n} x_i \leq U.$$

2. **Subset constraints.** Similarly, the firm may have a requirement that out of a subset $S \subseteq \{1, \ldots, n\}$, of the products, at most $U_S$ products may be included and at least $L_S$ products must be included. This is also easily modeled; we have

$$L_S \leq \sum_{i \in S} x_i \leq U_S.$$

3. **Precedence constraints.** The firm may require that if a product $i$ is included, then a product $i'$ must be included. This is easily modeled as

$$x_i \leq x_{i'}.$$

## 4. Choice model estimation

In Section 3.1, we introduced a choice model that is defined by a set of $K$ permutations $\sigma^1, \ldots, \sigma^K$ and a probability distribution $\boldsymbol{\lambda} = (\lambda^1, \ldots, \lambda^K)$ over this set. A natural question to ask is: where do the permutations and the probability distribution come from? One answer to this question is to estimate this model directly from transaction data, which we now describe. The type of transaction data that we will assume is available will be conceptually similar to the data that is assumed in Farias et al. (2013); we refer the reader to that paper for additional detail.

We assume that from previous operations, we have data corresponding to $M$ assortments of products, $S_1, \ldots, S_M \subseteq \{1, 2, \ldots, n\}$. For each $m$ and each option $i$ in $S_m \cup \{0\}$, we assume that we have $v_{i,m}$ which is the probability with which customers selected option $i$ when the assortment $S_m$ was offered to them. We assume that $v_{i,m}$ is fully accurate and noiseless, i.e., $v_{i,m}$ is exactly equal to $\mathbb{P}(i \mid S_m)$ for each $m$ and each $i \in S_m \cup \{0\}$. We use $\mathbf{v}$ to denote the vector of $v_{i,m}$ values. We will also use $P$ to denote the number of $(i, m)$ pairs; $\mathbf{v}$ is then a vector of dimension $P$.

Let us assume, for a moment, that we consider the entire set of $(n+1)!$ permutations on the total set of options $\{0, 1, 2, \ldots, n\}$; that is, we set $K = (n+1)!$ and allow the collection $\sigma^1, \ldots, \sigma^K$ to correspond to all of the possible permutations of the options. Then, as in Farias et al. (2013), we can define the matrix $\mathbf{A}$ as the matrix of $A^k_{i,m}$ values, where

$$A^k_{i,m} = \begin{cases} 1, & \text{if } i = \arg\min_{j \in S_m \cup \{0\}} \sigma^k(j), \\ 0, & \text{otherwise.} \end{cases}$$

In words, $A^k_{i,m}$ is 1 if under permutation $k$, the customer would have chosen option $i$ from the set of options $S_m \cup \{0\}$, and is 0 otherwise. The matrix $\mathbf{A}$, vector $\mathbf{v}$ and probability mass function $\boldsymbol{\lambda}$ are related in that $\boldsymbol{\lambda}$ must satisfy the linear system of equations

$$\mathbf{A}\boldsymbol{\lambda} = \mathbf{v}.$$

Therefore, one way that we may identify $\boldsymbol{\lambda}$ is to solve a problem where we minimize the $\ell_1$ error between $\mathbf{A}\boldsymbol{\lambda}$, which is the vector of predicted choice probabilities for the options in the assortments $S_1, \ldots, S_M$, and $\mathbf{v}$, which is the vector of actual choice probabilities for those same options. Mathematically, this problem can be stated as follows:

$$\begin{aligned} \underset{\boldsymbol{\lambda}}{\text{minimize}} \quad & \|\mathbf{A}\boldsymbol{\lambda} - \mathbf{v}\|_1 && \text{(2a)} \\ \text{subject to} \quad & \mathbf{1}^T \boldsymbol{\lambda} = 1, && \text{(2b)} \\ & \boldsymbol{\lambda} \geq \mathbf{0}. && \text{(2c)} \end{aligned}$$

Through standard LO modeling techniques (see, e.g., Bertsimas and Tsitsiklis 1997), this problem can be reformulated into the following LO problem:

$$\begin{aligned} \underset{\boldsymbol{\lambda}, \boldsymbol{\epsilon}^+, \boldsymbol{\epsilon}^-}{\text{minimize}} \quad & \mathbf{1}^T \boldsymbol{\epsilon}^+ + \mathbf{1}^T \boldsymbol{\epsilon}^- && \text{(3a)} \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{\lambda} + \boldsymbol{\epsilon}^+ - \boldsymbol{\epsilon}^- = \mathbf{v}, && \text{(3b)} \\ & \mathbf{1}^T \boldsymbol{\lambda} = 1, && \text{(3c)} \\ & \boldsymbol{\lambda}, \boldsymbol{\epsilon}^+, \boldsymbol{\epsilon}^- \geq \mathbf{0}. && \text{(3d)} \end{aligned}$$

At first glance, the problem that we have arrived at may not appear to be useful. In particular, the number of $\lambda^k$ variables is $(n+1)!$ – for even the smaller values of $n$ that may occur in practice, the resulting number of variables is too large for the problem to be directly solved by off-the-shelf solvers. The value of modeling the estimation problem in this way is that we have cast it as a large-scale LO problem where the columns of the $\lambda^k$ variables follow a specific combinatorial structure. Thus, we are able to apply column generation to solve the problem efficiently.

At a high level, our column generation procedure operates as follows. At each iteration, we maintain a collection of permutations $\sigma^1, \ldots, \sigma^K$. We refer to problem (3) with the full set of $(n+1)!$ permutations as the *master* problem, and with the restricted collection $\sigma^1, \ldots, \sigma^K$ of permutations as the *restricted master* problem. We begin the procedure with no permutations. We solve the corresponding restricted master. From the resulting optimal solution, we solve the subproblem to identify the new permutation $\sigma$ to add to $\sigma^1, \ldots, \sigma^K$ with the lowest reduced cost. If the optimal value of the subproblem is negative, we add the corresponding permutation to $\sigma^1, \ldots, \sigma^K$, update $K$ and go back and solve the new restricted master problem again; otherwise, if the optimal value of the subproblem is nonnegative, we terminate the procedure.

We now discuss the structure of the subproblem. Let $\boldsymbol{\alpha}$ and $\nu$ be the dual variables corresponding to constraints (3b) and (3c), respectively. We seek to find the permutation $\sigma$ and the corresponding column $\mathbf{a}$ of the matrix $\mathbf{A}$ such that the reduced cost of the corresponding $\lambda$ is as negative as possible. Recall that for a permutation $\sigma$, the column $\mathbf{a}$ is defined component-wise as $a_{i,m} = 1$ if under permutation $\sigma$, option $i$ is selected from the set of options $S_m \cup \{0\}$ and 0 otherwise. To model the permutation $\sigma$, we introduce a binary variable $z_{ij}$ for each distinct $i$ and $j$ in $\{0,1,2,\ldots,n\}$ that takes the value 1 if under $\sigma$, $i$ is preferred to $j$ ($\sigma(i) < \sigma(j)$) and 0 otherwise. The subproblem can then formulated as an integer optimization (IO) problem:

$$\begin{aligned}
\underset{\mathbf{z},\mathbf{a}}{\text{minimize}} \quad & 0 - \boldsymbol{\alpha}^T \mathbf{a} - \nu && \text{(4a)}\\
\text{subject to} \quad & a_{i,m} \le z_{ij}, \quad \forall\, m \in \{1,\ldots,M\},\ i,j \in S_m \cup \{0\},\ i \neq j, && \text{(4b)}\\
& z_{ij} + z_{ji} = 1, \quad \forall\, i,j \in \{0,1,2,\ldots,n\},\ i \neq j, && \text{(4c)}\\
& z_{ij} + z_{jk} - 1 \le z_{ik}, \quad \forall\, i,j,k \in \{0,1,2,\ldots,n\}, && \\
& \qquad\qquad\qquad\qquad i \neq j,\ i \neq k,\ j \neq k, && \text{(4d)}\\
& z_{ij} \in \{0,1\}, \quad \forall\, i,j \in \{0,1,2,\ldots,n\},\ i \neq j, && \text{(4e)}\\
& a_{i,m} \in \{0,1\}, \quad \forall\, m \in \{1,\ldots,M\},\ i \in S_m \cup \{0\}. && \text{(4f)}
\end{aligned}$$

In order of appearance, the constraints have the following meaning. Constraint (4b) links the values in the column $\mathbf{a}$ with the values of the permutation; in particular, if $a_{i,m} = 1$, then it must be that $z_{ij} = 1$ for every other $j$ in $\mathcal{S} \cup \{0\}$. Constraint (4c) represents non-reflexivity: either $i$ is ranked lower than $j$ or $j$ is ranked lower than $i$. Constraint (4d) represents transitivity: for any three distinct options $i$, $j$ and $k$, it must be that if $i$ is ranked lower than $j$ and $j$ is ranked lower than $k$, then $i$ is ranked lower than $k$. The objective function corresponds to the reduced cost of the permutation encoded by the $\mathbf{z}$ variables.

The full column generation procedure is presented as Algorithm 1.

We conclude our discussion of our estimation procedure by discussing two practical modifications that we will employ in our numerical experiments in Section 5. First of all, the current stopping criterion ensures that we solve the problem to full optimality; this happens when the objective $\|\mathbf{A}\boldsymbol{\lambda} - \mathbf{v}\|_1 = 0$, or equivalently, $\boldsymbol{\lambda}$ exactly solves $\mathbf{A}\boldsymbol{\lambda} = \mathbf{v}$. Alternatively, we may be satisfied with an approximate solution to this system of equations. To obtain an approximate solution, we may consider terminating when the objective value of the restricted master problem is close enough to zero, i.e., when $\|\mathbf{A}\boldsymbol{\lambda} - \mathbf{v}\|_1 \le P \cdot \epsilon$ for some $\epsilon > 0$. Observe that the quantity $\|\mathbf{A}\boldsymbol{\lambda} - \mathbf{v}\|_1 / P$ can be interpreted as the mean absolute error (MAE) of the current solution $\lambda^1, \ldots, \lambda^K$, $\sigma^1, \ldots, \sigma^K$ on the training set; thus, $\epsilon$ represents the training set MAE that we wish to achieve.

Second of all, note that although the subproblem (4) is an IO problem, it is not necessary to solve it as such. In particular, any solution $(\mathbf{z}, \mathbf{a})$ that is feasible for problem (4) and achieves a negative objective value corresponds to a permutation $\sigma$ whose $\lambda$ variable may enter the basis. Thus, rather than solving subproblem (4) exactly, we may opt to solve it approximately via a local

**Algorithm 1** Column generation algorithm.

```
Require: Choice probability vector $\mathbf{v}$, training assortments $S_1, \ldots, S_M$.
  Initialize $K$ to 0.
  Set $\mathbf{A}$ to be a $P \times 0$ (empty) matrix.
  Solve restricted master problem (3) with $\mathbf{A}$ to obtain dual variable values $\boldsymbol{\alpha}$ and $\nu$.
  Solve subproblem (4) with $\boldsymbol{\alpha}$, $\nu$ to obtain $\mathbf{z}$, $\mathbf{a}$.
  while $-\boldsymbol{\alpha}^T \mathbf{a} - \nu < 0$ do
    Update $K \leftarrow K + 1$.
    Set $\sigma^K$ as $\sigma^K(i) = \sum_{\substack{j=0 \\ j \neq i}}^{n} z_{ij}$.
    Set $\mathbf{A} \leftarrow [\mathbf{A}\ \mathbf{a}]$.
    Solve restricted master problem (3) with $\mathbf{A}$ to obtain primal variable values $\lambda^1, \ldots, \lambda^K$, dual
    variable values $\boldsymbol{\alpha}$ and $\nu$.
    Solve subproblem (4) with $\boldsymbol{\alpha}$, $\nu$ to obtain $\mathbf{z}$, $\mathbf{a}$.
  end while
  return $\sigma^1, \ldots, \sigma^K$ and $\lambda^1, \ldots, \lambda^K$.
```

search procedure. We consider a local search procedure that operates as follows. Starting from some initial (randomly chosen) permutation $\sigma$, we consider all neighboring permutations $\sigma'$ obtained by taking $\sigma$ and swapping the rankings of any two distinct options. We evaluate the reduced cost of each such $\sigma'$; if no neighboring $\sigma'$ improves on the reduced cost of $\sigma$, we terminate with $\sigma$ as the locally optimal solution. Otherwise, we move to the $\sigma'$ that most improves the reduced cost of $\sigma$ and repeat the procedure at this new permutation. If the locally optimal permutation does not have negative reduced cost, we can repeat the search starting at a new random permutation; we continue doing so until we find a locally optimal permutation with negative reduced cost or we have reached the maximum number of repetitions. Our preliminary experimentation with using this local search procedure within the column generation procedure suggested that it could find an approximate solution (satisfying $\|\mathbf{A}\boldsymbol{\lambda} - \mathbf{v}\|_1 \leq P \cdot \epsilon$) more rapidly than by solving problem (4) directly as an IO problem.

## 5. Computational results

In this section, we report on the results of our computational experiments. Our insights are as follows:

- In Section 5.1, we show that our MIO model is practically tractable. We show that it can be solved rapidly for large instances (large numbers of products and rankings), the LO relaxation provides a good approximation of the integer optimal value and that in a large proportion of instances, the LO is in fact integral.

- In Section 5.2, we show that constraints have a negligible impact on how efficiently problem (1) can be solved to full optimality. At the same time, we show that the ADXOpt local search procedure of Jagabathula (2014), which achieves strong performance in unconstrained and cardinality constrained problems, can be significantly suboptimal in the presence of complex constraints.
- In Section 5.3, we show that our estimation procedure can quickly obtain ranking-based models that yield accurate out-of-sample predictions of choice probabilities. We show that the procedure is relatively resistant to overfitting and learns more accurate models with more training data.
- In Section 5.4, we compare predictions of expected revenues from our approach to revenue predictions from (1) an MNL model fitted to the same data and (2) the worst-case approach of Farias et al. (2013). Although the MNL model is more accurate in some instances, it can be significantly less accurate than our approach when the underlying model is not an MNL model; moreover, even with additional data, it is not able to learn. The worst-case predictions using the approach of Farias et al. (2013) are in general much less accurate than those produced by our approach.
- In Section 5.5, we show that by combining our estimation and optimization procedures, we find assortments that achieve expected revenues within a few percent of the (unknown) true optimal revenue. As the amount of data increases, this optimality gap shrinks. Analogously to the lack of overfitting in prediction, we show that our combined estimation and optimization method is resistant to overfitting in optimization: as the number of column generation iterations increases, the optimality gap does not deteriorate but in fact improves.
- In Section 5.6, we compare our assortments against the fitted MNL assortments and the worst-case-optimal assortments (as determined by ADXOpt). We find that our approach is better over a wider range of models than the fitted MNL approach. We also find that our approach significantly outperforms the worst-case optimal approach.

We implemented our experiments in the Julia technical computing language (Bezanson et al. 2012). All mathematical optimization problems were modeled using the JuMP package for Julia (Lubin and Dunning 2015). All linear and mixed-integer linear optimization problems were solved using Gurobi 5.60 (Gurobi Optimization, Inc. 2015) and all nonlinear optimization problems were solved using IPOPT (Wächter and Biegler 2006).

## 5.1. Tractability of assortment optimization model

To test the tractability of the assortment optimization formulation (1), we consider the following experiment. For fixed values of the number of products $n$ and the number of permutations $K$, we randomly generate 100 instances, where we uniformly at random generate the set of rankings $\sigma^1, \dots, \sigma^K$ from the set of all possible permutations, the revenue $r_i$ of each product $i$ from the set

**Table 1** **Results of tractability experiment.**

| | | MIO | Rlx. | % | LO |
|---|---|---|---|---|---|
| $n$ | $K$ | Time | Gap (%) | Integral | Time (s) |
| 10 | 10 | 0.0 | 0.30 | 81.0 | 0.0 |
| 10 | 100 | 0.0 | 0.41 | 69.0 | 0.0 |
| 10 | 200 | 0.1 | 0.44 | 65.0 | 0.0 |
| 10 | 500 | 0.2 | 0.42 | 66.0 | 0.1 |
| 10 | 1000 | 0.4 | 0.44 | 63.0 | 0.2 |
| 20 | 10 | 0.0 | 0.11 | 85.0 | 0.0 |
| 20 | 100 | 0.1 | 0.60 | 55.0 | 0.0 |
| 20 | 200 | 0.2 | 0.48 | 46.0 | 0.1 |
| 20 | 500 | 0.8 | 0.62 | 42.0 | 0.2 |
| 20 | 1000 | 1.8 | 0.58 | 50.0 | 0.7 |
| 30 | 10 | 0.0 | 0.19 | 82.0 | 0.0 |
| 30 | 100 | 0.2 | 0.52 | 46.0 | 0.1 |
| 30 | 200 | 0.5 | 0.56 | 41.0 | 0.1 |
| 30 | 500 | 1.9 | 0.69 | 36.0 | 0.5 |
| 30 | 1000 | 5.2 | 0.88 | 29.0 | 1.6 |
| 40 | 10 | 0.0 | 0.11 | 75.0 | 0.0 |
| 40 | 100 | 0.4 | 0.45 | 43.0 | 0.3 |
| 40 | 200 | 1.1 | 0.69 | 33.0 | 0.2 |
| 40 | 500 | 3.6 | 0.89 | 23.0 | 0.8 |
| 40 | 1000 | 10.0 | 0.71 | 19.0 | 2.7 |

$\{1,\ldots,100\}$, and the probability distribution $\boldsymbol{\lambda}$ from the $(K-1)$-dimensional unit simplex. For each of these 100 instances, we solve both the LO relaxation and the actual MIO problem itself. We then record the average time to solve the MIO, the average time to solve the LO relaxation, the average relaxation gap (where the relaxation gap is defined as $100\% \times (Z_{LO} - Z_{MIO})/Z_{LO}$, where $Z_{LO}$ and $Z_{MIO}$ are the optimal values of the LO relaxation and the true MIO formulation, respectively) and the percentage of instances where the LO solution turned out to be integral.

Table 1 reports on the above metrics for different values of $n$ in $\{10,20,30,40\}$ and values of $K$ in $\{10,100,200,500,1000\}$. From Table 1, we can see that the MIO formulation is very tractable; in the largest collection of instances ($n=40$, $K=1000$), problem (1) can be solved in approximately 10 seconds on average. Moreover, the formulation is efficient, in the sense that the relaxation is a good approximation of the true integer formulation; for each value of $n$ and $K$, the average gap of the LO relaxation and the true MIO model is no more than 1%, and in a large number of cases (more than 20% for each combination of $n$ and $K$) the LO solution is integral.

## 5.2. Constrained assortment optimization

We next show the value of using our optimization model to accommodate constraints on the assortment. To do this, we consider the collection of instances from Section 5.1 corresponding to $n=30$ products and $K=100$ permutations. For each instance, we consider the corresponding assortment optimization problem with a randomly generated set of constraints. We solve the constrained assortment optimization problem exactly using our MIO model (1) and using the ADXOpt local search heuristic proposed by Jagabathula (2014). We adapt the ADXOpt local search heuristic to the constrained setting by only allowing additions, deletions or exchanges that ensure that the assortment will remain feasible at each iteration. We limit ADXOpt to one removal of each product.

We consider several different types of constraint sets:

1. **No constraints**. The corresponding problem is the unconstrained problem.

2. **Maximum subset**. A maximum subset constraint set is parametrized by an integer $C$ that specifies the number of constraints, an integer $B$ that specifies the number of products that participate in each constraint and an integer $U$ which specifies the maximum number of products we may select from the $B$ products. It has the form

$$\sum_{i=B(c-1)+1}^{Bc} x_{(i)} \leq U, \quad \forall\, c \in \{1,\ldots,C\},$$

where $(j)$ is the product with the $j$th highest marginal revenue. For example, with $C=2$, $B=3$ and $U=2$, the constraint set is

$$\begin{aligned} x_{(1)}+x_{(2)}+x_{(3)} &\leq 2, \\ x_{(4)}+x_{(5)}+x_{(6)} &\leq 2, \end{aligned}$$

which means that from among the top three products in marginal revenue, we may include at most two products, and from the next three products, we also may include at most two products. Such a constraint set could model a requirement that the assortment be diverse and not be biased towards certain groups of products.

3. **Precedence type 1**. A type 1 precedence constraint set is parametrized by an integer $C$ that specifies the number of constraints and an integer $B$ that specifies the number of products that participate in each constraint. It has the form

$$\sum_{j=B(c-1)+1}^{Bc} x_{(j)} \leq \sum_{j=Bc+1}^{B(c+1)} x_{(j)}, \quad \forall\, c \in \{1,\ldots,C\},$$

where, as for the maximum subset constraint set, $(j)$ indicates the product with the $j$th largest marginal revenue. For example, for $C=2$ and $B=3$, the constraint set is

$$\begin{aligned} x_{(1)}+x_{(2)}+x_{(3)} &\leq x_{(4)}+x_{(5)}+x_{(6)}, \\ x_{(4)}+x_{(5)}+x_{(6)} &\leq x_{(7)}+x_{(8)}+x_{(9)}, \end{aligned}$$

which means that the number of products we select from the top three products by marginal revenues (products (1) through (3)) cannot exceed the number of products we select from the next three products by marginal revenue (products (4) through (6)), which itself cannot exceed the number of products we select from the next three products by marginal revenue (products (7) through (9)). The constraint represents a form of precedence because including a product with a high marginal revenue mandates the inclusion of products with lower marginal revenues.

As with the maximum subset constraint, such a constraint could represent a business requirement that the firm must offer a diverse collection of products, and not offer just those products with the highest marginal revenue.

4. **Precedence type 2.** A type 2 precedence constraint set is parametrized by an integer $C$ that specifies the number of constraints and an integer $B$ that specifies the number of products that participate in each constraint. It has the form

$$\sum_{j \in J_c} x_j \leq (B-1) x_{j_c^*}, \quad \forall\, c \in \{1, \ldots, C\},$$

where for each $c$, $J_c$ is a subset of $\{1, \ldots, n\}$ of size $B-1$, and $j_c^*$ is a distinct product (i.e., not one of the products in $J_c$). In the experiments we will report on below, we randomly choose the set of products $J_c \cup \{j_c^*\}$ from $\{1, \ldots, n\}$ for each instance and each of the $C$ constraints.

In words, this constraint requires that if the assortment includes a product in $J_c$, then it must include the "main" product $j_c^*$. This type of constraint could represent a requirement that the firm offer certain items together. For example, the product $j_c^*$ could correspond to the "main" version of a product (e.g., regular Coca Cola) and the products in $J_c$ could correspond to alternative versions of this product (e.g., Diet Coke and Coke Zero) that can only be offered if the main product is offered.

Note that for all of these constraint sets, the empty assortment (in terms of the MIO formulation, this is represented by $x_i = 0$ for all $i \in \{1, \ldots, n\}$) is feasible and is still be used as the initial solution for ADXOpt, as originally described in Jagabathula (2014).

For each instance and each such constraint set, we add the constraint set to problem (1) and solve it to obtain the optimal value $Z^*_{MIO}$. We also apply ADXOpt to solve the constrained problem with the modifications described above; we let the $Z^*_{ADXOpt}$ denote the expected revenue of the ADXOpt assortment. For each instance and each constraint set, we compute the gap as $100\% \times (Z^*_{MIO} - Z^*_{ADXOpt})/Z^*_{MIO}$. We also record the time required to solve each instance under each constraint set for both the MIO and the ADXOpt approaches.

Table 2 reports the average gap of each constraint set, where the average is taken over the 100 instances, as well as the average time to solve the problem using our MIO approach and the ADXOpt approach. We can see that, in the absence of constraints, ADXOpt performs very well; the average optimality gap, over the 100 instances, is 0.07% and is effectively optimal. Similarly, for the

**Table 2 Results of constrained optimization comparison.**

| $n$ | $K$ | Constraint type | ADXOpt Gap (%) | MIO Time (s) | ADXOpt Time (s) |
|---|---|---|---|---|---|
| 30 | 100 | No constraints | 0.07 | 0.22 | 0.07 |
| 30 | 100 | Max. subset, $C = 2$, $B = 5$, $U = 3$ | 0.19 | 0.41 | 0.06 |
| 30 | 100 | Max. subset, $C = 2$, $B = 10$, $U = 3$ | 0.34 | 0.59 | 0.03 |
| 30 | 100 | Max. subset, $C = 3$, $B = 5$, $U = 3$ | 0.19 | 0.42 | 0.05 |
| 30 | 100 | Max. subset, $C = 3$, $B = 10$, $U = 3$ | 0.34 | 0.58 | 0.03 |
| 30 | 100 | Prec. type 1, $C = 1$, $B = 3$ | 0.72 | 0.21 | 0.06 |
| 30 | 100 | Prec. type 1, $C = 2$, $B = 3$ | 2.46 | 0.37 | 0.05 |
| 30 | 100 | Prec. type 1, $C = 1$, $B = 5$ | 1.92 | 0.37 | 0.06 |
| 30 | 100 | Prec. type 1, $C = 2$, $B = 5$ | 6.31 | 0.54 | 0.04 |
| 30 | 100 | Prec. type 1, $C = 1$, $B = 10$ | 7.44 | 0.58 | 0.03 |
| 30 | 100 | Prec. type 1, $C = 2$, $B = 10$ | 15.20 | 1.31 | 0.02 |
| 30 | 100 | Prec. type 2, $C = 3$, $B = 2$ | 0.47 | 0.23 | 0.07 |
| 30 | 100 | Prec. type 2, $C = 5$, $B = 2$ | 0.68 | 0.26 | 0.06 |
| 30 | 100 | Prec. type 2, $C = 10$, $B = 2$ | 1.17 | 0.25 | 0.04 |
| 30 | 100 | Prec. type 2, $C = 3$, $B = 5$ | 1.64 | 0.25 | 0.05 |
| 30 | 100 | Prec. type 2, $C = 5$, $B = 5$ | 1.98 | 0.21 | 0.03 |
| 30 | 100 | Prec. type 2, $C = 10$, $B = 5$ | 6.35 | 0.12 | 0.01 |

maximum subset constraint sets, ADXOpt also performs very well, with an average gap of no more than 0.34%. However, with more complex constraints – namely the type 1 and type 2 precedence constraint sets – ADXOpt is no longer able to ensure high quality solutions. For example, for the type 1 precedence constraint set, with just $C = 2$ constraints with $B = 3$ products, the average optimality gap is 2.46%. In other cases, it can be much higher (e.g., for type 1 precedence constraints with $C = 2$ constraints and $B = 10$ products, the gap is 15.20%). With regard to timing, note that both approaches are fast and their speed is relatively insensitive to the presence of constraints. These results suggest that, while ADXOpt delivers high quality solutions under simple constraints, it leaves value on the table in the presence of complicated constraints. On the other hand, the MIO approach allows the firm to capture all of this lost value, at negligible computational cost.

With regard to our comparison here, we wish to emphasize that ADXOpt was not originally designed to accommodate complex constraints. Given this, one may claim that the manner in which we have adapted ADXOpt to handle constraints is not the most appropriate; that one may further modify the procedure to attain better results. This is a fair criticism. The overall claim that we are making in this section is that our MIO modeling approach provides a simple and systematic way to accommodate a variety of constraints, allowing the user to obtain *provably* optimal solutions in operationally feasible times while *freeing the user* from the design of problem-specific algorithms. From this perspective, we do not believe that the aforementioned criticism invalidates our overall claim.

### 5.3. Estimation using column generation

To evaluate our estimation procedure, we proceed as follows. For different values of the number of products $n$, we consider different types of choice models. In particular, we consider the MMNL model with $T$ classes, for which the choice probability $\mathbb{P}(i \mid S)$ is given by

$$\mathbb{P}(i \mid S) = \sum_{t=1}^{T} p_t \cdot \frac{\exp(u_{t,i})}{\sum_{i' \in S} \exp(u_{t,i'}) + \exp(u_{t,0})}. \tag{5}$$

We randomly generate our MMNL models as follows. For each option $i$ and customer type $t$ we generate the values $q_{t,i}$ uniformly on $[0,1]$. For each customer type $t$, we then select four of the $n+1$ utilities $u_{t,1}, \dots, u_{t,n+1}$ randomly and set each chosen utility to $\log(Lq_{t,i})$, where $L$ is a predefined value, and the remaining utilities we set to $\log(0.1q_{t,i})$. The mixing probabilities $p_1, \dots, p_T$ are uniformly drawn from the $(T-1)$-dimensional unit simplex. We indicate these models by MMNL($L$, $T$); we consider $L \in \{5.0, 10.0, 100.0\}$ and $T \in \{1, 5, 10\}$.

For each number of products $n$ and each type of choice model – value of $T$ and $L$ – we generate 100 random instances. For each instance, we randomly generate two sets of 100 assortments and compute the choice probabilities under the ground truth model for each option in each assortment. We treat the first set of 100 assortments as training data that we may use to fit the model according to the estimation procedure in Section 4, and the second set of 100 assortments as testing data that we will use to evaluate the predictive accuracy of our estimated models. We vary the number of assortments $M$ that we use for training from the total 100 assortments and consider $M \in \{10, 20, 50, 100\}$. We run the estimation procedure and terminate it when the training mean absolute error (MAE) is below $10^{-3}$. To generate columns in the estimation procedure, we employ the local search procedure described in Section 4 with a maximum of ten repetitions; in all of our experiments, this was sufficient to find an entering column at each iteration. Using the resulting ranking-based model, we predict the choice probability of each option in each testing assortment, and compare each such probability to the corresponding true probability as per the ground truth model.

Table 3 reports the average training set and test set MAEs, where the average is taken over the 100 instances for each value of $n$ and each generating model, with a training set of $M = 20$ assortments. We also report the average time required to run the procedure in seconds ("Est. Time (s)"), the average number of iterations ("Num. Iter."), the average number of rankings in the final model with non-zero probability ($K$) and the average value of $P$ (recall that $P$ is the dimension of the $\mathbf{v}$ vector). From this table, we can see that although the test set MAE is higher than the training MAE by approximately an order of magnitude, it is still quite good (on the order of 0.01 – 0.02). Moreover, the time to run the procedure is quite modest (in the largest cases, those with $n = 30$, no more than one minute on average).

**Table 3 Results of estimation procedure.**

| $n$ | Generating model | $M$ | Train MAE | Test MAE | Est. Time (s) | Num. Iter. | $K$ | $P$ |
|---|---|---|---|---|---|---|---|---|
| 10 | MMNL(5.0, 1) | 20 | 0.0008 | 0.0174 | 0.68 | 105.0 | 65.0 | 120.3 |
| 10 | MMNL(5.0, 5) | 20 | 0.0010 | 0.0200 | 0.57 | 90.6 | 69.6 | 119.2 |
| 10 | MMNL(5.0, 10) | 20 | 0.0010 | 0.0197 | 0.53 | 86.4 | 72.5 | 120.7 |
| 10 | MMNL(10.0, 1) | 20 | 0.0009 | 0.0162 | 0.71 | 102.1 | 62.4 | 119.3 |
| 10 | MMNL(10.0, 5) | 20 | 0.0010 | 0.0185 | 0.59 | 94.8 | 69.2 | 120.8 |
| 10 | MMNL(10.0, 10) | 20 | 0.0010 | 0.0195 | 0.50 | 87.1 | 72.3 | 120.2 |
| 10 | MMNL(100.0, 1) | 20 | 0.0007 | 0.0165 | 0.79 | 107.4 | 59.2 | 119.3 |
| 10 | MMNL(100.0, 5) | 20 | 0.0009 | 0.0189 | 0.67 | 104.2 | 67.8 | 120.1 |
| 10 | MMNL(100.0, 10) | 20 | 0.0010 | 0.0209 | 0.56 | 96.6 | 72.9 | 120.7 |
| 20 | MMNL(5.0, 1) | 20 | 0.0009 | 0.0127 | 7.41 | 220.4 | 125.2 | 219.9 |
| 20 | MMNL(5.0, 5) | 20 | 0.0010 | 0.0154 | 4.09 | 168.4 | 124.9 | 221.1 |
| 20 | MMNL(5.0, 10) | 20 | 0.0010 | 0.0159 | 3.30 | 144.9 | 121.4 | 220.0 |
| 20 | MMNL(10.0, 1) | 20 | 0.0009 | 0.0129 | 9.42 | 264.5 | 127.2 | 219.6 |
| 20 | MMNL(10.0, 5) | 20 | 0.0010 | 0.0153 | 4.96 | 194.7 | 128.9 | 220.2 |
| 20 | MMNL(10.0, 10) | 20 | 0.0010 | 0.0169 | 3.71 | 162.1 | 127.2 | 220.9 |
| 20 | MMNL(100.0, 1) | 20 | 0.0008 | 0.0120 | 11.13 | 281.1 | 117.3 | 221.0 |
| 20 | MMNL(100.0, 5) | 20 | 0.0010 | 0.0151 | 7.53 | 244.4 | 126.6 | 220.9 |
| 20 | MMNL(100.0, 10) | 20 | 0.0010 | 0.0174 | 5.00 | 196.6 | 131.2 | 219.4 |
| 30 | MMNL(5.0, 1) | 20 | 0.0010 | 0.0104 | 23.49 | 311.8 | 178.3 | 320.5 |
| 30 | MMNL(5.0, 5) | 20 | 0.0010 | 0.0124 | 13.83 | 219.3 | 168.2 | 317.9 |
| 30 | MMNL(5.0, 10) | 20 | 0.0010 | 0.0129 | 10.76 | 181.8 | 156.1 | 319.9 |
| 30 | MMNL(10.0, 1) | 20 | 0.0010 | 0.0100 | 32.52 | 382.9 | 188.5 | 321.8 |
| 30 | MMNL(10.0, 5) | 20 | 0.0010 | 0.0126 | 16.49 | 254.1 | 171.6 | 317.6 |
| 30 | MMNL(10.0, 10) | 20 | 0.0010 | 0.0137 | 13.20 | 215.1 | 167.0 | 318.0 |
| 30 | MMNL(100.0, 1) | 20 | 0.0009 | 0.0093 | 47.79 | 473.9 | 174.2 | 320.2 |
| 30 | MMNL(100.0, 5) | 20 | 0.0010 | 0.0126 | 44.20 | 371.9 | 177.6 | 324.9 |
| 30 | MMNL(100.0, 10) | 20 | 0.0010 | 0.0142 | 25.20 | 283.3 | 175.6 | 319.1 |

We now turn our attention to the question of how the estimation procedure is affected by the amount of data. Table 4 shows how the accuracy and estimation time vary as the number of training assortments $M$ varies, with $n$ fixed to 30 and the utility scale parameter $L$ fixed to 5.0. (As in Table 3, the results are averaged over the 100 instances for each case.) As the amount of data increases, the complexity of the underlying models increases (as measured by the average of the number of permutations $K$ in the final model) and the test set MAE also decreases with additional data; in other words, the estimation procedure is able to learn more accurate models as the amount of data available increases. With regard to the time required to estimate the model, we find that even with 100 training assortments, the time required to run the procedure is still modest (no more than approximately five minutes on average).

Another interesting question to consider is how susceptible the procedure is to overfitting. To study this, we fix the number of training assortments $M$ to 20 and we consider those instances with

**Table 4 Results of estimation procedure as available data varies.**

| $n$ | Generating model | $M$ | Train MAE | Test MAE | Est. Time (s) | Num. Iter. | $K$ | $P$ |
|---|---|---|---|---|---|---|---|---|
| 30 | MMNL(5.0, 1) | 10 | 0.0009 | 0.0174 | 4.65 | 162.3 | 99.3 | 161.0 |
| 30 | MMNL(5.0, 1) | 20 | 0.0010 | 0.0104 | 23.49 | 311.8 | 178.3 | 320.5 |
| 30 | MMNL(5.0, 1) | 50 | 0.0010 | 0.0054 | 103.12 | 489.8 | 288.4 | 802.2 |
| 30 | MMNL(5.0, 1) | 100 | 0.0010 | 0.0036 | 302.39 | 637.0 | 388.0 | 1605.8 |
| 30 | MMNL(5.0, 5) | 10 | 0.0010 | 0.0192 | 3.42 | 123.5 | 96.8 | 158.7 |
| 30 | MMNL(5.0, 5) | 20 | 0.0010 | 0.0124 | 13.83 | 219.3 | 168.2 | 317.9 |
| 30 | MMNL(5.0, 5) | 50 | 0.0010 | 0.0065 | 72.23 | 373.5 | 289.0 | 795.9 |
| 30 | MMNL(5.0, 5) | 100 | 0.0010 | 0.0043 | 231.02 | 524.9 | 414.1 | 1597.8 |
| 30 | MMNL(5.0, 10) | 10 | 0.0010 | 0.0190 | 3.09 | 107.8 | 93.7 | 160.3 |
| 30 | MMNL(5.0, 10) | 20 | 0.0010 | 0.0129 | 10.76 | 181.8 | 156.1 | 319.9 |
| 30 | MMNL(5.0, 10) | 50 | 0.0010 | 0.0073 | 68.88 | 345.1 | 291.3 | 799.6 |
| 30 | MMNL(5.0, 10) | 100 | 0.0010 | 0.0047 | 225.90 | 493.9 | 422.4 | 1601.0 |

**Table 5 Results of estimation procedure as training MAE tolerance decreases.**

| Train MAE Tol. | Train MAE | Test MAE | Time (s) | Num. Iter. | $K$ |
|---|---|---|---|---|---|
| 0.1000 | 0.0941 | 0.0961 | 0.1 | 3.5 | 3.5 |
| 0.0500 | 0.0464 | 0.0532 | 0.3 | 10.7 | 10.7 |
| 0.0100 | 0.0098 | 0.0214 | 1.9 | 38.1 | 37.4 |
| 0.0050 | 0.0049 | 0.0172 | 3.7 | 67.3 | 64.0 |
| 0.0010 | 0.0010 | 0.0131 | 11.0 | 183.6 | 158.3 |
| 0.0005 | 0.0005 | 0.0124 | 15.4 | 241.8 | 198.7 |
| 0.0001 | $9.83 \times 10^{-5}$ | 0.0117 | 24.7 | 340.1 | 264.4 |
| $5.00 \times 10^{-5}$ | $4.83 \times 10^{-5}$ | 0.0116 | 27.4 | 363.3 | 278.6 |
| $1.00 \times 10^{-5}$ | $8.58 \times 10^{-6}$ | 0.0115 | 30.3 | 388.1 | 293.6 |
| $5.00 \times 10^{-6}$ | $3.98 \times 10^{-6}$ | 0.0115 | 30.7 | 391.9 | 296.6 |
| $1.00 \times 10^{-6}$ | $4.24 \times 10^{-7}$ | 0.0115 | 31.2 | 395.6 | 299.6 |

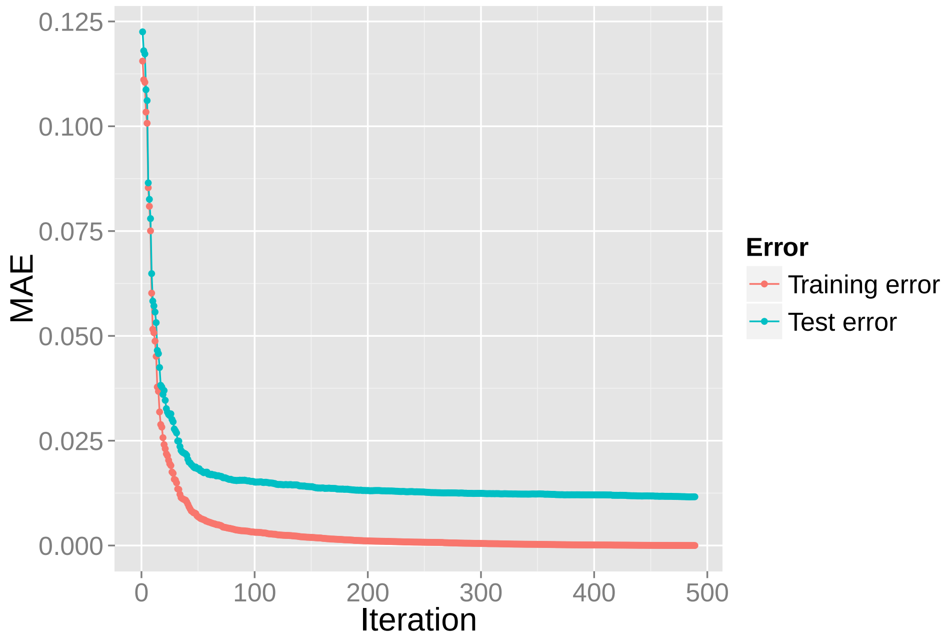
Results correspond to MMNL(5.0, 10) instances with $n = 30$ products and $M = 20$ training assortments.

$n = 30$, $T = 10$ and $L = 5.0$. We run the estimation procedure and vary the training MAE tolerance; for each threshold we choose, we save the model at termination (the set of permutations and the probability distribution), and measure the MAE of this model on the test set. Table 5 displays the results from this experiment, averaged over the 100 instances corresponding to each case. From this table, we can see that although the test set MAE is generally higher than the training set MAE, it does not increase when the training MAE is further decreased; rather, it appears to converge. Figure 1 plots how the training MAE and test MAE evolve for one instance with the number of iterations to further emphasize the above point.

## 5.4. Comparison of revenue predictions

We now compare the predictive power of our approach with that of two other approaches. In the first approach, we fit an MNL model to a training set of assortments using maximum likelihood

**Figure 1** **Evolution of training error and testing error with each column generation iteration for one MMNL instance with $n = 30$, $T = 10$, $L = 5.0$ and $M = 20$ training assortments.**



estimation and using this fitted model, we predict the expected revenue of each assortment in the test set. In the second approach, we consider the worst-case approach of Farias et al. (2013) with the same training set of assortments and use it to make predictions of the expected revenue on the test set of assortments. Our implementation of the worst-case approach uses randomized sampling as described in Farias et al. (2013). This approach involves randomly sampling a subset of size $K_{\text{sample}}$ of all of the possible permutations of the $n+1$ options and solving the sampled version of the worst-case problem. One of the difficulties of using sampling is that the sampled worst-case LO problem may turn out to be infeasible. To address this issue, we proceed as follows. For a given value of $K_{\text{sample}}$, we randomly sample $K_{\text{sample}}$ rankings, and check if the sampled problem is feasible. If it is feasible, we use that sample to make all worst-case predictions on the test set. If it is not feasible, we sample again at the current value of $K_{\text{sample}}$ and check again. If we encounter infeasibility after ten such checks, we move on to the next value of $K_{\text{sample}}$. The sequence of $K_{\text{sample}}$ values that our procedure scans through is

$$1 \times 10^4, 2 \times 10^4, \ldots, 9 \times 10^4,\ 1 \times 10^5, 2 \times 10^5, \ldots, 9 \times 10^5,\ 1 \times 10^6.$$

This approach lead to a feasible set of sampled permutations $\mathcal{K}_{\text{sample}}$ in all of our numerical comparisons. Our goal in applying this procedure was to ensure that we find a feasible set of permutations quickly.

In addition to these three approaches, we also consider another version of our column generation approach, which we refer to as the “averaged” column generation approach. Recall that our

**Table 6** **Results of revenue prediction comparison for $n = 20$ instances with $M = 20$ training assortments.**

| | | | CG | | AvgCG | | MNL | | WC | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Generating model | $M$ | AE | ME | AE | ME | AE | ME | AE | ME | AT (s) |
| 20 | MMNL(5.0, 1) | 20 | 1.72 | 13.07 | 1.16 | 8.28 | 0.00 | 0.00 | 3.83 | 16.93 | 0.56 |
| 20 | MMNL(5.0, 5) | 20 | 1.90 | 8.88 | 1.36 | 6.41 | 2.50 | 9.57 | 6.27 | 15.99 | 0.32 |
| 20 | MMNL(5.0, 10) | 20 | 1.77 | 7.40 | 1.29 | 5.22 | 2.05 | 7.91 | 11.83 | 23.19 | 0.44 |
| 20 | MMNL(10.0, 1) | 20 | 1.81 | 15.40 | 1.34 | 10.88 | 0.00 | 0.00 | 3.16 | 19.38 | 0.82 |
| 20 | MMNL(10.0, 5) | 20 | 1.98 | 9.69 | 1.51 | 7.35 | 2.79 | 11.64 | 4.78 | 15.16 | 0.36 |
| 20 | MMNL(10.0, 10) | 20 | 1.96 | 8.92 | 1.52 | 6.62 | 2.41 | 9.33 | 8.04 | 17.86 | 0.30 |
| 20 | MMNL(100.0, 1) | 20 | 1.82 | 16.41 | 1.36 | 12.27 | 0.00 | 0.00 | 2.61 | 21.19 | 2.64 |
| 20 | MMNL(100.0, 5) | 20 | 1.94 | 10.38 | 1.60 | 8.07 | 3.54 | 14.73 | 4.27 | 17.06 | 0.97 |
| 20 | MMNL(100.0, 10) | 20 | 2.07 | 9.19 | 1.71 | 7.25 | 2.88 | 11.49 | 5.75 | 15.61 | 0.39 |

implementation of our column generation procedure involves solving the subproblem using a local search, which starts from a random permutation. Given this, and given the fact that in general there may be many probability distributions $\boldsymbol{\lambda}$ that satisfy the linear system $\mathbf{A}\boldsymbol{\lambda} = \mathbf{v}$, repeated executions of the column generation procedure with the same data may produce different collections of rankings and probability distributions over them. Inspired by ensemble methods in statistical learning that combine multiple prediction methods into a single, more accurate prediction method (for further details see, e.g., Hastie et al. 2009), we consider averaging the probability distributions produced by these repeated runs. Specifically, given $\tau$ runs of the column generation procedure and the corresponding probability distributions $\boldsymbol{\lambda}^1, \dots, \boldsymbol{\lambda}^\tau$, we produce the averaged probability distribution $\boldsymbol{\lambda}^{\text{avg}}$ as

$$\boldsymbol{\lambda}^{\text{avg}} = \frac{1}{\tau} \sum_{i=1}^{\tau} \boldsymbol{\lambda}^i.$$

In our implementation of this averaged approach, we use the same column generation implementation as in Section 5.3 for $\tau = 10$ repetitions.

For each instance that we consider, we record the average absolute revenue error (AE) and maximum absolute revenue error (ME) over the assortments in the test set of that instance; we also record the average time (AT) in seconds to make a prediction, taken over the assortments in the test set. Table 6 reports the metrics for each prediction approach, averaged over the instances from Section 5.3 for $n = 20$ products and $M = 20$ training set assortments. (Note that we only report the AT metric for the worst case approach, as it was on the order of milliseconds for the other three approaches.) In the table and the discussion that follows, we use "CG", "AvgCG", "MNL" and "WC" to indicate the single pass column generation, averaged column generation, fitted MNL and worst-case methods, respectively.

With regard to MNL, we can see that when $T = 1$ – the ground truth model is a single-class MNL model – fitting an MNL model leads to essentially perfect predictions, as we would expect. However, for $T > 1$, the predictions become considerably less accurate and in particular, are less

**Table 7 Results of revenue prediction comparison between CG and MNL approaches for $n = 20$ instances with $L = 100.0$, $T \in \{5, 10\}$, as number of training assortments $M$ varies.**

| | | | CG | | MNL | |
|---|---|---|---|---|---|---|
| $n$ | Generating model | $M$ | AE | ME | AE | ME |
| 20 | MMNL(100.0, 5) | 10 | 3.40 | 16.38 | 3.67 | 15.30 |
| 20 | MMNL(100.0, 5) | 20 | 1.94 | 10.38 | 3.54 | 14.73 |
| 20 | MMNL(100.0, 5) | 50 | 0.78 | 6.00 | 3.40 | 14.45 |
| 20 | MMNL(100.0, 5) | 100 | 0.41 | 4.50 | 3.37 | 14.24 |
| 20 | MMNL(100.0, 10) | 10 | 3.35 | 13.61 | 3.03 | 12.15 |
| 20 | MMNL(100.0, 10) | 20 | 2.07 | 9.19 | 2.88 | 11.49 |
| 20 | MMNL(100.0, 10) | 50 | 0.90 | 5.03 | 2.81 | 11.11 |
| 20 | MMNL(100.0, 10) | 100 | 0.47 | 3.62 | 2.78 | 10.96 |

accurate than those produced by CG. It is also interesting to compare MNL to CG as the amount of data (the number of training assortments $M$) varies. Table 7 presents the same accuracy metrics as Table 6 as $M$ varies in $\{10, 20, 50, 100\}$ for $L = 100.0$ and $T \in \{5, 10\}$. From this table, we can see that while the prediction error of CG decreases significantly as $M$ increases, the error of the fitted MNL model decreases only slightly. In words, the MNL approach is not able to learn the true model with additional data, because it is constrained to a single parametric form, while our approach is able to learn from this additional data.

With regard to the worst-case approach, we see from Table 6 that over all of the types of instances – values of $L$ and $T$ – the column generation approach yields significantly more accurate predictions of revenue than the worst-case approach of Farias et al. (2013). For example, for $L = 5.0$ and $T = 10$, the average revenue error, averaged over 100 instances, is 1.77 for the column generation approach, while for the worst-case approach it is 11.83 – approximately an order of magnitude higher. Moreover, since each revenue prediction in the worst-case approach involves solving a large-scale LO problem, the average time to make predictions is considerably higher; the largest average time is on the order of 2.6 seconds per prediction, whereas for both CG and the force-fitted MNL model it is on the order of milliseconds.

Why does the approach of Farias et al. (2013) give less accurate predictions? There are two reasons for this. The first reason is that the worst-case approach predicts the lowest possible revenue; it finds the probability distribution that is consistent with the data and that results in the lowest possible revenue. Depending on the nature of the choice model and the data, the set of probability distributions that are consistent with the data may be quite large and the revenue predictions may thus be quite conservative. The second reason for the inaccuracy is that the worst-case approach predicts the lowest possible revenue *for each assortment*. Although each such prediction arises from a probability distribution that is consistent with the data, the revenue-minimizing probability distribution *may not be the same for each assortment*; thus, the ensemble of revenue predictions produced by the worst-case approach may not be realized by a *single* choice model. Due to this property, we therefore expect that the worst-case approach may exhibit some error in the aggregate that cannot be avoided.

Lastly, we can see that averaging leads to more accurate predictions. In particular, from Table 6, we can see that AvgCG in all cases leads to lower average and maximum errors than CG. This improvement can be understood from a statistical learning perspective by considering bias and variance; namely, averaging reduces the variance in the mean square error made by CG that is induced by the randomness of the estimation procedure, without changing the bias.

## 5.5. Combining estimation and optimization

We now consider combining our estimation approach with our optimization approach. To evaluate this combined approach for a given instance, we first estimate the rankings $\sigma^1, \dots, \sigma^K$ and probabilities $\lambda^1, \dots, \lambda^K$ from the data for that instance using our column generation procedure, and we then use these estimated rankings and probabilities to formulate and solve problem (1), yielding an assortment. We then evaluate the *true* revenue of the assortment under the model that generated the data of that instance and compare it to the optimal revenue for that generating model. Using $R_{\text{true}}(S)$ to denote the true revenue of the assortment $S$ and $R^*_{\text{true}}$ to denote the optimal true revenue, we define the optimality gap $G$ as

$$G = 100\% \times \frac{R^*_{\text{true}} - R_{\text{true}}(S)}{R^*_{\text{true}}}. \tag{6}$$

The smaller $G$ is, the better our approach performs; a value of 0% indicates that our approach captures all of the revenue that is possible under the model that generated the data.

Table 8 presents results for our approach for the same instances considered in Section 5.3. (As in our other comparisons, we report the averages of each metric over the 100 instances for each value of $n$ and generating model. We also re-use the same rankings and probabilities that were estimated in Section 5.3 for Table 3.) From this table, we can see that our approach results in an optimality gap on the order of a few percent, using only $M = 20$ training assortments. We can also see that the total time for the combined approach – estimating the rankings and probabilities from the transaction data and then solving the MIO – is also quite small; in the most extreme case, it is no more than 50 seconds on average.

How does the optimality gap change as the data increases? Table 9 shows the average optimality gap for the instance set from Section 5.3, restricted to $n = 30$ and $L = 5.0$, as $M$ varies in $\{10, 20, 50, 100\}$. We can see that as the amount of data increases, the average optimality gap decreases. At $M = 10$ training assortments, it is on average 3-4%, but decreases to below 1.5% with $M = 100$ assortments. This table complements Table 4 in Section 5.3, which showed that

**Table 8** Results of combining the estimation and optimization procedures over a wide range of MMNL models.

| $n$ | Generating model | $M$ | Gap (%) | Est. Time (s) | Opt. Time (s) | Total Time (s) |
|---|---|---|---|---|---|---|
| 10 | MMNL(5.0,1) | 20 | 2.98 | 0.68 | 0.02 | 0.70 |
| 10 | MMNL(5.0,5) | 20 | 1.98 | 0.57 | 0.02 | 0.59 |
| 10 | MMNL(5.0,10) | 20 | 1.62 | 0.53 | 0.02 | 0.55 |
| 10 | MMNL(10.0,1) | 20 | 1.87 | 0.71 | 0.02 | 0.73 |
| 10 | MMNL(10.0,5) | 20 | 1.43 | 0.59 | 0.02 | 0.61 |
| 10 | MMNL(10.0,10) | 20 | 2.10 | 0.50 | 0.02 | 0.53 |
| 10 | MMNL(100.0,1) | 20 | 2.86 | 0.79 | 0.02 | 0.81 |
| 10 | MMNL(100.0,5) | 20 | 1.38 | 0.67 | 0.02 | 0.69 |
| 10 | MMNL(100.0,10) | 20 | 2.09 | 0.56 | 0.02 | 0.58 |
| 20 | MMNL(5.0,1) | 20 | 4.24 | 7.41 | 0.16 | 7.57 |
| 20 | MMNL(5.0,5) | 20 | 2.39 | 4.09 | 0.13 | 4.22 |
| 20 | MMNL(5.0,10) | 20 | 1.98 | 3.30 | 0.12 | 3.42 |
| 20 | MMNL(10.0,1) | 20 | 2.92 | 9.42 | 0.12 | 9.54 |
| 20 | MMNL(10.0,5) | 20 | 1.93 | 4.96 | 0.12 | 5.08 |
| 20 | MMNL(10.0,10) | 20 | 2.47 | 3.71 | 0.16 | 3.87 |
| 20 | MMNL(100.0,1) | 20 | 3.57 | 11.13 | 0.09 | 11.22 |
| 20 | MMNL(100.0,5) | 20 | 3.01 | 7.53 | 0.14 | 7.67 |
| 20 | MMNL(100.0,10) | 20 | 1.80 | 5.00 | 0.13 | 5.13 |
| 30 | MMNL(5.0,1) | 20 | 2.94 | 23.49 | 0.39 | 23.88 |
| 30 | MMNL(5.0,5) | 20 | 2.10 | 13.83 | 0.39 | 14.22 |
| 30 | MMNL(5.0,10) | 20 | 2.30 | 10.76 | 0.47 | 11.22 |
| 30 | MMNL(10.0,1) | 20 | 3.51 | 32.52 | 0.43 | 32.95 |
| 30 | MMNL(10.0,5) | 20 | 2.45 | 16.49 | 0.50 | 16.98 |
| 30 | MMNL(10.0,10) | 20 | 2.00 | 13.20 | 0.49 | 13.69 |
| 30 | MMNL(100.0,1) | 20 | 4.35 | 47.79 | 0.29 | 48.09 |
| 30 | MMNL(100.0,5) | 20 | 2.20 | 44.20 | 0.71 | 44.90 |
| 30 | MMNL(100.0,10) | 20 | 2.43 | 25.20 | 0.65 | 25.86 |

**Table 9** Results of combining the estimation and optimization procedures as the amount of available data (the number of training assortments $M$) varies.

| $n$ | Generating model | $M$ | Gap (%) | Est. Time (s) | Opt. Time (s) | Total Time (s) |
|---|---|---|---|---|---|---|
| 30 | MMNL(5.0,1) | 10 | 3.85 | 4.65 | 0.19 | 4.84 |
| 30 | MMNL(5.0,1) | 20 | 2.94 | 23.49 | 0.39 | 23.88 |
| 30 | MMNL(5.0,1) | 50 | 2.05 | 103.12 | 0.76 | 103.87 |
| 30 | MMNL(5.0,1) | 100 | 1.47 | 302.39 | 1.08 | 303.47 |
| 30 | MMNL(5.0,5) | 10 | 3.45 | 3.42 | 0.19 | 3.61 |
| 30 | MMNL(5.0,5) | 20 | 2.10 | 13.83 | 0.39 | 14.22 |
| 30 | MMNL(5.0,5) | 50 | 1.38 | 72.23 | 0.88 | 73.11 |
| 30 | MMNL(5.0,5) | 100 | 0.98 | 231.02 | 1.62 | 232.64 |
| 30 | MMNL(5.0,10) | 10 | 3.50 | 3.09 | 0.26 | 3.35 |
| 30 | MMNL(5.0,10) | 20 | 2.30 | 10.76 | 0.47 | 11.22 |
| 30 | MMNL(5.0,10) | 50 | 1.36 | 68.88 | 1.16 | 70.03 |
| 30 | MMNL(5.0,10) | 100 | 0.78 | 225.90 | 2.22 | 228.12 |

**Table 10** **Results of combining the estimation and optimization procedures as training MAE tolerance varies.**

| Train MAE Tol. | Gap (%) | Est. Time (s) | Opt. Time (s) | Total Time (s) |
|---|---|---|---|---|
| 0.1000 | 20.95 | 0.06 | 0.00 | 0.06 |
| 0.0500 | 10.87 | 0.29 | 0.01 | 0.30 |
| 0.0100 | 4.58 | 1.94 | 0.06 | 2.01 |
| 0.0050 | 2.94 | 3.70 | 0.14 | 3.84 |
| 0.0010 | 2.06 | 11.04 | 0.49 | 11.53 |
| 0.0005 | 2.11 | 15.42 | 0.59 | 16.00 |
| 0.0001 | 1.87 | 24.74 | 0.88 | 25.62 |
| $5.00 \times 10^{-5}$ | 1.86 | 27.36 | 0.93 | 28.29 |
| $1.00 \times 10^{-5}$ | 1.87 | 30.27 | 0.93 | 31.19 |
| $5.00 \times 10^{-6}$ | 1.90 | 30.72 | 1.01 | 31.74 |
| $1.00 \times 10^{-6}$ | 1.87 | 31.16 | 0.98 | 32.14 |

Results correspond to MMNL(5.0,10) instances with $n = 30$ products and $M = 20$ training assortments.

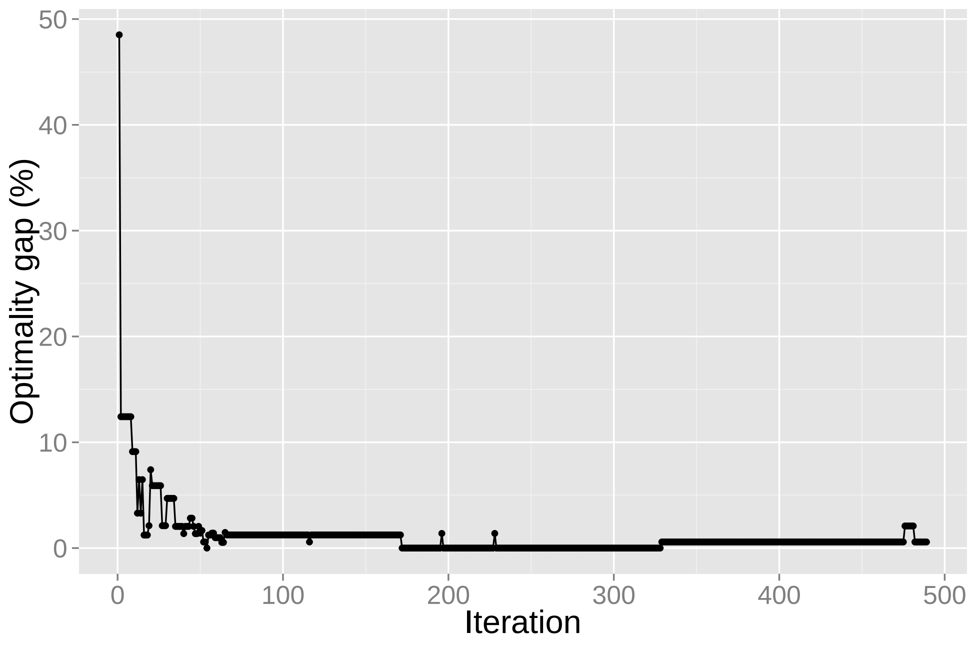
increasing data led to more accurate out-of-sample predictions; here, we have shown that more data translates to making more effective decisions.

Finally, we consider the question of whether it is possible to overfit the model from the perspective of optimization: namely, if we fit the training data too much, will we make worse decisions? Table 10 shows how the average optimality gap varies as the training MAE tolerance is decreased for those instances corresponding for $n = 30$, $T = 10$ and $L = 5.0$ from Section 5.3. We can see that as the tolerance decreases, the optimality gap on average in general decreases. Although the relationship is not monotonic, the optimality gap does not dramatically worsen when the column generation procedure is used to fit the training data to a very high level of precision. To provide a better visualization of this relationship, we show in Figure 2 how the optimality gap varies with the number of column generation iterations for a single instance (the same one studied in Figure 1).

## 5.6. Comparison of combined estimation and optimization procedure

Finally, we compare our combined estimation-optimization approach with other approaches. For each instance that we consider, we run the corresponding optimization procedure for each choice model that we considered in Section 5.4: for our estimated finite permutation model, we solve the MIO problem (1); for the fitted MNL model, we find the revenue-ordered subset with the highest predicted revenue (see Talluri and van Ryzin 2004); and for the worst-case approach, we apply the ADXOpt heuristic of Jagabathula (2014) with at most one removal for each product. As in Section 5.4, we consider optimizing the the ranking-based model produced by a single pass of the column generation procedure as well as optimizing the averaged ranking-based model that results from ten runs of the column generation procedure.

**Figure 2** **Evolution of optimality gap with each column generation iteration for one MMNL instance with $n = 30$, $T = 10$, $L = 5.0$ and $M = 20$ training assortments.**



For the final assortment produced by each approach, we compute the gap of this assortment relative to the *true* optimal revenue for the underlying ground truth model. We record the total time required for each approach: for our approach, we record the total of the time required for estimation (using the procedure in Section 4) and optimization using the MIO formulation (1); for the MNL approach, we record the total of the time required for maximum likelihood estimation and optimization via enumeration of the revenue-ordered subsets; and for the worst-case approach, we record the time required for optimization using ADXOpt, which includes the time expended each time that the worst-case revenue is computed. Each objective function evaluation for the worst-case/ADXOpt approach involves solving the worst-case model from Farias et al. (2013), which is a large-scale LO problem. We solve it by solving the sampled problem using the same sampled permutations that were used to make the revenue predictions in Section 5.4.

Table 11 reports the results of this comparison, which are averaged over the 100 instances corresponding to each value of $n$ and each generating model. In the table and in the discussion that follows, we use "CG+MIO", "AvgCG+MIO", "MNL" and "WC+ADXOpt" to indicate the single pass column generation and MIO combination, the averaged column generation and MIO combination, the MNL approach and the worst-case and ADXOpt combination, respectively.

With regard to WC+ADXOpt, CG+MIO is generally better and in a number of cases significantly so. For example for $n = 20$, $L = 5.0$, $T = 10$, the MIO approach achieves revenues that are on average 2.0% from the true optimal value, while the worst-case/ADXOpt approach achieves

**Table 11 Results of comparison of combined estimation-optimization approaches for $n = 20$, MMNL($\cdot,\cdot$) instances.**

| | | | CG+MIO | | AvgCG+MIO | | MNL | | WC+ADXOpt | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Generating model | $M$ | Gap (%) | Time (s) | Gap (%) | Time (s) | Gap (%) | Time (s) | Gap (%) | Time (s) |
| 20 | MMNL(5.0,1) | 20 | 4.24 | 7.57 | 2.18 | 73.49 | 0.00 | 0.69 | 4.46 | 314.79 |
| 20 | MMNL(5.0,5) | 20 | 2.39 | 4.22 | 0.96 | 48.58 | 3.88 | 0.73 | 5.26 | 177.34 |
| 20 | MMNL(5.0,10) | 20 | 1.98 | 3.42 | 0.92 | 37.10 | 3.25 | 0.71 | 13.65 | 250.50 |
| 20 | MMNL(10.0,1) | 20 | 2.92 | 9.54 | 0.97 | 107.00 | 0.00 | 0.70 | 6.30 | 463.60 |
| 20 | MMNL(10.0,5) | 20 | 1.93 | 5.08 | 1.20 | 54.21 | 4.27 | 0.72 | 4.17 | 193.51 |
| 20 | MMNL(10.0,10) | 20 | 2.47 | 3.87 | 1.01 | 43.71 | 3.55 | 0.74 | 6.17 | 171.88 |
| 20 | MMNL(100.0,1) | 20 | 3.57 | 11.22 | 0.96 | 118.24 | 0.00 | 0.76 | 3.31 | 1432.92 |
| 20 | MMNL(100.0,5) | 20 | 3.01 | 7.67 | 1.43 | 82.41 | 7.89 | 0.79 | 3.72 | 573.27 |
| 20 | MMNL(100.0,10) | 20 | 1.80 | 5.13 | 1.04 | 55.29 | 5.03 | 0.80 | 4.10 | 212.08 |

revenues that are on average 13.7% below the true optimal value. In one case ($L = 100.0$, $T = 1$) the worst-case/ADXOpt approach yields a lower gap, although the difference is quite small (3.31% compared to 3.57% for our approach). Furthermore, with regard to the running time, our estimation and MIO procedure together require strikingly less time than the worst-case approach of Farias et al. (2013) and the ADXOpt local search heuristic of Jagabathula (2014); in our approach, the average time required to both estimate the ranking-based model and optimize it is no more than 12 seconds in the most extreme case, whereas for the worst-case/ADXOpt approach, the average time is usually on the order of 200 seconds, and in one collection of instances ($L = 10.0$, $T = 5$) more than 20 minutes on average.

With regard to the MNL approach, we see that for the cases where $T = 1$ – the ground truth is a single-class MNL model – fitting an MNL model to the data and optimizing it results in an optimality gap of 0%, as we would expect. However, for $T > 1$, we can see that fitting and optimizing MNL can be quite suboptimal, and performs worse than our approach; for example with $L = 100.0$, $T = 5$, the average optimality gap of our approach is 3.01% compared to 7.89% for the MNL approach. To further compare the MNL approach to ours, we can also consider how the optimality gap changes as the number of training assortments $M$ is varied. Table 12 shows how the optimality gap of the MNL approach and our approach vary as $M$ varies for a subset of instances. Analogously to Table 7, which showed that the MNL approach does not become more accurate with more data, this table shows that the MNL approach does not result in more optimal decisions with more data. In contrast, our approach leads to improved decisions with more data.

Lastly, we observe that optimizing the averaged model (AvgCG+MIO) leads to even better performance than optimizing a model from a single column generation run (CG+MIO). For example, with $L = 10.0$ and $T = 10$, averaging reduces the gap from 2.47% to 1.01%. This improvement is particularly noteworthy given the relatively low amount of data that was available for building the

**Table 12** **Results of optimality gap comparison between MNL and CG+MIO approaches as number of training assortments $M$ varies, for $n = 20$, MMNL instances with $L = 100.0$ and $T \in \{5, 10\}$.**

| | | | CG+MIO | | MNL | |
|---|---|---|---|---|---|---|
| $n$ | Generating model | $M$ | Gap (%) | Time (s) | Gap (%) | Time (s) |
| 20 | MMNL(100.0,5) | 10 | 5.39 | 1.34 | 8.21 | 0.74 |
| 20 | MMNL(100.0,5) | 20 | 3.01 | 7.67 | 7.89 | 0.79 |
| 20 | MMNL(100.0,5) | 50 | 1.54 | 42.39 | 8.07 | 0.91 |
| 20 | MMNL(100.0,5) | 100 | 0.98 | 116.41 | 7.47 | 1.10 |
| 20 | MMNL(100.0,10) | 10 | 3.58 | 1.04 | 5.47 | 0.75 |
| 20 | MMNL(100.0,10) | 20 | 1.80 | 5.13 | 5.03 | 0.80 |
| 20 | MMNL(100.0,10) | 50 | 0.88 | 34.94 | 4.76 | 0.91 |
| 20 | MMNL(100.0,10) | 100 | 0.53 | 95.72 | 5.05 | 1.11 |

model. Comparing AvgCG+MIO to the other two methods, we can see that AvgCG+MIO delivers significantly better performance than WC+ADXOpt in all instances and MNL in all instances with $T > 1$. Based on these results, we believe that our approach has the potential to capture significant value in practical assortment decisions in the presence of limited data.

## 6. Conclusions

In this paper, we have presented a practical method for transforming limited historical transaction data into effective assortment decisions. Our method consists of two pieces: an estimation procedure for extracting a flexible, generic choice model from the data and an assortment optimization procedure for finding the best assortment given the estimated choice model. Modern mathematical optimization plays a key role in both pieces: the estimation piece is based on efficiently solving a large-scale LO problem using column generation, while the assortment optimization piece is based on solving a practically tractable MIO problem. We show that our methodology is scalable, flexible, leads to accurate revenue predictions and leads to near-optimal assortments that outperform alternative parametric and non-parametric approaches.

There are a number of promising directions for future research. In this paper we have assumed that the estimation occurs only once before the assortment decision is made, which is also made only once. However, in a practical setting, one may make multiple assortment decisions over time: each assortment decision will yield new data on the behavior of the market, allowing the firm to change the assortment over time in response to this data. Thus, framing the problem in a dynamic setting with learning is a valuable next step. In a different direction, one may consider how to extend the procedure when the data is richer and more fine-grained, and when the decision of the firm is at a similar resolution: for example, a firm may track transactions made by individual customers and attributes of those customers, and may be able to make assortment decisions that are targeted to individual customers. The challenge in this setting is to estimate a model that predicts the choice probability of each item in an assortment *given a particular customer's attributes*, and to then use this model to optimally target customers.

## Acknowledgments

The work of the second author was supported by a PGS-D award from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

## References

A. Belloni, R. Freund, M. Selove, and D. Simester. Optimizing product line designs: Efficient methods and comparisons. *Management Science*, 54(9):1544–1552, 2008.

M. E. Ben-Akiva and S. R. Lerman. *Discrete choice analysis: theory and application to travel demand*, volume 9. MIT press, 1985.

D. Bertsimas and V. V. Mišić. Robust product line design. *Under review*, 2014.

D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific, Belmont, MA, 1997.

J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

J. Blanchet, G. Gallego, and V. Goyal. A Markov chain approximation to choice modeling. *Submitted*, 2013. Available at `http://www.columbia.edu/~vg2277/MC_paper.pdf`.

J. J. M. Bront, I. Méndez-Díaz, and G. Vulcano. A column generation algorithm for choice-based network revenue management. *Operations Research*, 57(3):769–784, 2009.

Celect, Inc., 2014. Accessed February 11, 2015; available at `http://www.celect.net`.

J. Davis, G. Gallego, and H. Topaloglu. Assortment planning under the multinomial logit model with totally unimodular constraint structures. Technical report, Department of IEOR, Columbia University. Available at `http://www.columbia.edu/~gmg2/logit_const.pdf`, 2013.

J. M. Davis, G. Gallego, and H. Topaloglu. Assortment optimization under variants of the nested logit model. *Operations Research*, 62(2):250–273, 2014.

A. Désir and V. Goyal. Near-optimal algorithms for capacity constrained assortment optimization. *Available at SSRN 2543309*, 2014.

V. F. Farias, S. Jagabathula, and D. Shah. A nonparametric approach to modeling choice with limited data. *Management Science*, 59(2):305–322, 2013.

J. B. Feldman and H. Topaloglu. Revenue Management Under the Markov Chain Choice Model. *Working paper*, 2014. Available at `http://people.orie.cornell.edu/huseyin/publications/mc_revenue.pdf`.

J. B. Feldman and H. Topaloglu. Capacity constraints across nests in assortment optimization under the nested logit model. *Operations Research*, forthcoming, 2015. Available at `http://legacy.orie.cornell.edu/huseyin/publications/nested_capacitated_full.pdf`.

M. Fisher and R. Vaidyanathan. Which Products Should You Stock? A new approach to assortment planning turns an art into a science. *Harvard Business Review*, page 108, 2012.

G. Gallego and H. Topaloglu. Constrained assortment optimization for the nested logit model. *Management Science*, 60(10):2583–2601, 2014.

Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual, 2015. URL `{http://www.gurobi.com}`.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning.* Springer, 2009.

IBM. IBM – DemandTec Assortment Optimization, 2015. Accessed February 11, 2015; available at `http://www-03.ibm.com/software/products/en/assortment-optimization`.

S. Jagabathula. Assortment optimization under general choice. *Available at SSRN*, 2014.

JDA Software Group, Inc. JDA Assortment Optimization — JDA Software, 2015. Accessed February 11, 2015; available at `http://www.jda.com/solutions/assortment-optimization/`.

A. G. Kök, M. L. Fisher, and R. Vaidyanathan. Assortment planning: Review of literature and industry practice. In Narendra Agrawal and Stephen A. Smith, editors, *Retail Supply Chain Management*, volume 122 of *International Series in Operations Research & Management Science*, pages 99–153. Springer US, 2009. ISBN 978-0-387-78902-6. doi: 10.1007/978-0-387-78902-6_6. URL `http://dx.doi.org/10.1007/978-0-387-78902-6_6`.

G. Li, P. Rusmevichientong, and H. Topaloglu. The d-level nested logit model: Assortment and price optimization problems. Technical report, Cornell University, School of Operations Research and Information Engineering. Available at `http://legacy.orie.cornell.edu/~huseyin/publications/publications.html`, 2013.

M. Lubin and I. Dunning. Computing in Operations Research Using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

Oracle Corporation. Oracle Retail – World Class Commerce Solutions — Oracle, 2015. Accessed June 1, 2015; available at `https://www.oracle.com/industries/retail/index.html`.

P. Rusmevichientong and H. Topaloglu. Robust assortment optimization in revenue management under the multinomial logit choice model. *Operations Research*, 60(4):865–882, 2012.

P. Rusmevichientong, Z.-J. M. Shen, and D. B. Shmoys. Dynamic assortment optimization with a multinomial logit choice model and capacity constraint. *Operations Research*, 58(6):1666–1680, 2010.

P. Rusmevichientong, D. Shmoys, C. Tong, and H. Topaloglu. Assortment optimization under the multinomial logit model with random choice parameters. *Production and Operations Management*, 23(11): 2023–2039, 2014.

K. Talluri and G. van Ryzin. Revenue management under a general discrete choice model of consumer behavior. *Management Science*, 50(1):15–33, 2004.

R. H. Thaler and C. R. Sunstein. *Nudge*. Yale University Press, 2008.

K. E. Train. *Discrete choice methods with simulation*. Cambridge university press, 2009.

G. van Ryzin and G. Vulcano. A market discovery algorithm to estimate a general class of nonparametric choice models. *Management Science*, 61(2):281–300, 2015.

A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.