Assortment Optimization under the Decision Forest Model

Yi-Chun Akchen
School of Management, University College London, London E14 5AB, United Kingdom.
yi-chun.akchen@ucl.ac.uk

Velibor V. Mišić
UCLA Anderson School of Management, University of California, Los Angeles, California 90095, United States,
velibor.misic@anderson.ucla.edu

Problem definition: We study the problem of finding the optimal assortment that maximizes expected revenue under the decision forest model, a recently proposed nonparametric choice model that is capable of representing any discrete choice model and in particular, can be used to represent non-rational customer behavior. This problem is of practical importance because it allows a firm to tailor its product offerings to profitably exploit deviations from rational customer behavior, but at the same time is challenging due to the extremely general nature of the decision forest model. *Methodology/Results*: We approach this problem from a mixed-integer optimization perspective and present two different formulations. We theoretically compare the two formulations in strength, and analyze when they are integral in the special case of a single tree. We further propose a methodology for solving the two formulations at a large-scale based on Benders decomposition, and show that the Benders subproblem can be solved efficiently by primal-dual greedy algorithms when the master solution is fractional for one of the formulations, and in closed form when the master solution is binary for both formulations. Using synthetically generated instances, we demonstrate the practical tractability of our formulations and our Benders decomposition approach, and their edge over heuristic approaches. *Managerial implications:* In a case study based on a real-world transaction data, we demonstrate that our proposed approach can factor the behavioral anomalies observed in consumer choice into assortment decision and create higher revenue.

Key words: decision trees; choice modeling; integer optimization; Benders decomposition; choice overload

1. Introduction

Assortment optimization is a basic operational problem faced by many firms. In its simplest form, the problem can be posed as follows. A firm has a set of products that it can offer, and a set of customers who have preferences over those products; what is the set of products the firm should offer so as to maximize the revenue that results when the customers choose from these products?

While assortment optimization and the related problem of product line design have been studied extensively under a wide range of choice models, the majority of research in this area focuses on rational choice models, specifically those that follow the random utility maximization (RUM) assumption. A significant body of research in the behavioral sciences shows that customers behave in ways that deviate significantly from predictions made by RUM models. In addition, there is a substantial number of empirical examples of firms that make assortment decisions in ways that directly exploit customer irrationality. For example, the paper of Kivetz et al. (2004) provides an example of an assortment of document preparation systems from Xerox that are structured around

the decoy effect, and an example of an assortment of servers from IBM that are structured around the compromise effect.

A recent paper by Chen and Mišić (2022) proposed a new choice model called the *decision forest model* for capturing customer irrationalities. This model involves representing the customer population as a probability distribution over binary trees, with each tree representing the decision process of one customer type. In a key result of the paper, the authors showed that this model is universal: *every discrete choice model is representable as a decision forest model*. While the paper of Chen and Mišić (2022) alludes to the downstream assortment optimization problem, it is entirely focused on model representation and prediction: it does not provide any answer to how one can select an optimal assortment with respect to a decision forest model.

In the present paper, we present a methodology for assortment optimization under the decision forest model, based on mixed-integer optimization. Our approach allows the firm to obtain assortments that are either provably optimal or within a desired optimality gap for a given decision forest model. At the same time, the approach easily allows the firm to incorporate business rules as linear constraints in the MIO formulation. Most importantly, given the universality property of the decision forest model, our optimization approach allows a firm to optimally tailor its assortment to any kind of predictable irrationality in the firm's customer population.

We make the following specific contributions:

1. We formally define the assortment optimization problem under the decision forest model and analyze its computational hardness. Specifically, assuming the tree depth is bounded by a constant k, we prove that it is NP-hard to approximate the problem within a factor that grows exponentially with k. Given this computational challenge, we then adopt an integer programming approach and present two formulations: SplitMIO and ProductMIO. We analyze the constraint matrices of the two formulations and show that in the special case of a single purchase decision tree, SplitMIO is integral when each product appears at most once in the splits of the tree, and ProductMIO is always integral regardless of the tree structure.

2. We propose a Benders decomposition approach for solving the two formulations at a large scale. We show that Benders cuts for the linear optimization relaxations of SplitMIO can be obtained via a greedy algorithm that solves both the primal and dual of the subproblem. We also provide a simple example to show that the same type of greedy algorithm fails to solve the primal subproblem of ProductMIO. We further show how to obtain Benders cuts for the integer solutions of both SplitMIO and ProductMIO in closed form.

3. We present numerical experiments using both synthetic and real-world data. We first use the synthetic data to examine the formulation strength of SplitMIO and ProductMIO models, and to demonstrate the scalability of the Benders decomposition approach for the SplitMIO

formulation in problem instances involving up to 3000 products, 500 trees, and 512 leaves per tree. We then use a real-world dataset to demonstrate how the decision forest model can factor a behavioral anomaly (*choice overload*) into its assortment decision and create higher revenue.

We organize the paper as follows. Section 2 reviews the related literature. Section 3 defines the assortment problem, characterizes its inapproximability, and presents the two MIO formulations. Section 4 proposes a Benders decomposition approach for these formulations. Sections 5 and 6 present the numerical results involving both the synthetic and real-world data. All proofs are relegated to the appendix.

2. Literature review

Assortment optimization has been extensively studied in the operations management community; we refer readers to Gallego and Topaloglu (2019) for a recent review of the literature. The literature on assortment optimization has focused on developing approaches for finding the optimal assortment under many different rational choice models, such as the multinomial logit model (MNL) model (Talluri and Van Ryzin 2004, Sumida et al. 2020), the latent class MNL model (Bront et al. 2009, Méndez-Díaz et al. 2014), the Markov chain choice model (Feldman and Topaloglu 2017, Désir et al. 2020) and the ranking-based model (Aouad et al. 2020, 2018, Feldman et al. 2019).

In addition to the assortment optimization literature, our paper is also related to the literature on product line design found in the marketing community. While assortment optimization is more often focused on the tactical decision of selecting which existing products to offer, where the products are ones that have been sold in the past and the choice model comes from transaction data involving those products, the product line design problem involves selecting which new products to offer, where the products are candidate products (i.e., they have not been offered before) and the choice model comes from conjoint survey data, where customers are asked to rate or choose between hypothetical products. Research in this area has considered different approaches to solve the problem under the ranking-based/first-choice model (McBride and Zufryden 1988, Belloni et al. 2008, Bertsimas and Mišić 2019) and the MNL model (Chen and Hausman 2000, Schön 2010).

Our paper is related to Bertsimas and Mišić (2019), which presents integer optimization formulations of the product line design problem when the choice model is a ranking-based model. As we will see later, our formulation SplitMIO can be viewed a generalization of the formulation Bertsimas and Mišić (2019), to the decision forest model. In addition, the paper of Bertsimas and Mišić (2019) develops a specialized Benders decomposition approach for its formulation, which uses the fact that one can solve the subproblem associated with each customer type by applying a greedy algorithm. We will show in Section 4 that this same property generalizes to the SplitMIO formulation, leading to a tailored Benders decomposition algorithm for solving SplitMIO at scale.

Beyond these specific connections, the majority of the literature on assortment optimization and product line design considers rational choice models, whereas our paper contributes a methodology for non-rational assortment optimization. Fewer papers have focused on choice modeling for non-rational customer behavior; besides the decision forest model, other models include the generalized attraction model (GAM; Gallego et al. 2015), the generalized stochastic preference model (Berbeglia 2018) and the generalized Luce model (Echenique and Saito 2019). An even smaller set of papers has considered assortment optimization under non-rational choice models, which we now review. The paper of Flores et al. (2017) considers assortment optimization under the two-stage Luce model, and develops a polynomial time algorithm for solving the unconstrained assortment optimization problem. The paper of Rooderkerk et al. (2011) considers a context-dependent utility model where the utility of a product can depend on other products that are offered and that can capture compromise, attraction and similarity effects; the paper empirically demonstrates how incorporating context effects leads to a predicted increase of 5.4% in expected profit.

Relative to these papers, our paper differs in that it considers the decision forest model. As noted earlier, the decision forest model can represent any type of choice behavior, and as such, an assortment optimization methodology based on such a model is attractive in terms of allowing a firm to take the next step from a high-fidelity model to a decision. In addition, our methodology is built on mixed-integer optimization. This is advantageous because it allows a firm to leverage continuing improvements in integer optimization solvers such as Gurobi (Gurobi Optimization, LLC 2024) and CPLEX (IBM 2024), as well as continuing improvements in computer hardware. At the same time, integer optimization allows firms to accommodate business requirements using linear constraints, enhancing the practical applicability of the approach. Lastly, integer optimization also allows one to take advantage of well-studied large-scale solution methods. One such method that we focus on in this paper is Benders decomposition, which has seen an impressive resurgence in recent years for delivering state-of-the-art performance on large-scale problems such as hub location (Contreras et al. 2011), facility location (Fischetti et al. 2017) and set covering (Cordeau et al. 2019); see also Rahmaniani et al. (2017) for a review of the recent literature.

In addition to the assortment optimization and product line design literatures, our formulations also have connections with others that have been proposed in the broader optimization literature. The formulation SplitMIO that we will present later can be viewed as a special case of the mixed-integer optimization formulation of Mišić (2020) for optimizing the predicted value of a tree ensemble model, such as a random forest or a boosted tree model. The other formulation, ProductMIO, also has a connection to the integer optimization literature on formulating disjunctive constraints through independent branching schemes (Vielma et al. 2010, Vielma and Nemhauser 2011, Huchette and Vielma 2019); we also discuss this connection in more detail in Section 3.4.

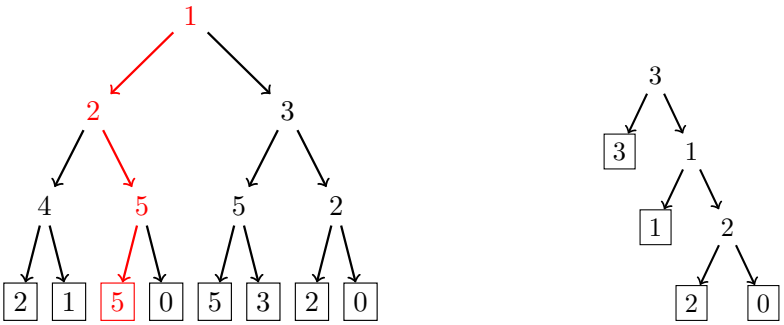


Figure 1 **(left) Example of a purchase decision tree for $n = 5$ products. Leaf nodes are enclosed in squares, while split nodes are not enclosed. The number on each node corresponds either to $v(t, s)$ for splits, or $c(t, \ell)$ for leaves. The path highlighted in red indicates how a customer following this tree to maps the assortment $S = \{1, 3, 4, 5\}$ to a leaf. For this assortment, the customer's decision is to purchase product 5; (right) Example of a purchase decision tree that represents a preference ranking $\{3 \succ 1 \succ 2 \succ 0\}$.**

3. Optimization model

In this section, we define the decision forest assortment optimization problem (Section 3.1), analyze its computational complexity (Section 3.2), and subsequently present the two mixed-integer formulations, SplitMIO (Section 3.3) and ProductMIO (Section 3.4). Lastly, we show that when the number of trees is assumed to be constant, the decision forest assortment optimization problem can be solved in polynomial time (Section 3.5).

3.1. Problem definition

We first briefly review the decision forest model of Chen and Mišić (2022), and then formally state the assortment optimization problem. We assume that there are n products, indexed from 1 to n, and let $\mathcal{N} = \{1, \ldots, n\}$ denote the set of all products. An assortment S corresponds to a subset of $\mathcal{N}$. When offered S, a customer may choose to purchase one of the products in S, or to not purchase anything from S at all; we use the index 0 to denote the latter possibility, which we will also refer to as the no-purchase option or the outside option.

The basic building block of the decision forest model is a purchase decision tree. Each tree represents the purchasing behavior of one type of customer when presented with an assortment S. Specifically, for an assortment S, the customer behaves as follows: the customer starts at the root of the tree. The customer checks whether the product corresponding to the root node is contained in S; if it is, she proceeds to the left child, and if not, she proceeds to the right child. She then checks again with the product at the new node, and the process repeats, until the customer reaches a leaf; the option that is at the leaf represents the choice of that customer. Figure 1 (left) shows an example of a purchase decision tree being used to map an assortment to a purchase decision.

Formally, a purchase decision tree is a directed binary tree, with each leaf node corresponding to an option in $\mathcal{N} \cup \{0\}$, and each non-leaf (or *split*) node corresponding to a product in $\mathcal{N}$. We use

splits(t) to denote the set of split nodes of tree t, and **leaves**(t) to denote the set of leaf nodes. We use $c(t,\ell)$ to denote the purchase decision of leaf ℓ of tree t, i.e., the option chosen by tree t if the assortment is mapped to leaf ℓ. We use $v(t,s)$ to denote the product that is checked at split node s in tree t. We make the following assumption about our purchase decision trees.

ASSUMPTION 1. *Let t be a purchase decision tree. For any two split nodes s and s' of t such that s' is a descendant of s, $v(t,s) \neq v(t,s')$.*

This assumption states that once a product appears on a split s, it cannot appear on any subsequent split s' that is reached by proceeding to the left or right child of s; in other words, each product in $\mathcal{N}$ appears at most once along the path from the root node to a leaf node, for every leaf node. As discussed in Chen and Mišić (2022), this assumption is not restrictive, as any tree for which this assumption is violated has a set of splits and leaves that are redundant and unreachable, and the tree can be modified to obtain an equivalent tree that satisfies the assumption.

The decision forest model assumes that the customer population is represented by a collection of trees or a *forest* F. Each tree $t \in F$ corresponds to a different customer type. We use λ_t to denote the probability associated with customer type/tree t, and $\boldsymbol{\lambda} = (\lambda_t)_{t\in F}$ to denote the probability distribution over the forest F. For each tree t, we use $\eta(t,S)$ to denote the choice that a customer type following tree t will make when given the assortment S. For a given assortment $S \subseteq \mathcal{N}$ and a given choice $j \in S \cup \{0\}$, we use $\mathbf{P}^{(F,\boldsymbol{\lambda})}(j \mid S)$ to denote the choice probability, i.e., the probability of a random customer choosing j when offered the assortment S. It is defined as

$$\mathbf{P}^{(F,\boldsymbol{\lambda})}(j \mid S) = \sum_{t\in F} \lambda_t \cdot \mathbb{I}\{\eta(t,S) = j\}. \tag{1}$$

We now define the assortment optimization problem. We use ρ_i to denote the marginal revenue of product i; for convenience, we use $\rho_0 = 0$ to denote the revenue of the no-purchase option. The assortment optimization problem that we wish to solve is

$$\underset{S\subseteq\mathcal{N}}{\text{maximize}} \sum_{i\in S} \rho_i \cdot \mathbf{P}^{(F,\boldsymbol{\lambda})}(i \mid S). \tag{2}$$

This is a challenging problem due to the generality of the choice model $\mathbf{P}^{(F,\boldsymbol{\lambda})}(\cdot \mid \cdot)$. Notably, it subsumes the assortment optimization problem under the ranking-based model, which is known to be NP-hard (Aouad et al. 2018, Feldman et al. 2019). This conclusion arises from the observation that, in the special case where each tree $t \in F$ branches exclusively to the right – meaning the left child of every split node in tree t is always a leaf node (as illustrated in the right panel of Figure 1) – the forest F simplifies into a collection of preference rankings. This observation directly leads to the following corollary.

COROLLARY 1. *The assortment optimization problem* (2) *is NP-Hard.*

3.2. Inapproximability Results

We strengthen the complexity results for the assortment optimization problem (2) by providing an inapproximability gap based on the depth of the decision trees. First, we define the depth of a tree t as the maximal distance from the root of tree t to any leaf node $\ell \in \mathbf{leaves}(t)$. For example, both trees in Figure 1 are of depth three. Second, we let AODF(k) denote problem (2) where each tree $t \in F$ can have depth at most k. With these definitions, we have the following result.

THEOREM 1. *If $P \neq NP$, for every constant $k \geq 7$ and $\epsilon > 0$, there is no polynomial-time $\left(2^{k-2\lceil\sqrt{k}\rceil}/(2ek) - \epsilon\right)$-approximate algorithm for AODF(k+1), even when each tree has equal probability weight in the model, i.e., $\lambda_t = 1/|F|$.*

We prove Theorem 1 by constructing an approximation-preserving reduction between AODF($k+1$) and the Max 1-in-EkSAT problem (Guruswami and Trevisan 2005), for which the same inapproximability result holds. The Max 1-in-EkSAT problem is a variant of the classic SAT problem (Garey and Johnson 1979), where each clause is satisfied if and only if exactly one literal in the clause is true, and each clause consists of exactly k literals. In our reduction, we construct a decision tree with depth $k+1$ and $O(k^2)$ leaf nodes for each clause in the Max 1-in-EkSAT problem.

Theorem 1 implies that the inapproximability gap of the assortment problem (2) grows *at least* exponentially with respect to the tree depth, since the ratio in Theorem 1 follows $2^{k-2\lceil\sqrt{k}\rceil}/(2ek) = 2^{k-2\lceil\sqrt{k}\rceil-\log k-1-\log_2 e} = 2^{k-o(k)}$. The following proposition further shows that this exponential dependence is asymptotically tight in k (i.e., when k is large), as there exists an approximation algorithm achieving approximation ratio 2^k.

PROPOSITION 1. *AODF(k) can be approximated within factor 2^k.*

Before we proceed to solve the challenging assortment optimization problem (2), we pause to make two remarks on Theorem 1. First, recall that any ranking of length k can be represented by a decision tree that is constrained to branch to the right and of depth k. Aouad et al. (2018) shows that the assortment optimization under the ranking-based model is NP-hard to be approximated within $O(k^{1-\epsilon})$ factor provided that each ranking in the model has a length of at most k (see Section 3.2 of Aouad et al. (2018)). Theorem 1 shows that the assortment problem becomes much harder when optimizing over decision trees instead of rankings. Specifically, the inapproximability factor grows from a linear rate $O(k)$ to an exponential rate $O(2^k)$ as a function of k.

Second, together with Chen and Mišić (2022), Theorem 1 highlights a fundamental tradeoff between the representation power and tractability in the decision forest model. In particular, Chen and Mišić (2022) shows that as the decision trees grow deeper, the model becomes more expressive in capturing choice behavior; at its maximum depth, n, the decision forest model can represent

any discrete choice model (see Theorems 1 and 2 in Chen and Mišić (2022)). At the same time, our Theorem 1 establishes that deeper trees also lead to increased computational complexity for assortment optimization, with inapproximability growing exponentially in tree depth. This tradeoff underscores the importance of managing tree depth to ensure that the model accurately reflects consumer choice while maintaining tractability for downstream operational tasks.

Given the inapproximability of problem (2), we consider a mixed-integer optimization (MIO) approach to solve the problem and present two formulations in the next two subsections.

3.3. Formulation 1: SplitMIO

We now present our first formulation of the assortment optimization problem (2) as a mixed-integer optimization (MIO) problem. To formulate the problem, we introduce some additional notation. For notational convenience we let $r_{t,\ell} = \rho_{c(t,\ell)}$ be the revenue of the purchase option of leaf ℓ of tree t. We let $\mathbf{left}(s)$ denote the set of leaf nodes that are to the left of split s (i.e., can only be reached by taking the left branch at split s), and similarly, we let $\mathbf{right}(s)$ denote the leaf nodes that are to the right of s. We introduce two sets of decision variables. For each $i \in \mathcal{N}$, we let x_i be a binary decision variable that is 1 if product i is included in the assortment, and 0 otherwise. For each tree $t \in F$ and leaf $\ell \in \mathbf{leaves}(t)$, we let $y_{t,\ell}$ be a binary decision variable that is 1 if the assortment encoded by $\mathbf{x}$ is mapped to leaf ℓ of tree t, and 0 otherwise.

With these definitions, our first formulation, SplitMIO, is given below.

$$
\begin{aligned}
\textsc{SplitMIO}: \quad \underset{\mathbf{x},\mathbf{y}}{\text{maximize}} \quad & \sum_{t \in F} \lambda_t \cdot \left[\sum_{\ell \in \mathbf{leaves}(t)} r_{t,\ell} y_{t,\ell} \right] && \text{(3a)} \\
\text{subject to} \quad & \sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} = 1, \quad \forall\ t \in F, && \text{(3b)} \\
& \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} \leq x_{v(t,s)}, \quad \forall\ t \in F,\ s \in \mathbf{splits}(t), && \text{(3c)} \\
& \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell} \leq 1 - x_{v(t,s)}, \quad \forall\ t \in F,\ s \in \mathbf{splits}(t), && \text{(3d)} \\
& x_i \in \{0,1\}, \quad \forall\ i \in \mathcal{N}, && \text{(3e)} \\
& y_{t,\ell} \geq 0, \quad \forall\ t \in F,\ \ell \in \mathbf{leaves}(t). && \text{(3f)}
\end{aligned}
$$

In order of appearance, the constraints in this formulation have the following meaning. Constraint (3b) requires that for each customer type t, the assortment encoded by $\mathbf{x}$ is mapped to exactly one leaf. Constraint (3c) imposes that for a split s in tree t, if product $v(t,s)$ is not in the assortment, then the assortment cannot be mapped to any of the leaves that are to the left of split s in tree t. Constraint (3d) is the symmetric case of constraint (3c) for the right-hand subtree of split s of tree t. The last two constraints require that $\mathbf{x}$ is binary and $\mathbf{y}$ is nonnegative. Note that

it is not necessary to require $\mathbf{y}$ to be binary, as the constraints ensure that each $y_{t,\ell}$ automatically takes the correct value whenever $\mathbf{x}$ is binary. Finally, the objective function corresponds to the expected per-customer revenue of the assortment.

The SplitMIO formulation is related to two existing MIO formulations in the literature. First, it can be viewed as a specialized case of the MIO formulation in Mišić (2020). In that paper, the author develops a formulation for tree ensemble optimization, i.e., the problem of setting the independent variables in a tree ensemble model (e.g., a random forest or a gradient boosted tree model) to maximize the value predicted by that ensemble. Later, in Section 4, we introduce a specialized algorithm that leverages a primal-dual greedy approach to efficiently solve the linear relaxation of SplitMIO. This method, which enhances the tractability of the SplitMIO formulation, represents our contribution to the literature and was not considered in Mišić (2020). Second, the SplitMIO formulation also relates to the MIO formulation for the product line design problem under the ranking-based model (Bertsimas and Mišić 2019). Recall that the ranking-based model can be regarded as a special case of the decision forest model (illustrated in the right panel of Figure 1). In this special case, the formulation (3) can be reduced to the MIO formulation for product line design under ranking-based models presented in Bertsimas and Mišić (2019).

Before continuing to our second formulation, we establish an important property of problem (3) when $|F| = 1$. When $|F| = 1$, we can show that $\mathcal{F}_{\text{SplitMIO}}$ is integral in a particular special case. (Note that in the statement of the proposition below, we drop the index t to simplify notation.)

Proposition 2. *Let $(F, \boldsymbol{\lambda})$ be a decision forest model consisting of a single tree, i.e., $|F| = 1$. Then $\mathcal{F}_{\text{SplitMIO}}$ is integral, i.e., every extreme point $(\mathbf{x}, \mathbf{y})$ of the polyhedron $\mathcal{F}_{\text{SplitMIO}}$ satisfies $\mathbf{x} \in \{0,1\}^N$, if and only if $v(s) = i$ for at most one $s \in \mathbf{splits}$ for every product $i \in \mathcal{N}$.*

The proof of this result (see Appendix A.3) follows by showing that the constraint matrix defining $\mathcal{F}_{\text{SplitMIO}}$ is totally unimodular. Proposition 2 is significant because it implies that for $|F| = 1$, the distinction between trees where each product appears at most once in any split and trees where a product may appear two or more times as a split is sharp. This insight provides the motivation for our second formulation, ProductMIO, which we present next.

3.4. Formulation 2: ProductMIO

The second formulation of problem (2) that we will present is motivated by the behavior of SplitMIO when a product participates in two or more splits. In particular, observe that a product i may participate in two different splits s_1 and s_2 in the same decision tree t. In this case, constraint (3c) in the SplitMIO will result in two constraints:

$$\sum_{\ell \in \mathbf{left}(s_1)} y_{t,\ell} \leq x_i, \tag{4}$$

$$\sum_{\ell \in \mathbf{left}(s_2)} y_{t,\ell} \leq x_i. \tag{5}$$

In the above two constraints, observe that $\mathbf{left}(s_1)$ and $\mathbf{left}(s_2)$ are disjoint (this is a direct consequence of Assumption 1). Given this and constraint (3b) that requires the $y_{t,\ell}$ variables to sum to 1, we can come up with a constraint that strengthens constraints (4) and (5) by combining them:

$$\sum_{\ell \in \mathbf{left}(s_1)} y_{t,\ell} + \sum_{\ell \in \mathbf{left}(s_2)} y_{t,\ell} \leq x_i. \tag{6}$$

In general, one can aggregate all the $y_{t,\ell}$ variables that are to the left of all splits involving a product i to produce a single left split constraint for product i. The same can also be done for the right split constraints. Generalizing this principle leads to the following alternate formulation, which we refer to as ProductMIO:

$$\begin{aligned}
\textsc{ProductMIO}: \quad \underset{\mathbf{x},\mathbf{y}}{\text{maximize}} \quad & \sum_{t \in F} \lambda_t \cdot \left[\sum_{\ell \in \mathbf{leaves}(t)} r_{t,\ell} y_{t,\ell} \right] && \text{(7a)} \\
\text{subject to} \quad & \sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} = 1, \quad \forall\ t \in F, && \text{(7b)} \\
& \sum_{\substack{s \in \mathbf{splits}(t): \\ v(t,s)=i}} \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} \leq x_i, \quad \forall\ t \in F,\ i \in \mathcal{N}, && \text{(7c)} \\
& \sum_{\substack{s \in \mathbf{splits}(t): \\ v(t,s)=i}} \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell} \leq 1 - x_i, \quad \forall\ t \in F,\ i \in \mathcal{N}, && \text{(7d)} \\
& x_i \in \{0,1\}, \quad \forall\ i \in \mathcal{N}, && \text{(7e)} \\
& y_{t,\ell} \geq 0, \quad \forall\ t \in F,\ \ell \in \mathbf{leaves}(t). && \text{(7f)}
\end{aligned}$$

ProductMIO differs from SplitMIO in several ways. First, note that while both formulations have the same number of variables, formulation ProductMIO has a smaller number of constraints. In particular, SplitMIO has one left and one right split constraints for each split in each tree, whereas ProductMIO has one left and one right split constraint for each product. When the trees involve a large number of splits, this can lead to a sizable reduction in the number of constraints. Note also that when a product does not appear in any splits of a tree, we can also safely omit constraints (7c) and (7d) for that product.

The second difference with formulation SplitMIO, as we have already mentioned, is in formulation strength. Let $\mathcal{F}_{\textsc{ProductMIO}}$ be the feasible region of the linear optimization (LO) relaxation of ProductMIO. The following proposition formalizes the fact that formulation ProductMIO is at least as strong as formulation SplitMIO.

Proposition 3. *For any decision forest model* $(F, \boldsymbol{\lambda})$, $\mathcal{F}_{\textsc{ProductMIO}} \subseteq \mathcal{F}_{\textsc{SplitMIO}}$.

The proof follows straightforwardly using the logic given above; we thus omit the proof.

The last major difference is in how ProductMIO behaves when $|F| = 1$. We saw that a sufficient condition for $\mathcal{F}_{\text{SplitMIO}}$ to be integral when $|F| = 1$ is that each product appears in at most one split in the tree. In contrast, formulation ProductMIO is *always* integral when $|F| = 1$.

Proposition 4. *For any decision forest model $(F, \boldsymbol{\lambda})$ with $|F| = 1$, $\mathcal{F}_{\text{ProductMIO}}$ is integral.*

In contrast to Proposition 2, we take a different approach to proving the integrality property, as the constraint matrix of the ProductMIO formulation is generally not totally unimodular (see the example in Section A.4). The proof of Proposition 4, given in Appendix A.5, leverages a connection between ProductMIO and another type of formulation in the literature. In particular, a stream of papers in the mixed-integer optimization community (Vielma et al. 2010, Vielma and Nemhauser 2011, Huchette and Vielma 2019) has considered a general approach for deriving small and strong formulations of disjunctive constraints using independent branching schemes; we briefly review the most general such approach from Huchette and Vielma (2019) and showcase its connection to ProductMIO. In this approach, one has a finite ground set J, and is interested in optimizing over a particular subset of the $(|J|-1)-$dimensional unit simplex over J, $\Delta^J = \{\boldsymbol{\lambda} \in \mathbb{R}^J \mid \sum_{j \in J} \lambda_j = 1; \boldsymbol{\lambda} \geq \mathbf{0}\}$. The specific subset that we are interested in is called a *combinatorial disjunctive constraint* (CDC), and is given by $\text{CDC}(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} Q(S)$, where $\mathcal{S}$ is a finite collection of subsets of J and $Q(S) = \{\boldsymbol{\lambda} \in \Delta \mid \lambda_j \leq 0 \text{ for } j \in J \setminus S\}$ for any $S \subseteq J$. This approach is very general: for example, by associating each j with a point $\mathbf{x}^j$ in $\mathbb{R}^n$, one can use $\text{CDC}(\mathcal{S})$ to model an optimization problem over a union of polyhedra, where each polyhedron is the convex hull of a collection of vertices in $S \in \mathcal{S}$.

A *k-way independent branching scheme of depth t* is a representation of $\text{CDC}(\mathcal{S})$ as a sequence of t choices between k alternatives. Specifically, we have $\text{CDC}(\mathcal{S}) = \bigcap_{m=1}^{t} \bigcup_{i=1}^{k} Q(L_m^i)$, where $L_m^i \subseteq J$. In the special case that $k = 2$, we can write $\text{CDC}(\mathcal{S}) = \cap_{m=1}^{t} (L_m \cup R_m)$ where $L_m, R_m \subseteq J$. This representation is known as a *pairwise independent branching scheme* and the constraints of the corresponding MIO can be written simply as

$$\sum_{j \in L_m} \lambda_j \leq z_m, \quad \forall\ m \in \{1, \ldots, t\}, \tag{8a}$$

$$\sum_{j \in R_m} \lambda_j \leq 1 - z_m, \quad \forall\ m \in \{1, \ldots, t\}, \tag{8b}$$

$$z_m \in \{0, 1\}, \quad \forall\ m \in \{1, \ldots, t\}, \tag{8c}$$

$$\sum_{j \in J} \lambda_j = 1, \tag{8d}$$

$$\lambda_j \geq 0, \quad \forall\ j \in J. \tag{8e}$$

This particular special case is important because it is always integral (see Theorem 1 of Vielma et al. 2010). Moreover, we can see that ProductMIO bears a strong resemblance to formulation (8). Constraints (8a) and (8a) correspond to constraints (7c) and (7d), respectively. In terms of variables, the λ_j and z_m variables in formulation (8) correspond to the $y_{t,\ell}$ and x_i variables in ProductMIO, respectively.

One notable difference is that in practice, one would use formulation (8) in a modular way; specifically, one would be faced with a problem where the feasible region can be written as $\text{CDC}(\mathcal{S}_1) \cap \text{CDC}(\mathcal{S}_2) \cap \cdots \cap \text{CDC}(\mathcal{S}_G)$, where each $\mathcal{S}_g$ is a collection of subsets of J. To model this feasible region, one would introduce a set of $z_{g,m}$ variables for the gth CDC, enforce constraints (8a) - (8c) for the gth CDC, and use only one set of λ_j variables for the whole formulation. Thus, the λ_j variables are the "global" variables, while the $z_{g,m}$ variables would be "local" and specific to each CDC. In contrast, in ProductMIO, the x_i variables (the analogues of z_m) are the "global" variables, while the $y_{t,\ell}$ variables (the analogues of λ_j) are the "local" variables.

3.5. Polynomial time algorithm when $|F| = O(1)$

We conclude Section 3 by discussing a special setting in which the assortment problem (2) is efficiently solvable. When $|F| = 1$, the assortment optimization problem is trivial to solve, because one can simply find which leaf corresponds to the highest value of $r_{t,\ell}$, and then follow the path from the leaf to the root backwards to identify which products must be present and absent in the assortment to ensure that it is mapped to this leaf. If the number of leaves is polynomial in n, then one can solve this problem in time polynomial in n.

In the case that $|F| = O(1)$, i.e., the number of trees is assumed to be constant, it turns out that the assortment problem can also be solved efficiently, which we formalize in the proposition below.

Proposition 5. *Assume $|F| = O(1)$ and $|\mathbf{leaves}(t)|$ is polynomial in n for each tree $t \in F$. Then the assortment problem* (2) *can be solved optimally in a polynomial time.*

Combined with Theorem 1, Proposition 5 shows that it is the number of trees $|F|$, rather than the structure of each tree (e.g., the number of leaves) that drives the complexity of the assortment problem (2) from being tractable to inapproximable (unless P = NP). Recall that in the reduction that proves Theorem 1, the number of leaf nodes in each tree is roughly $k^2/2$, which is $O(1)$ under the theorem's assumption of constant k. Consequently, compared to Theorem 1, Proposition 5 suggests that even if the trees become significantly more complex – where the number of leaves in each tree increases from $O(1)$ to a polynomial in n – the assortment optimization problem remains solvable in polynomial time, provided that the number of trees $|F|$ is $O(1)$.

4. Solution methodology based on Benders decomposition

While the formulations in Section 3 bring the assortment optimization problem under the decision forest choice model closer to being solvable in practice, the effectiveness of these formulations can be limited in large-scale problems. In particular, consider the case where there is a large number of trees in the decision forest model and each tree consists of a large number of splits and leaves. In this setting, both formulations – SplitMIO and ProductMIO– will have a large number of $y_{t,\ell}$ variables and a large number of constraints to link those variables with the x_i variables, and may require significant computation time.

At the same time, SplitMIO and ProductMIO share a common problem structure. In particular, they have two sets of variables: the $\mathbf{x}$ variables, which determine the products that are to be included, and the $(\mathbf{y}_t)_{t\in F}$ variables, which model the choice of each customer type. In addition, for any two trees t, t' such that $t \neq t'$, the $\mathbf{y}_t$ variables and $\mathbf{y}_{t'}$ variables do not appear together in any constraints. Thus, one can view each of our two formulations as *a two-stage stochastic program*, where each tree t corresponds to a scenario; the variable $\mathbf{x}$ corresponds to the first-stage decision; and the variable $\mathbf{y}_t$ corresponds to the second-stage decision under scenario t, which is appropriately constrained by the first-stage decision $\mathbf{x}$.

Thus, one can apply Benders decomposition to solve the problem. At a high level, Benders decomposition involves using linear optimization duality to represent the optimal value of the second-stage problem for each tree t as a piecewise-linear concave function of $\mathbf{x}$, and to eliminate the $(\mathbf{y}_t)_{t\in F}$ variables. One can then re-write the optimization problem in epigraph form, resulting in an optimization problem in terms of the $\mathbf{x}$ variable and an auxiliary epigraph variable θ_t for each tree t, and a large family of constraints linking $\mathbf{x}$ and θ_t for each tree t. Although the constraints for each tree t are too many to be enumerated, one can solve the problem through constraint generation.

The main message of this section is that the second-stage problem can be solved in a computationally efficient manner. In the remaining sections, we carefully analyze the second-stage problem for both formulations. For SplitMIO, we show that the second-stage problem can be solved by a greedy algorithm when $\mathbf{x}$ is fractional (Section 4.1). For ProductMIO, we show that the same greedy approach does not solve the second-stage problem in the fractional case (Section 4.2). For both formulations, when $\mathbf{x}$ is binary, we characterize the primal and dual solutions in closed form (Section 4.3). Lastly, in Section 4.4, we briefly describe our overall algorithmic approach to solving the assortment optimization problem, which involves solving the Benders reformulation of the relaxed problem, followed by the Benders reformulation of the integer problem.

4.1. Benders reformulation of the SplitMIO relaxation

The Benders reformulation of the LO relaxation of SplitMIO can be written as

$$\underset{\mathbf{x},\boldsymbol{\theta}}{\text{maximize}} \quad \sum_{t\in F} \lambda_t \theta_t \tag{9a}$$

$$\text{subject to} \quad \theta_t \le G_t(\mathbf{x}), \quad \forall\ t \in F, \tag{9b}$$

$$\mathbf{x} \in [0,1]^n, \tag{9c}$$

where the function $G_t(\mathbf{x})$ is the optimal value of the following subproblem for tree t:

$$G_t(\mathbf{x}) = \underset{\mathbf{y}_t}{\text{maximize}} \quad \sum_{\ell\in\mathbf{leaves}(t)} r_{t,\ell}\cdot y_{t,\ell} \tag{10a}$$

$$\text{subject to} \quad \sum_{\ell\in\mathbf{leaves}(t)} y_{t,\ell} = 1, \tag{10b}$$

$$\sum_{\ell\in\mathbf{left}(s)} y_{t,\ell} \le x_{v(t,s)}, \quad \forall\ s\in\mathbf{splits}(t), \tag{10c}$$

$$\sum_{\ell\in\mathbf{right}(s)} y_{t,\ell} \le 1 - x_{v(t,s)}, \quad \forall\ s\in\mathbf{splits}(t), \tag{10d}$$

$$y_{t,\ell} \ge 0, \quad \forall\ \ell\in\mathbf{leaves}(t). \tag{10e}$$

We now present a greedy algorithm for solving problem (10) in Algorithm 1. This algorithm requires as input an ordering τ of the leaves in nondecreasing revenue. In particular, we require a bijection $\tau : \{1,\dots,|\mathbf{leaves}(t)|\} \to \mathbf{leaves}(t)$ such that $r_{t,\tau(1)} \ge r_{t,\tau(2)} \ge \cdots \ge r_{t,\tau(|\mathbf{leaves}(t)|)}$, i.e., an ordering of leaves in nondecreasing revenue. In addition, in the definition of Algorithm 1, we use $\mathbf{LS}(\ell)$ and $\mathbf{RS}(\ell)$ to denote the sets of left and right splits of ℓ, respectively, which are defined as

$$\mathbf{LS}(\ell) = \{s\in\mathbf{splits}(t) \mid \ell\in\mathbf{left}(s)\} \quad \text{and} \quad \mathbf{RS}(\ell) = \{s\in\mathbf{splits}(t) \mid \ell\in\mathbf{right}(s)\}.$$

In words, $\mathbf{LS}(\ell)$ is the set of splits for which we must proceed to the left in order to be able to reach ℓ, and $\mathbf{RS}(\ell)$ is the set of splits for which we must proceed to the right to reach ℓ. A split $s\in\mathbf{LS}(\ell)$ if and only if $\ell\in\mathbf{left}(s)$, and similarly, $s\in\mathbf{RS}(\ell)$ if and only if $\ell\in\mathbf{right}(s)$.

Intuitively, this algorithm progresses through the leaves from highest to lowest revenue, and sets the $y_{t,\ell}$ variable of each leaf ℓ to the highest value it can be set to without violating the left and right split constraints (10c) and (10d) and without violating the constraint $\sum_{\ell\in\mathbf{leaves}(t)} y_{t,\ell} \le 1$. At each iteration, the algorithm additionally keeps track of which constraint became tight through the event set $\mathcal{E}$. An A_s event indicates that the left split constraint (10c) for split s became tight; a B_s event indicates that the right split constraint (10d) for split s became tight; and a C event indicates that the constraint $\sum_{\ell\in\mathbf{leaves}(t)} y_{t,\ell} \le 1$ became tight. When a C event is not triggered, Algorithm 1 looks for the split which has the least remaining capacity (line 15). In the case that

the arg min is not unique and there are two or more splits that are tied, we break ties by choosing the split s with the lowest depth $d(s)$ (i.e., the split closest to the root node of the tree).

The function f keeps track of which leaf ℓ was being checked when an A_s / B_s / C event occurred. We note that both $\mathcal{E}$ and f are not needed to find the primal solution, but they are essential to determining the dual solution in the dual procedure (Algorithm 2, which we will define shortly).

Algorithm 1 Primal greedy algorithm for SplitMIO.

Require: Bijection $\tau : \{1, \ldots, |\mathbf{leaves}(t)|\} \to \mathbf{leaves}(t)$ such that $r_{t,\tau(1)} \geq r_{t,\tau(2)} \geq \cdots \geq r_{t,\tau(|\mathbf{leaves}(t)|)}$

Initialize $y_{t,\ell} \leftarrow 0$ for each $\ell \in \mathbf{leaves}(t)$.
for $i = 1, \ldots, |\mathbf{leaves}(t)|$ **do**
  Set $q_C \leftarrow 1 - \sum_{j=1}^{i-1} y_{t,\tau(j)}$.
  for $s \in \mathbf{LS}(\tau(i))$ **do**
    Set $q_s \leftarrow x_{v(t,s)} - \sum_{1 \leq j \leq i-1 : \tau(j) \in \mathbf{left}(s)} y_{t,\tau(j)}$
  for $s \in \mathbf{RS}(\tau(i))$ **do**
    Set $q_s \leftarrow 1 - x_{v(t,s)} - \sum_{1 \leq j \leq i-1 : \tau(j) \in \mathbf{right}(s)} y_{t,\tau(j)}$
  Set $q_{A,B} \leftarrow \min_{s \in \mathbf{LS}(\tau(i)) \cup \mathbf{RS}(\tau(i))} q_s$
  Set $q^* \leftarrow \min\{q_C, q_{A,B}\}$
  Set $y_{t,\tau(i)} \leftarrow q^*$
  if $q^* = q_C$ **then**
    Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{C\}$.
    Set $f(C) = \tau(i)$.
  else
    Set $s^* \leftarrow \arg\min_{s \in \mathbf{LS}(\tau(i)) \cup \mathbf{RS}(\tau(i))} q_s$
    if $s^* \in \mathbf{LS}(\tau(i))$ **then**
      Set $e = A_{s^*}$
    else
      Set $e = B_{s^*}$
    if $e \notin \mathcal{E}$ **then**
      Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$.
      Set $f(e) = \tau(i)$.

It turns out that Algorithm 1 returns a feasible solution that is an extreme point of the polyhedron defined in problem (10). We state the result formally as the following theorem.

Theorem 2. *Fix $t \in F$. Let $\mathbf{y}_t$ be a solution to problem* (10) *produced by Algorithm 1. Then: (a) $\mathbf{y}_t$ is a feasible solution to problem* (10)*; and (b) $\mathbf{y}_t$ is an extreme point of the feasible region of problem* (10).

Theorem 2 implies that problem (10) is feasible, and since the problem is bounded, it has a finite optimal value. By strong duality, the optimal value of problem (10) is equal to the optimal value of its dual:

$$\underset{\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t}{\text{minimize}} \quad \sum_{s \in \mathbf{splits}(t)} x_{v(t,s)} \cdot \alpha_{t,s} + \sum_{s \in \mathbf{splits}(t)} (1 - x_{v(t,s)}) \beta_{t,s} + \gamma_t \tag{11a}$$

$$\text{subject to} \quad \sum_{s : \ell \in \mathbf{left}(s)} \alpha_{t,s} + \sum_{s : \ell \in \mathbf{right}(s)} \beta_{t,s} + \gamma_t \geq r_{t,\ell}, \quad \forall\ \ell \in \mathbf{leaves}(t), \tag{11b}$$

$$\alpha_{t,s} \geq 0, \quad \forall\ s \in \mathbf{splits}(t), \tag{11c}$$

$$\beta_{t,s} \geq 0, \quad \forall\ s \in \mathbf{splits}(t). \tag{11d}$$

Letting $\mathcal{D}_{t,\text{SPLITMIO}}$ denote the set of feasible solutions to the dual subproblem (11), we can formulate the master problem (9) as

$$\begin{aligned}
\underset{\mathbf{x},\boldsymbol{\theta}}{\text{maximize}} \quad & \sum_{t \in F} \lambda_t \theta_t && \text{(12a)}\\
\text{subject to} \quad & \theta_t \leq \sum_{s \in \mathbf{splits}(t)} x_{v(t,s)} \cdot \alpha_{t,s} + \sum_{s \in \mathbf{splits}(t)} (1 - x_{v(t,s)}) \beta_{t,s} + \gamma_t, \\
& \qquad \forall\ (\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t) \in \mathcal{D}_{t,\text{SPLITMIO}}, && \text{(12b)}\\
& \mathbf{x} \in [0,1]^n. && \text{(12c)}
\end{aligned}$$

The value of this formulation, relative to the original formulation, is that we have replaced the $(\mathbf{y}_t)_{t \in F}$ variables and the constraints that link them to the $\mathbf{x}$ variables, with a large family of constraints in terms of $\mathbf{x}$. Although this new formulation is still challenging, the advantage of this formulation is that it is suited to constraint generation.

The constraint generation approach to solving problem (12) involves starting the problem with no constraints and then, for each $t \in F$, checking whether constraint (12b) is violated. If constraint (12b) is not violated for any $t \in F$, then we conclude that the current solution $\mathbf{x}$ is optimal. Otherwise, for any $t \in F$ such that constraint (12b) is violated, we add the constraint corresponding to the $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ solution at which the violation occurred, and solve the problem again to obtain a new $\mathbf{x}$. The procedure then repeats at the new $\mathbf{x}$ solution until no more violated constraints have been found.

The crucial step to solving this problem is being able to solve the dual subproblem (11); that is, for a fixed $t \in F$, either asserting that the current solution $\mathbf{x}$ satisfies constraint (12b) for all $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t) \in \mathcal{D}_{t,\text{SPLITMIO}}$ or identifying a $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ at which constraint (12b) is violated. This amounts to solving the dual subproblem (11) and comparing its objective value to θ_t.

Fortunately, it turns out that we can solve the dual subproblem (11) using a specialized algorithm, in the same way that we can solve the primal subproblem (10) using Algorithm 1. Algorithm 2 uses auxiliary information obtained during the execution of Algorithm 1. In the definition of Algorithm 2, we use $d(s)$ to denote the depth of an arbitrary split, where the root split corresponds to a depth of 1, and $d_{\max} = \max_{s \in \mathbf{splits}(t)} d(s)$ is the depth of the deepest split in the tree. In addition, we use $\mathbf{splits}(t,d) = \{s \in \mathbf{splits}(t) \mid d(s) = d\}$ to denote the set of all splits at a particular depth d. Algorithm 2 proceeds sequentially from the root to the deepest splits, updating the dual variables $\alpha_{t,s}$ and $\beta_{t,s}$ for each split s based on the tree topology and the auxiliary information $\mathcal{E}$ and f obtained from Algorithm 1. When $A_s \in \mathcal{E}$, constraint (10c) in the primal problem is tight,

whereas constraint (10d) is not. This implies that the dual variable $\beta_{t,s}$, associated with the latter constraint, must remain zero. The other dual variable, $\alpha_{t,s}$, is then updated according to constraint (11b) in the dual problem. Specifically, the auxiliary function f in Algorithm 1 tracks the leaf node $\ell = f(A_s)$ with nonzero $y_{t,\ell}$ when A_s occurs. Since the corresponding constraint for ℓ in constraint (11b) is tight, it is used to calibrate $\alpha_{t,s}$. A symmetric argument applies when $B_s \in \mathcal{E}$ instead. We provide a worked example of the execution of both Algorithms 1 and 2 in Appendix B.

Algorithm 2 Dual greedy algorithm for SplitMIO.

Initialize $\alpha_{t,s} \leftarrow 0, \beta_{t,s} \leftarrow 0$ for all $s \in \mathbf{splits}(t)$, $\gamma_t \leftarrow 0$
Set $\gamma \leftarrow r_{f(C)}$
for $d = 1, \ldots, d_{\max}$ **do**
  for $s \in \mathbf{splits}(t, d)$ **do**
    if $A_s \in \mathcal{E}$ **then**
      Set $\alpha_{t,s} \leftarrow r_{t,f(A_s)} - \gamma_t - \sum_{s' \in \mathbf{LS}(f(A_s)): A_{s'} \in \mathcal{E}, d(s') < d} \alpha_{t,s'} - \sum_{s' \in \mathbf{RS}(f(A_s)): B_{s'} \in \mathcal{E}, d(s') < d} \beta_{t,s'}$
    if $B_s \in \mathcal{E}$ **then**
      Set $\beta_{t,s} \leftarrow r_{t,f(B_s)} - \gamma_t - \sum_{s' \in \mathbf{LS}(f(A_s)): A_{s'} \in \mathcal{E}, d(s') < d} \alpha_{t,s'} - \sum_{s' \in \mathbf{RS}(f(A_s)): B_{s'} \in \mathcal{E}, d(s') < d} \beta_{t,s'}$

We can show that the dual solution produced by Algorithm 2 is a feasible extreme point solution of the dual subproblem (11).

Theorem 3. *Fix $t \in F$. Let $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ be a solution to problem* (11) *produced by Algorithm 2. Then: (a) $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a feasible solution to problem* (11)*; and (b) $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is an extreme point of the feasible region of problem* (11)*.*

Lastly, and most importantly, we show that the solutions produced by Algorithms 1 and 2 are optimal for their respective problems. Thus, Algorithm 2 is a valid procedure for identifying values of $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ at which constraint (12b) is violated.

Theorem 4. *Fix $t \in F$. Let $\mathbf{y}_t$ be a solution to problem* (10) *produced by Algorithm 1 and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ be a solution to problem* (11) *produced by Algorithm 2. Then: (a) $\mathbf{y}_t$ is an optimal solution to problem* (10)*; and (b) $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is an optimal solution to problem* (11)*.*

The proof of this result follows by verifying that the two solutions satisfy complementary slackness.

Before continuing, we pause to make two remarks. First, Algorithms 1 and 2 can be viewed as the generalization of the algorithms arising in the Benders decomposition approach to the ranking-based assortment optimization problem (Bertsimas and Mišić 2019). The results of that paper show that the primal subproblem of the MIO formulation in Bertsimas and Mišić (2019) can be solved via a greedy algorithm (analogous to Algorithm 1) and the dual subproblem can be solved via an algorithm that uses information from the primal algorithm (analogous to Algorithm 2). This generalization is not straightforward. The main challenge in this generalization is designing the sequence

of updates in the greedy algorithm according to the tree topology. For example, in Algorithm 1, one considers all left/right splits and the y values of their left/right leaves when constructing the lowest upper bound of y_ℓ for each leaf node ℓ. Also, as shown in Algorithm 2, the dual variables $\alpha_{t,s}$ and $\beta_{t,s}$ have to be updated according to the tree topology and the events $A_{s'}$ and $B_{s'}$ of the split s' with smaller depth. In contrast, in the ranking-based assortment optimization problem, one only needs to calculate the "capacities" (the q_s values in Algorithm 1) by subtracting the y values of the preceding products in the ranking, which does not inherit any topological structure. For these reasons, the primal and dual Benders subproblems for the decision forest assortment problem are much more challenging than those of the ranking-based assortment problem.

Second, while the dual problem (11) can be solved directly using commercial software such as Gurobi (Gurobi Optimization, LLC 2024), the proposed greedy algorithm is generally more efficient. In Appendix D.3, we numerically demonstrate its efficiency, showing that when solving the relaxed SplitMIO problem, the greedy algorithm can reduce runtime by up to 90% compared to the approach where each constraint is added via solving the dual subproblem using Gurobi.

4.2. Benders reformulation of the ProductMIO relaxation

We also consider a Benders reformulation of the linear relaxation of ProductMIO. The Benders master problem is given by formulation (9) where the function $G_t(\mathbf{x})$ is defined as the optimal value of the ProductMIO subproblem for tree t. To aid in the definition of the subproblem, let $P(t)$ denote the set of products that appear in the splits of tree t:

$$P(t) = \{i \in \mathcal{N} \mid i = v(t,s) \text{ for some } s \in \mathbf{splits}(t)\}.$$

With a slight abuse of notation, let $\mathbf{left}(i)$ denote the set of leaves for which product i must be included in the assortment for those leaves to be reached, and similarly, let $\mathbf{right}(i)$ denote the set of leaves for which product i must be excluded from the assortment for those leaves to be reached; formally,

$$\mathbf{left}(i) = \bigcup_{s \in \mathbf{splits}(t): v(t,s)=i} \mathbf{left}(s), \qquad \mathbf{right}(i) = \bigcup_{s \in \mathbf{splits}(t): v(t,s)=i} \mathbf{right}(s).$$

With these definitions, we can write down the ProductMIO subproblem as follows:

$$\begin{aligned}
G_t(\mathbf{x}) = \quad & \underset{\mathbf{y}_t}{\text{maximize}} && \sum_{\ell \in \mathbf{leaves}(t)} r_{t,\ell} \cdot y_{t,\ell} && \text{(13a)}\\
& \text{subject to} && \sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} = 1, && \text{(13b)}\\
& && \sum_{\ell \in \mathbf{left}(i)} y_{t,\ell} \le x_i, \quad \forall\ i \in P(t), && \text{(13c)}\\
& && \sum_{\ell \in \mathbf{right}(i)} y_{t,\ell} \le 1 - x_i, \quad \forall\ i \in P(t), && \text{(13d)}\\
& && y_{t,\ell} \ge 0, \quad \forall\ \ell \in \mathbf{leaves}(t). && \text{(13e)}
\end{aligned}$$

In the same way as SplitMIO, one can consider solving problem (13) using a greedy approach, where one iterates through the leaves from highest to lowest revenue, and sets each leaf's $y_{t,\ell}$ variable to the highest possible value without violating any of the constraints. Unlike SplitMIO, it unfortunately turns out that this greedy approach is not always optimal. We formalize this claim as a proposition in Section C of the appendix, supported by a counterexample.

4.3. Bender Cuts for Integer Master Solutions

We further propose closed form expressions for the structure of the optimal primal and dual Benders subproblem solutions for integer solutions $\mathbf{x}$.

4.3.1. SplitMIO Our results in Section 4.1 for obtaining primal and dual solutions for the subproblem of SplitMIO apply for any $\mathbf{x} \in [0,1]^n$; in particular, they apply for fractional choices of $\mathbf{x}$, thus allowing us to solve the Benders reformulation of the relaxation of SplitMIO.

In the special case that $\mathbf{x}$ is a candidate integer solution of SplitMIO, we can find optimal solutions to the primal and dual subproblems of SplitMIO in closed form.

Theorem 5. *Fix $t \in F$, and let $\mathbf{x} \in \{0,1\}^n$. Define the primal subproblem solution $\mathbf{y}_t$ as*

$$y_{t,\ell} = \begin{cases} 1 & \text{if } \ell = \ell^*, \\ 0 & \text{if } \ell \neq \ell^*, \end{cases}$$

where ℓ^* denotes the leaf that the assortment encoded by $\mathbf{x}$ is mapped to. Define the dual subproblem solution $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ as

$$\alpha_{t,s} = \begin{cases} \max\{0, \max_{\ell \in \mathbf{left}(s)} r_{t,\ell} - r_{t,\ell^*}\} & \text{if } s \in \mathbf{RS}(\ell^*), \\ 0 & \text{otherwise}, \end{cases}$$
$$\beta_{t,s} = \begin{cases} \max\{0, \max_{\ell \in \mathbf{right}(s)} r_{t,\ell} - r_{t,\ell^*}\} & \text{if } s \in \mathbf{LS}(\ell^*), \\ 0 & \text{otherwise}, \end{cases}$$
$$\gamma_t = r_{t,\ell^*}.$$

Then: (a) $\mathbf{y}_t$ is a feasible solution to problem (10); (b) $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a feasible solution to problem (11); and (c) $\mathbf{y}_t$ and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ are optimal for problems (10) and (11), respectively.

The significance of Theorem 5 is that it provides a simpler means to checking for violated constraints when $\mathbf{x}$ is binary than applying Algorithms 1 and 2. In particular, for the integer version of SplitMIO, a similar derivation as in Section 4.1 leads us to the following Benders reformulation of the integer problem for the SplitMIO formulation:

$$\underset{\mathbf{x},\boldsymbol{\theta}}{\text{maximize}} \quad \sum_{t \in F} \lambda_t \theta_t \tag{14a}$$
$$\text{subject to} \quad \theta_t \leq \sum_{s \in \mathbf{splits}(t)} x_{v(t,s)} \cdot \alpha_{t,s} + \sum_{s \in \mathbf{splits}(t)} (1 - x_{v(t,s)}) \beta_{t,s} + \gamma_t,$$
$$\forall\ (\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t) \in \mathcal{D}_{t,\text{SplitMIO}}, \tag{14b}$$
$$\mathbf{x} \in \{0,1\}^n. \tag{14c}$$

To check whether constraint (14b) is violated for a particular $\mathbf{x}$ and a tree t, we can simply use Theorem 5 to determine the optimal value of the subproblem, and compare it against θ_t; if the constraint corresponding to the dual solution of Theorem 5 is violated, we add that constraint to the problem. In our implementation of Benders decomposition, we embed the constraint generation process for the integer problem (14) within the branch-and-bound tree, using a technique referred to as *lazy constraint generation*; we discuss this more in Section 4.4.

4.3.2. ProductMIO We also consider ProductMIO. We begin by writing down the dual of the subproblem, for which we need to define several additional sets. We let $\mathbf{LP}(\ell)$ denote the set of "left products" of leaf ℓ (those products that must be included in the assortment for leaf ℓ to be reached), and let $\mathbf{RP}(\ell)$ denote the set of "right products" of leaf ℓ (those products that must be excluded from the assortment for leaf ℓ to be reached). Note that $\ell \in \mathbf{left}(i)$ if and only if $i \in \mathbf{LP}(\ell)$, and similarly $\ell \in \mathbf{right}(i)$ if and only if $i \in \mathbf{RP}(\ell)$.

With these definitions, the dual of the primal subproblem (13) is

$$\underset{\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t}{\text{minimize}} \quad \sum_{i \in P(t)} \alpha_{t,i} x_i + \sum_{i \in P(t)} \beta_{t,i}(1 - x_i) + \gamma_t \tag{15a}$$

$$\text{subject to} \quad \sum_{i \in \mathbf{LP}(\ell)} \alpha_{t,i} + \sum_{i \in \mathbf{RP}(\ell)} \beta_{t,i} + \gamma_t \geq r_\ell, \quad \forall\ \ell \in \mathbf{leaves}(t), \tag{15b}$$

$$\alpha_{t,i} \geq 0, \quad \forall\ i \in P(t), \tag{15c}$$

$$\beta_{t,i} \geq 0, \quad \forall\ i \in P(t). \tag{15d}$$

In the case that $\mathbf{x}$ is integer, we can obtain optimal solutions to the primal subproblem (13) and its dual (15) in closed form.

Theorem 6. *Fix $t \in F$ and let $\mathbf{x} \in \{0,1\}^n$. Let $\mathbf{y}_t$ be defined as in Theorem 5 and let $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ be defined as*

$$\alpha_{t,i} = \begin{cases} \max\{0, \max_{\ell \in \mathbf{left}(i)} r_{t,\ell} - r_{t,\ell^*}\}, & \text{if } i \in \mathbf{RP}(\ell^*), \\ 0 & \text{otherwise}, \end{cases} \tag{16}$$

$$\beta_{t,i} = \begin{cases} \max\{0, \max_{\ell \in \mathbf{right}(i)} r_{t,\ell} - r_{t,\ell^*}\}, & \text{if } i \in \mathbf{LP}(\ell^*), \\ 0 & \text{otherwise}, \end{cases} \tag{17}$$

$$\gamma_t = r_{t,\ell^*}. \tag{18}$$

Then: (a) $\mathbf{y}_t$ is a feasible solution for problem (13); (b) $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a feasible solution for problem (15); and (c) $\mathbf{y}_t$ and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ are optimal for problems (13) and (15), respectively.

4.4. Overall Benders algorithm

We conclude Section 4 by summarizing how the results are used. In our overall algorithmic approach below, we first focus on SplitMIO, as the subproblem can be solved for the formulation when $\mathbf{x}$ is either fractional or binary.

1. *Relaxation phase.* We first solve the relaxed problem (12) using ordinary constraint generation. Given a solution $\mathbf{x} \in [0,1]^n$, we generate Benders cuts by running the primal-dual procedure (Algorithm 1 followed by Algorithm 2).

2. *Integer phase.* In the integer phase, we add all of the Benders cuts generated in the relaxation phase to the integer version of problem (12). We then solve the problem as an integer optimization problem, where we generate Benders cuts for integer solutions using the closed form expressions in Theorem 5. We add these cuts using *lazy constraint generation*. That is, we solve the master problem using a single branch-and-bound tree, and we check whether the main constraint (12b) of the Benders formulation is violated at every integer solution generated in the branch-and-bound tree.

For ProductMIO, as the subproblem can only be solved when $\mathbf{x}$ is binary, its overall Benders algorithm directly starts with the *Integer Phase*.

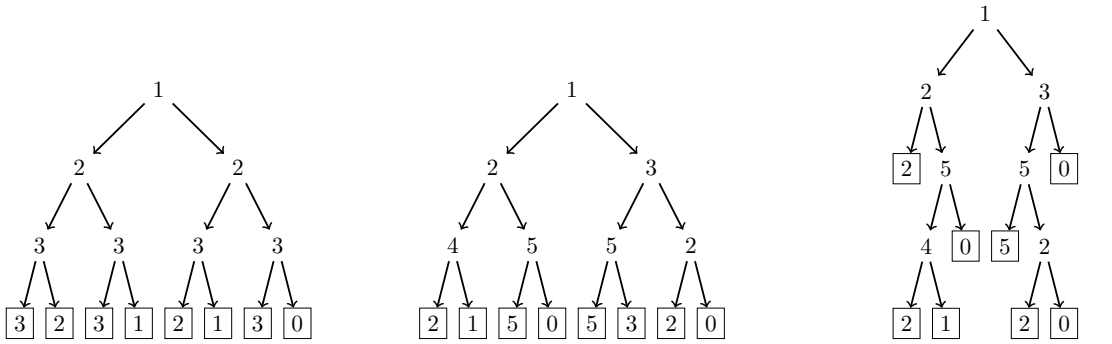
We conclude this section by highlighting the advantages of the SplitMIO formulation in the large-scale regime. At first glance, following Section 3, ProductMIO may appear preferable to SplitMIO since it is a tighter formulation (Proposition 3) with fewer constraints. However, in large-scale settings, the proposed primal-dual greedy algorithm enables significantly faster computation of the linear relaxation for SplitMIO compared to ProductMIO. As a result, as outlined above, SplitMIO can benefit from a more efficient warm start during the relaxation phase. Additionally, the objective value of the linear relaxation provides a valid upper bound for the assortment problem (2). This upper bound is particularly valuable in large-scale instances where solving SplitMIO or ProductMIO exactly may be impractical. A strong upper bound helps reduce the number of branches explored in the branch-and-bound procedure and serves as a benchmark for assessing the suboptimality of a given assortment solution. In Appendix D.3, we demonstrate that, with the greedy algorithm, solving the relaxed SplitMIO problem can be up to 95% faster than solving the relaxed ProductMIO while providing comparable upper bounds.

5. Numerical Experiments with Synthetic Data

In this section, we examine the formulation strength of SplitMIO and ProductMIO (Section 5.2) and demonstrate the scalability of the Benders decomposition approach(Section 5.3). We focus on synthetically generated instances so that we can scale the problem size. In Appendix D, we report our implementation details and include additional numerical results.

5.1. Background

To test our method, we generate three different families of synthetic decision forest instances, which differ in the topology of the trees and the products that appear in the splits:



(a) T1 tree ($d = 3$). (b) T2 tree ($d = 3$). (c) T3 tree ($L = 8$).

Figure 2 Examples of T1, T2 and T3 trees.

1. **T1 instances**. A T1 forest consists of balanced trees of depth d (i.e., trees where all leaves are at depth $d+1$). For each tree, we sample d products $i_1, \ldots, i_d$ uniformly without replacement from $\mathcal{N}$. Then, for every depth $d' \in \{1, \ldots, d\}$, we set the split product $v(t, s)$ as $v(t, s) = i_{d'}$ for every split s that is at depth d'.

2. **T2 instances**. A T2 forest consists of balanced trees of depth d. For each tree, we set the split products at each split iteratively, starting at the root, in the following manner: (a) Initialize $d' = 1$; (b) For all splits s at depth d', set $v(s, t) = i_s$ where i_s is drawn uniformly at random from the set $\mathcal{N} \setminus \cup_{s' \in A(s)} \{v(t, s')\}$, where $A(s)$ is the set of ancestor splits to split s (i.e., all splits appearing on the path from the root node to split s); (c) Increment $d' \leftarrow d' + 1$; and (d) If $d' > d$, stop; otherwise, return to Step (b).

3. **T3 instances**. A T3 forest consists of unbalanced trees with L leaves. Each tree is generated according to the following iterative procedure: (a) Initialize t to a tree consisting of a single leaf; (b) Select a leaf ℓ uniformly at random from $\mathbf{leaves}(t)$, and replace it with a split s and two child leaves ℓ_1, ℓ_2. For split s, set $v(s, t) = i_s$ where i_s is drawn uniformly at random from $\mathcal{N} \setminus \cup_{s' \in A(s)} \{v(t, s')\}$; (c) If $|\mathbf{leaves}(t)| = L$, terminate; otherwise, return to Step (b).

For all three types of forests, we generate the purchase decision $c(t, \ell)$ for each leaf ℓ in each tree t in the following way: for each leaf ℓ, we uniformly at random choose a product $i \in \cup_{s \in \mathbf{LS}(\ell)} \{v(t, s)\} \cup \{0\}$. In words, the purchase decision is chosen to be consistent with the products that are known to be in the assortment if leaf ℓ is reached. Figure 2 shows an example of each type of tree (T1, T2, and T3). Given a forest of any of the three types above, we generate the customer type probability vector $\boldsymbol{\lambda} = (\lambda_t)_{t \in F}$ by drawing it uniformly from the $(|F| - 1)-$dimensional unit simplex.

In our experiments, we fix the number of products $n = 100$ and vary the number of trees $|F| \in \{50, 100, 200, 500\}$, and the number of leaves $|\mathbf{leaves}(t)| \in \{8, 16, 32, 64\}$. (Note that the chosen values for $|\mathbf{leaves}(t)|$ correspond to depths of $\{3, 4, 5, 6\}$ for the T1 and T2 instances.) For each combination of n, $|F|$ and $|\mathbf{leaves}(t)|$ and each type of instance (T1, T2 and T3) we randomly

Type T1			Type T2			Type T3		
$\|F\|$	$G^{\text{int}}_{\text{SplitMIO}}$	$G^{\text{int}}_{\text{ProductMIO}}$	$\|F\|$	$G^{\text{int}}_{\text{SplitMIO}}$	$G^{\text{int}}_{\text{ProductMIO}}$	$\|F\|$	$G^{\text{int}}_{\text{SplitMIO}}$	$G^{\text{int}}_{\text{ProductMIO}}$
50	0.9	0.0	50	0.2	0.2	50	0.2	0.2
100	2.5	0.1	100	1.0	1.0	100	0.5	0.5
200	5.6	0.2	200	5.4	5.3	200	4.1	3.9
500	15.8	3.3	500	16.7	16.4	500	14.2	14.0

Table 1 Average integrality gap of SplitMIO and ProductMIO for T1, T2 and T3 instances.

generate 20 problem instances, where a problem instance consists of a decision forest model and the product marginal revenues $\rho_1, \ldots, \rho_n$. For each instance, the decision forest model is generated according to the process described above and the product revenues are sampled uniformly with replacement from the set $\{1, \ldots, 100\}$.

5.2. Experiment #1: Formulation Strength

Our first experiment is to simply understand how the two formulations – SplitMIO and ProductMIO– compare in terms of formulation strength. Recall from Proposition 3 that ProductMIO is at least as strong as SplitMIO. For a given instance and a given formulation $\mathcal{M}$ (one of SplitMIO and ProductMIO), we define the integrality gap $G^{\text{int}}_{\mathcal{M}} \equiv 100\% \times (Z_{\mathcal{M}} - Z^*)/Z^*$, where Z^* is the optimal objective value of the integer problem and $Z_{\mathcal{M}}$ is the optimal objective of the LO relaxation. We consider the T1, T2 and T3 instances with $n = 100$, $|F| \in \{50, 100, 200, 500\}$ and $|\mathbf{leaves}(t)| = 8$. We restrict our focus to instances with $n = 100$ and $|\mathbf{leaves}(t)| = 8$, as the optimal value Z^* of the integer problem could be computed within one hour for these instances.

Table 1 displays the average integrality gap of each of the two formulations for each combination of n and $|F|$ and each instance type. From this table, we can see that the integrality gap of both SplitMIO and ProductMIO is in general about 0 to 17%. Note that the difference between ProductMIO and SplitMIO is most pronounced for the T1 instances, as the decision forests in these instances exhibit the highest degree of repetition of products within the splits of a tree. In contrast, the difference is much smaller for the T2 and T3 instances, where there is less repetition of products within the splits of the tree (as the trees are not forced to have the same product appear on all of the splits at a particular depth). Note that these results align with our theoretical analysis of ProductMIO and SplitMIO (specifically Proposition 3). Note also that when a tree does not have any repeated products in its splits, the corresponding constraints of ProductMIO and SplitMIO are the same, and ProductMIO does not confer an advantage over SplitMIO. Thus, as one constructs trees with less and less repetition in the splits, the improvement in the integrality gap of ProductMIO over SplitMIO should become smaller, with the two formulations having the same gap in the most extreme case where no product is repeated.

We also test the tractability of SplitMIO and ProductMIO when they are solved as integer programs (i.e., not as relaxation). We present the results in Section D.1.

5.3. Experiment #2: Benders Decomposition for Large-Scale Problems

In this experiment, we report on the performance of our Benders decomposition approach for solving large scale instances of SplitMIO. We focus on the SplitMIO formulation, as we are able to efficiently generate Benders cuts for both fractional and integral values of $\mathbf{x}$.

We deviate from our previous experiments by generating a collection of T3 instances with $n \in \{200, 500, 1000, 2000, 3000\}$, $|F| = 500$ trees and $|\mathbf{leaves}(t)| = 512$ leaves. As before, the marginal revenue ρ_i of each product i is chosen uniformly at random from $\{1, \ldots, 100\}$. For each value of n, we generate 5 instances. For each instance, we solve the SplitMIO problem subject to the constraint $\sum_{i=1}^{n} x_i = b$, where b is set as $b = \mu n$ and we vary $\mu \in \{0.02, 0.04, 0.06, 0.08, 0.10, 0.12\}$.

We compare three different methods: the two-phase Benders method described in Section 4.4, using the SplitMIO cut results (Section 4.1 and Section 4.3); the divide-and-conquer (D&C) heuristic; and the direct solution approach, where we attempt to directly solve the full SplitMIO formulation using Gurobi (Gurobi Optimization, LLC 2024). The D&C heuristic is a type of local search heuristic proposed in the product line design literature (see Green and Krieger 1993; see also Belloni et al. 2008). In this heuristic, one iterates through the b products currently in the assortment, and replaces a single product with the product outside of the assortment that leads to the highest improvement in the expected revenue; this process repeats until the assortment can no longer be improved. We choose the initial assortment uniformly at random from the collection of assortments of size b. For each instance, we repeat the D&C heuristic 10 times, and retain the best solution. We do not impose a time limit on the D&C heuristic. For the Benders approach, we impose a time limit of one hour on the LO phase, and impose a time limit of one hour on the integer phase. For the direct solution approach, we impose a time limit of two hours, in order to be comparable to the Benders approach.

Table 2 reports the performance of the three methods – the Benders approach, the D&C heuristic and direct solution of SplitMIO– across all combinations of n and μ. We consider several metrics. The metric $G_{\mathcal{M}}$ is defined as $G_{\mathcal{M}} = (Z' - Z_{\mathcal{M}})/Z' \times 100\%$, i.e., it is the optimality gap of the solution obtained by method $\mathcal{M}$ – either the Benders approach, the direct approach or the D&C heuristic – relative to the objective value of the best solution obtained out of the three methods, which is indicated by Z'. Lower values of $G_{\mathcal{M}}$ indicate that the approach tends to deliver solutions that are close to being the best out of the three methods, and a value of 0% implies that the solution obtained by the approach is the best (or tied for the best) out of the three methods. The metric $T_{\mathcal{M}}$ indicates the solution time required for approach $\mathcal{M}$ in seconds. Finally, for the Benders and direct approaches, we compute the optimality gap $O_{\mathcal{M}}$, which is defined as $O_{\mathcal{M}} = (Z_{\text{UB},\mathcal{M}} - Z_{\text{LB},\mathcal{M}})/Z_{\text{UB},\mathcal{M}} \times 100\%$, where $Z_{\text{UB},\mathcal{M}}$ and $Z_{\text{LB},\mathcal{M}}$ are the best upper and lower bounds, respectively, obtained from either the Benders or direct approach after the computation time limit is

n	μ	b	G_{Benders}	$G_{\text{D\&C}}$	G_{Direct}	T_{Benders}	$T_{\text{D\&C}}$	T_{Direct}	O_{Benders}	O_{Direct}
200	0.02	4	0.00	0.00	0.00	14.54	2.86	2060.81	0.00	0.00
200	0.04	8	0.00	0.04	0.61	187.96	4.46	7015.65	0.00	7.72
200	0.06	12	0.11	0.00	0.77	3615.82	7.34	7200.25	8.23	16.71
200	0.08	16	0.70	0.00	2.86	3612.43	9.97	7200.47	17.38	23.39
200	0.10	20	0.47	0.01	6.94	3613.19	15.07	7200.16	22.13	30.38
200	0.12	24	0.67	0.12	7.68	3614.96	18.93	7200.25	27.27	34.03
500	0.02	10	0.00	0.19	0.00	16.71	11.24	184.29	0.00	0.00
500	0.04	20	0.00	0.20	0.08	1640.78	28.93	7200.21	0.48	3.06
500	0.06	30	0.02	0.68	3.57	3630.19	65.75	7200.26	8.10	11.89
500	0.08	40	0.00	0.48	1.05	3623.51	115.36	7200.57	14.26	14.82
500	0.10	50	0.00	1.49	6.88	3625.89	177.26	7200.18	18.26	23.74
500	0.12	60	0.69	0.95	4.11	3631.14	223.61	7200.38	22.94	25.30
1000	0.02	20	0.00	0.29	0.00	18.83	47.62	156.49	0.00	0.00
1000	0.04	40	0.00	1.65	2.76	2823.19	183.33	7200.18	1.05	4.10
1000	0.06	60	0.00	2.48	6.61	3671.12	397.35	7200.18	6.16	12.53
1000	0.08	80	0.00	2.92	7.86	3662.76	687.02	7200.44	10.07	17.39
1000	0.10	100	0.00	2.31	8.65	3670.95	1052.76	7200.17	13.60	20.99
1000	0.12	120	0.00	2.13	5.92	3696.24	1552.76	7200.22	15.77	20.73
2000	0.02	40	0.00	1.36	0.00	14.55	328.25	70.02	0.00	0.00
2000	0.04	80	0.00	3.49	0.00	1774.85	1259.21	1057.28	0.21	0.00
2000	0.06	120	0.00	4.26	21.38	3774.67	2578.07	7200.16	2.42	23.29
2000	0.08	160	0.00	4.63	100.00	3806.42	4431.39	7200.28	5.21	100.00
2000	0.10	200	0.00	5.23	100.00	3925.28	6435.67	7200.17	7.16	100.00
2000	0.12	240	0.74	0.94	100.00	4319.62	10949.83	7200.16	11.91	100.00
3000	0.02	60	0.00	1.97	0.00	16.16	923.12	32.40	0.00	0.00
3000	0.04	120	0.00	4.03	0.00	1883.80	3365.35	1541.40	0.08	0.00
3000	0.06	180	0.00	5.26	0.04	4144.41	7620.89	7200.16	1.81	1.80
3000	0.08	240	0.00	4.55	99.98	4554.74	13624.07	7209.41	4.23	99.99
3000	0.10	300	0.00	4.04	99.98	5013.31	31137.18	7200.38	6.17	99.98
3000	0.12	360	0.32	1.12	99.98	6999.60	43173.66	7216.75	9.83	99.99

Table 2 Comparison of the Benders decomposition approach, the D&C heuristic and direct solution of SplitMIO in terms of solution quality, computation time and optimality gap.

exhausted. The value reported of each metric is the average over the five replications corresponding to the particular (n, μ) combination.

Comparing the performance of the Benders approach with the D&C heuristic, we can see that in general, the Benders approach is able to find better solutions than the D&C heuristic. In particular, for larger instances, G_{Benders} is lower than $G_{\text{D\&C}}$ (for example, with $n = 2000$, $\mu = 0.08$, the Benders solution is in general the best one, and the D&C solution has an expected revenue that is 4.6% worse). In addition, from a computation time standpoint, the Benders approach compares quite favorably to the D&C heuristic. While the D&C heuristic is faster for small problems with low n and/or low μ, it can require a significant amount of time for $n = 2000$ or $n = 3000$. In addition to this comparison against the D&C heuristic, in Appendix D.2, we also provide a comparison of the MIO solutions for the smaller T1, T2 and T3 instances used in the previous two sections against three other heuristic solutions; in those instances, we similarly find that solutions obtained from our MIO formulations can be significantly better than heuristic solutions.

Comparing the performance of the Benders approach with the direct solution approach, our results indicate two types of behavior. The first type of behavior corresponds to "easy" instances. These are instances with $\mu \in \{0.02, 0.04\}$ for which it is sometimes possible to directly solve SplitMIO to optimality within the two hour time limit. For example, with $n = 2000$ and $\mu = 0.04$, all five instances are solved to optimality by the direct approach. For those instances, the Benders approach is either able to prove optimality (for example, for $n = 200$ and $\mu = 0.04$, $O_{\text{Benders}} = 0\%$) or terminate with a low optimality gap (for example, for $n = 3000$ and $\mu = 0.04$, $O_{\text{Benders}} = 0.08\%$); among all instances with $\mu \in \{0.02, 0.04\}$, the average optimality gap is no more than about 1.05%. More importantly, the solution obtained by the Benders approach is at least as good as the solution obtained after two hours of direct solution of SplitMIO, which can be seen from the fact that G_{Benders} is 0.0% in all of these (n, μ) combinations.

The second type of behavior corresponds to "hard" instances, which are the instances with $\mu \in \{0.06, 0.08, 0.10, 0.12\}$. For these instances, Gurobi generally struggles to solve the LO relaxation of SplitMIO within the two hour time limit. When this happens, the integer solution returned by Gurobi is obtained from applying heuristics before solving the root node of the branch-and-bound tree, which is often quite suboptimal. (This often results in integer solutions with an objective value of 0.0, leading to values of O_{Direct} and G_{Direct} that are close to 100%.) In contrast, the Benders approach delivers significantly better solutions; among all of these instances, G_{Benders} is within 1%, indicating that on average the Benders approach is within 1% of the best solution of each instance. In addition, O_{Benders} is generally lower than O_{Direct}, indicating that the Benders approach in general makes more progress towards proving optimality than the direct approach.

Overall, these results suggest that our Benders approach can deliver high quality solutions to large-scale instances of the assortment optimization problem in a reasonable computational time-frame that are at least as good, and often significantly better, than those obtained by the D&C heuristic or those obtained by directly solving the problem using Gurobi.

6. Numerical Experiments with Real Transaction Data

We examine the performance of the optimal assortment generated by the decision forest model on problem instances calibrated with a real-world transaction dataset. Different from Section 5, the problem instances in this section are of a smaller scale, which help us glean operational insights.

6.1. Background

We consider the IRI Academic Dataset (Bronnenberg et al. 2008), which is comprised of real-world transaction records of store sales for thirty product categories from forty-seven U.S. markets. The same dataset has been used by Jagabathula and Rusmevichientong (2019) to empirically

demonstrate the loss of rationality in consumer choice and by Chen and Mišić (2022) to evaluate the predictive performance of the decision forest model.

We follow the literature to pre-process the raw transaction data. We first aggregate items with the same vendor code as a product, a common pre-processing technique in the marketing science community (Bronnenberg and Mela 2004, Nijs et al. 2007). Following the setup in the literature (Jagabathula and Rusmevichientong 2019, Chen and Mišić 2022) and focusing on the data from the first two weeks of the calendar year 2007 due to the data volume, we select the top nine purchased items as the products and combine the remaining items as the outside/no-purchase option. We further transform the transactions into assortment-choice pairs as follows. We first call $\mathcal{T}$ the set of transactions. For each transaction $\tau \in \mathcal{T}$, we have the following information: the week of the purchase τ_{week}, the store where the transaction happened τ_{store}, the sold product τ_{prod}, and the selling price τ_{price}. Let $\mathcal{W}$ and $\mathcal{Z}$ be the nonrepeated collection of $\{\tau_{\text{week}}\}_{\tau\in\mathcal{T}}$ and $\{\tau_{\text{store}}\}_{\tau\in\mathcal{T}}$, respectively. For each week $w \in \mathcal{W}$ and store $s \in \mathcal{Z}$, we define $S_{w,s} = \bigcup_{\tau\in\mathcal{T}}\{\tau_{\text{prod}} \mid \tau_{\text{week}} = w, \tau_{\text{store}} = s\}$ as the set of products that was purchased at least once at store s during week w. The set of transactions $\mathcal{T}$ is thus transformed into the set of assortment-choice pairs as $\{(S_{\tau_{\text{week}},\tau_{\text{store}}}, \tau_{\text{prod}})\}_{\tau\in\mathcal{T}}$, which will be used for choice model estimation. We further define the marginal revenue ρ_i as the average of the historical prices $(\tau_{\text{price}})_{\tau\in\mathcal{T}:\tau_{\text{prod}}=i}$ for $i \in \mathcal{N}$.

6.2. Results: Improvement in Expected Revenue

We estimate both the ranking-based model and the decision forest model from the assortment-choice pairs $\{(S_{t_{\text{week}},t_{\text{store}}}, t_{\text{prod}})\}_{t\in\mathcal{T}}$ in each product category by maximum likelihood estimation. For details on estimating the ranking-based model, see van Ryzin and Vulcano (2014, 2017), and for the decision forest model, refer to Chen and Mišić (2022). In particular, we follow Chen and Mišić (2022) to warm start the solver by setting the initial solution as the estimated ranking-based model. For simplicity, we set the tree depth limit as three (i.e., leaves can have depth at most four). Typically, the tree depth is determined by cross validation. Chen and Mišić (2022) have reported that trees of depth three lead to good predictive performance, while deeper trees tend to overfit.

We further denote S^{RM} and S^{DF} as the optimal assortments under the estimated ranking-based model and the decision forest model, respectively. Note that we obtain S^{RM} using an integer programming approach (Bertsimas and Mišić 2019, Feldman et al. 2019). We use $Z^*_{\mathcal{C}}$ to denote the maximal expected revenue under a given choice model $\mathcal{C}$ and $Z_{\mathcal{C}}(S)$ to denote the expected revenue of assortment S. Obviously, $Z^*_{\text{RM}} = Z_{\text{RM}}(S^{\text{RM}})$ and $Z^*_{\text{DF}} = Z_{\text{DF}}(S^{\text{DF}})$. To discuss the relative performance of assortments S^{RM} and S^{DF}, we define the following two metrics: $RI_{\text{DF}} = 100\% \times \frac{Z_{\text{DF}}(S^{\text{DF}}) - Z_{\text{DF}}(S^{\text{RM}})}{Z_{\text{DF}}(S^{\text{RM}})}$ and $RI_{\text{RM}} = 100\% \times \frac{Z_{\text{RM}}(S^{\text{RM}}) - Z_{\text{RM}}(S^{\text{DF}})}{Z_{\text{RM}}(S^{\text{DF}})}$. The two metrics measure the relative improvement of the optimal assortment S^{DF} (or S^{RM}) from S^{RM} (or S^{DF}) under the decision forest

Category	RI_{DF}	RI_{RM}	Δ_{H}	Category	RI_{DF}	RI_{RM}	Δ_{H}
Beer	32.8	5.9	3	Mayonnaise	0.0	0.0	0
Blades	0.0	0.0	0	Milk	5.2	0.7	1
Carbonated Beverages	4.8	3.8	4	Mustard / Ketchup	5.6	1.1	1
Cigarettes	2.5	8.3	1	Paper Towels	1.0	2.4	2
Coffee	26.3	11.0	4	Peanut Butter	10.9	5.6	3
Cold Cereal	6.1	0.3	2	Photo	0.0	0.0	0
Deodorant	3.7	2.2	3	Salty Snacks	12.0	3.7	2
Diapers	0.0	0.0	0	Shampoo	19.0	1.4	4
Facial Tissue	0.2	0.6	1	Soup	13.0	5.0	3
Frozen Dinners	2.7	5.2	2	Spaghetti / Italian Sauce	4.9	3.3	2
Frozen Pizza	10.9	4.0	4	Sugar Substitutes	10.7	6.0	1
Household Cleaners	16.0	6.6	4	Toilet Tissue	0.0	0.0	0
Hotdogs	18.7	1.4	4	Toothbrush	0.0	0.0	0
Laundry Detergent	0.1	1.9	1	Toothpaste	2.7	0.9	2
Margarine / Butter	5.3	12.9	1	Yogurt	3.8	2.1	1
				Average over all categories	7.3	3.2	1.9

Table 3 The relative performance of the assortments generated by the decision forest model and the ranking-based model in each product category in the IRI Academic Dataset

model (or the ranking-based model). Note that both metrics are non-negative by their definitions. We also define the Hamming distance $\Delta_{\mathrm{H}} = \sum_{i=1}^{n} |\mathbb{I}[i \in S^{\mathrm{DF}}] - \mathbb{I}[i \in S^{\mathrm{RM}}]|$ to measure how different the two assortments are.

Table 3 summarizes the comparison of S^{DF} and S^{RM} in terms of expected revenue under the measures RI_{DF} and RI_{RM}. When the ground truth is the estimated decision forest model, the assortment S^{DF} can outperform S^{RM} up to 32% and with 7% improvement on average in the expected revenue. Meanwhile, when the ground truth is the estimated ranking-based model, the assortment S^{DF} only performs 3% worse than the optimal assortment S^{RM} on average. Recall that the class of the ranking-based models is equivalent to the class of choice models of random utility maximization (RUM) principle, or, the class of rational choice models (Jagabathula and Rusmevichientong 2019). Table 3 suggests that the optimal assortment generated by the decision forest model can be quite beneficial when customer choice deviates from the rational choice theory, and does not lose much even if customers are strictly rational. We also remark that on average, the two assortments S^{DF} and S^{RM} only differ from each other with Hamming distance 1.9, while they have similar sizes: the average sizes of S^{DF} and S^{RM} are 4.9 and 5.3, respectively.

Note that, in this real-data setting, the problem instances involve significantly fewer products compared to the synthetic-data setting described in Section 5. As a result, the optimal assortments S^{DF} and S^{RM} are obtained almost instantaneously (in less than one second) using the integer programming approach.

6.3. Case Study: Beer Category

To investigate why we observed high relative improvement with the assortment S^{DF} over S^{RM} in some categories, we look into the Beer category, where the relative improvement is 32.8%.

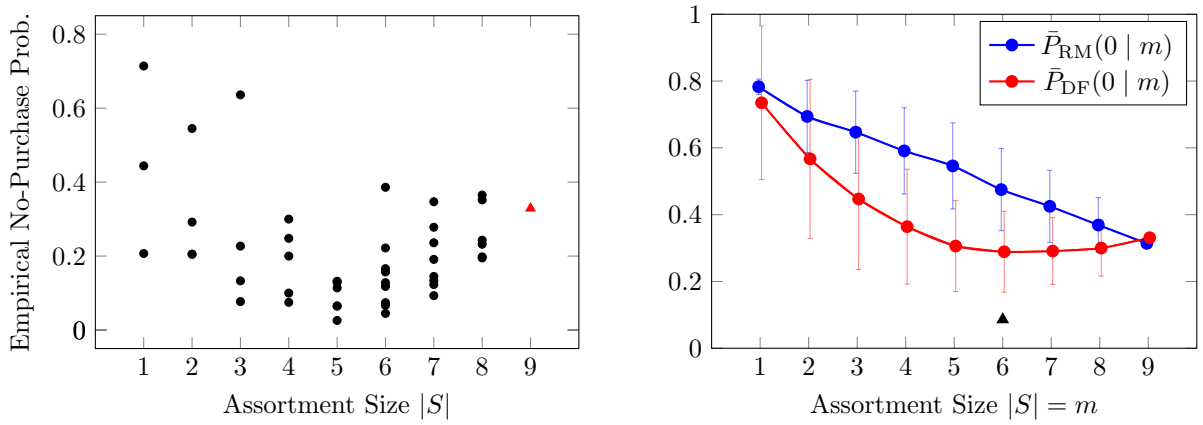


Figure 3 **(left) The choice overload effect in the Beer category. Each scatter point represents the empirical no-purchase probability $\bar{P}(0 \mid S)$ for an historical assortment $S \in \mathcal{S}_{\mathsf{hist}}$; (right) The average no-purchase probability of the estimated ranking-based and decision forest models (with the error bars as the standard deviation). The black triangle represents that $P_{\mathsf{DF}}(0 \mid S^{\mathsf{DF}}) = 8.6\%$**

We first observe that the consumer choice within the Beer category exhibits a notable phenomenon known as *choice overload*, a well-documented behavioral anomaly in the field of marketing science (Iyengar and Lepper 2000, Chernev et al. 2015, Long et al. 2023). This phenomenon describes the situation in which an abundance of available products can actually deter customers from making a purchase. Choice overload contradicts with the principles of rational choice theory. According to this theory, when a store increases the assortment size, a rational consumer should theoretically be more inclined to make a purchase from the assortment. This is because a larger assortment would increase the chance that the customer finds one product that aligns with her preference. In the context of choice modeling, a rational choice model satisfies the following:

$$P(0 \mid S) \geq P(0 \mid S') \quad \text{whenever} \quad S \subseteq S'. \tag{19}$$

Note that equation (19) directly stems from the *regularity property*, which the ranking-based model and other RUM choice models, such as the mixed-MNL model, always adhere to (Rieskamp et al. 2006, Jagabathula and Rusmevichientong 2019). However, real-world scenarios often deviate from equation (19). When provided with more options, customers may become fatigued from the search process or experience a decrease in their confidence in decision-making, leading them to opt not to make a purchase. In the consumer psychology literature, Iyengar and Lepper (2000) conducted a field experiment demonstrating that individuals are more likely to purchase gourmet jams or chocolates when presented with a smaller assortment of size six rather than a more extensive assortment of size 24 or 30.

Figure 3 (left) visually depicts the prevalence of choice overload within the Beer category of the IRI dataset. Each data point in the figure corresponds to a pair $(|S|, \bar{P}(0 \mid S))$, where $|S|$

represents the assortment size of a historical assortment $S \in \mathcal{S}_{\text{hist}} \equiv \{S_{w,s} \mid w \in \mathcal{W}, s \in \mathcal{S}\}$, and $\bar{P}(0 \mid S)$ represents the empirical probability of the no-purchase option being chosen when the assortment S was offered. For a better illustration, we have excluded assortments with very few transactions (less than or equal to 10) from Figure 3 (left). In this figure, the rightmost data point, denoted by a red triangle, corresponds to the no-purchase probability when the assortment contains all available products ($|S| = 9$). It is important to note that in a scenario where equation (19) holds, indicating the absence of choice overload, each data point should exhibit a higher y-value than the red triangle. However, this condition is met for only 8 out of the 41 data points in the figure, suggesting a violation of equation (19) within the Beer category.

The presence of choice overload in the data has a significant impact on the estimation outcomes of both the ranking-based model and the decision forest model, leading to divergent assortment decisions. To begin, we define $\bar{P}_{\mathcal{C}}(0 \mid m)$ as the average no-purchase probability when presented with an assortment of size m; formally, $\bar{P}_{\mathcal{C}}(0 \mid m) = \sum_{S \subseteq \mathcal{N} : |S| = m} P_{\mathcal{C}}(0 \mid S) \big/ |\{S \subseteq \mathcal{N} : |S| = m\}|$, where $P_{\mathcal{C}}(0 \mid S)$ represents the no-purchase probability given assortment S under a choice model $\mathcal{C}$. We consider two choice models, the decision forest model (DF) and the ranking-based model (RM), both estimated from data. Figure 3 (right) presents the average no-purchase probability curves, $\bar{P}_{\mathcal{C}}(0 \mid m)$, for these two models, with the error bars representing the standard deviation. Recall that the ranking-based model adheres to the regularity property and strictly follows equation (19). Consequently, the resulting no-purchase probability curve (depicted in blue) does not exhibit the choice overload effect; it consistently decreases as we expand the assortment size. In contrast, the decision forest model is not bound by the regularity property and has the capacity to learn the choice overload effect from the data. The resulting no-purchase probability curve (shown in red) indicates that customers are indeed more inclined to make a purchase as the assortment initially expands. However, after reaching an assortment size of $|S| = 6$, the choice overload effect becomes evident. Customers become less motivated, and we observe an increase in the no-purchase probability when $|S| \geq 7$.

The extent to which the decision forest model and the ranking-based model capture the phenomenon of choice overload from the data significantly influences their downstream assortment decisions. It is important to note that within the dataset, the marginal revenues ρ_i of the products exhibit low variation, ranging between 8.14 and 10.83 USD (keeping in mind that beer is often sold in packs of six). Since prices are quite uniform and well above zero, the optimal strategy is *not* about making customers choose a specific product but rather about making them buy *any* product from the offered assortment, i.e., reducing the no-purchase probability.

For the ranking-based model, the optimal assortment S^{RM} includes all products, resulting in $|S^{\text{RM}}| = 9$, since this is the only way it can minimize the no-purchase probability. In contrast, the

decision forest model, having captured the choice overload effect from the data, takes a different approach. It examines assortments of size six and identifies one that decreases the no-purchase probability from $P_{\mathrm{DF}}(0 \mid S^{\mathrm{RM}}) = 33.1\%$ to $P_{\mathrm{DF}}(0 \mid S^{\mathrm{DF}}) = 8.6\%$ (see the black triangle in Figure 3 (right)). This strategic choice results in a remarkable 32% improvement in expected revenue, measured according to the decision forest model. This highlights the advantage of the assortment planning approach proposed in this paper: by leveraging the decision forest model's ability to capture consumer choice patterns, businesses can tailor their product offerings to effectively capitalize on consumers' departures from strictly rational purchasing behavior.

7. Conclusions

We developed a mixed-integer optimization methodology for solving the assortment optimization problem under the decision forest model, which accounts for both rational and non-rational customer behaviors. We first established the inapproximability of the problem, then presented two formulations, linking them to the optimization models in the literature. To enhance scalability, we proposed a large-scale solution approach based on Benders decomposition. Through synthetic data experiments, we demonstrated the efficiency of our methods, and with real-world data, we showed how consumer behavioral anomalies can be learned and leveraged within our framework.

Acknowledgments

We sincerely thank the department editor Huseyin Topaloglu, the associate editor and two anonymous reviewers for their thoughtful comments that have helped to significantly improve the paper.

References

A. Aouad, V. Farias, R. Levi, and D. Segev. The approximability of assortment optimization under ranking preferences. *Operations Research*, 66(6):1661–1669, 2018.

A. Aouad, V. Farias, and R. Levi. Assortment optimization under consider-then-choose choice models. *Management Science*, 2020.

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

G. Berbeglia. The generalized stochastic preference choice model. *arXiv preprint arXiv:1803.04244*, 2018.

G. Berbeglia, A. Garassino, and G. Vulcano. A comparative empirical study of discrete choice models in retail operations. *Management Science*, 68(6):4005–4023, 2022.

D. Bertsimas and V. V. Mišić. Exact first-choice product line optimization. *Operations Research*, 67(3):651–670, 2019.

D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*. Dynamic Ideas, Belmont, MA, 1997.

D. Bertsimas and R. Weismantel. *Optimization over integers*, volume 13. Dynamic Ideas Belmont, 2005.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1): 65–98, 2017.

B. J. Bronnenberg and C. F. Mela. Market roll-out and retailer adoption for new brands. *Marketing Science*, 23(4):500–518, 2004.

B. J. Bronnenberg, M. W. Kruger, and C. F. Mela. Database paper—the IRI marketing data set. *Marketing science*, 27(4): 745–748, 2008.

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.

K. D. Chen and W. H. Hausman. Mathematical properties of the optimal product line selection problem using choice-based conjoint analysis. *Management Science*, 46(2):327–332, 2000.

Y.-C. Chen and V. V. Mišić. Decision forest: A nonparametric approach to modeling irrational choice. *Management Science*, 68(10):7065–7791, 2022.

A. Chernev, U. Böckenholt, and J. Goodman. Choice overload: A conceptual review and meta-analysis. *Journal of Consumer Psychology*, 25(2):333–358, 2015.

I. Contreras, J.-F. Cordeau, and G. Laporte. Benders decomposition for large-scale uncapacitated hub location. *Operations Research*, 59(6):1477–1490, 2011.

J.-F. Cordeau, F. Furini, and I. Ljubić. Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research*, 275(3):882–896, 2019.

A. Désir, V. Goyal, D. Segev, and C. Ye. Constrained assortment optimization under the markov chain–based choice model. *Management Science*, 66(2):698–721, 2020.

F. Echenique and K. Saito. General Luce model. *Economic Theory*, 68(4):811–826, 2019.

J. Feldman, A. Paul, and H. Topaloglu. Assortment optimization with small consideration sets. *Operations Research*, 67(5): 1283–1299, 2019.

J. B. Feldman and H. Topaloglu. Revenue management under the Markov chain choice model. *Operations Research*, 65(5): 1322–1342, 2017.

M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

A. Flores, G. Berbeglia, and P. Van Hentenryck. Assortment and price optimization under the two-stage luce model. *arXiv preprint arXiv:1706.08599*, 2017.

D. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

G. Gallego and H. Topaloglu. *Assortment Optimization*, pages 129–160. Springer New York, 2019.

G. Gallego, R. Ratliff, and S. Shebalov. A general attraction model and sales-based linear program for network revenue management under customer choice. *Operations Research*, 63(1):212–232, 2015.

M. R. Garey and D. S. Johnson. *Computers and intractability*. W. H. Freeman New York, 1979.

P. E. Green and A. M. Krieger. Conjoint analysis with product-positioning applications. *Handbooks in operations research and management science*, 5:467–515, 1993.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2024. URL `https://www.gurobi.com`.

V. Guruswami and L. Trevisan. The complexity of making unique choices: Approximating 1-in-k SAT. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 99–110. Springer, 2005.

A. J. Hoffman and J. B. Kruskal. *Integral boundary points of convex polyhedra*, pages 223–246. Number 38 in Annals of Mathematics. Princeton University Press, Princeton, NJ, 1956.

J. Huchette and J. P. Vielma. A combinatorial approach for small and strong formulations of disjunctive constraints. *Mathematics of Operations Research*, 44(3):793–820, 2019.

IBM. *IBM ILOG CPLEX Optimization Studio*. IBM, Armonk, NY, 2024. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

S. S. Iyengar and M. R. Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79(6):995, 2000.

S. Jagabathula and P. Rusmevichientong. The limit of rationality in choice modeling: Formulation, computation, and implications. *Management Science*, 65(5):2196–2215, 2019.

T. Kamishima. Nantonac collaborative filtering: recommendation based on order responses. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 583–588, 2003.

R. Kivetz, O. Netzer, and V. Srinivasan. Extending compromise effect models to complex buying situations and other context effects. *Journal of Marketing Research*, 41(3):262–268, 2004.

X. Long, J. Sun, H. Dai, D. Zhang, J. Zhang, Y. Chen, H. Hu, and B. Zhao. The choice overload effect in online retailing platforms. *Available at SSRN 3890056*, 2023.

M. Lubin and I. Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

R. D. McBride and F. S. Zufryden. An integer programming approach to the optimal product line selection problem. *Marketing Science*, 7(2):126–140, 1988.

I. Méndez-Díaz, J. J. Miranda-Bront, G. Vulcano, and P. Zabala. A branch-and-cut algorithm for the latent-class logit assortment problem. *Discrete Applied Mathematics*, 164:246–263, 2014.

V. V. Mišić. Optimization of tree ensembles. *Operations Research*, 68(5):1605–1624, 2020.

V. R. Nijs, S. Srinivasan, and K. Pauwels. Retail-price drivers and retailer profits. *Marketing Science*, 26(4):473–487, 2007.

R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

J. Rieskamp, J. R. Busemeyer, and B. A. Mellers. Extending the bounds of rationality: Evidence and theories of preferential choice. *Journal of Economic Literature*, 44(3):631–661, 2006.

R. P. Rooderkerk, H. J. Van Heerde, and T. H. A. Bijmolt. Incorporating context effects into a choice model. *Journal of Marketing Research*, 48(4):767–780, 2011.

C. Schön. On the optimal product line selection problem with price discrimination. *Management Science*, 56(5):896–902, 2010.

M. Sumida, G. Gallego, P. Rusmevichientong, H. Topaloglu, and J. Davis. Revenue-utility tradeoff in assortment optimization under the multinomial logit model with totally unimodular constraints. *Management Science*, 2020.

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

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, 2014.

G. van Ryzin and G. Vulcano. An expectation-maximization method to estimate a rank-based choice model of demand. *Operations Research*, 65(2):396–407, 2017.

J. P. Vielma and G. L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming*, 128(1-2):49–72, 2011.

J. P. Vielma, S. Ahmed, and G. Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations Research*, 58(2):303–315, 2010.

Appendix A: Proofs

A.1. Proof of Theorem 1

In what follows, we will transform the Max 1-in-EkSAT problem (Guruswami and Trevisan 2005) of n variables into an instance of assortment problem AODF$(k+1)$ over $n+1$ products. Notice that according to Theorem 9 of Guruswami and Trevisan (2005), unless P = NP, for any $\epsilon > 0$ and constant $k \geq 7$, there is no polynomial time $(2^{k-2\lceil\sqrt{k}\rceil}/(2ek) - \epsilon)$ approximation algorithm for the Max 1-in-EkSAT problem.

We first describe the Max 1-in-EkSAT problem, which is a variant of the SAT problem. In the Max 1-EkSAT problem, one has n boolean variables $x_1, \ldots, x_n$ and a collection of clauses $C^1, C^2, \ldots, C^M$. Each clause consists of exactly k literals and each literal is either one of the binary variables or its negation. A clause is satisfied if *exactly* one literal is true. For example, consider the clause $C^1 = \{x_1, \neg x_2, x_3\}$. The clause is satisfied if $(x_1, x_2, x_3) = (\texttt{true}, \texttt{true}, \texttt{false})$. However, C^1 is *not* satisfied if $(x_1, x_2, x_3) = (\texttt{true}, \texttt{false}, \texttt{false})$, since in this case, the two literals x_1 and $\neg x_2$ are true. In the Max 1-in-EkSAT problem, one finds the assignment of boolean variables $x_1, \ldots, x_n$ that maximizes the number of satisfied clauses.

Given an instance of the Max 1-in-EkSAT problem with n variables, we show how the problem can be transformed into an instance of the assortment problem AODF$(k+1)$ of $n+1$ products. First, we construct an instance of the latter problem such that each of its first n products corresponds to a binary variable in the Max 1-in-EkSAT problem. Specifically, we let the ith product in AODF$(k+1)$ correspond to variable x_i in the Max 1-in-EkSAT problem, for $i = 1, 2, \ldots, n$. The $(n+1)$-st product in AODF$(k+1)$ is designed to ensure that the revenue of the assortment can correspond to the number of satisfied clauses. Furthermore, the action of assigning `true` to variable x_i in the Max 1-in-EkSAT problem corresponds to the action of including product i in the assortment for AODF$(k+1)$, implying that we will traverse to the left if we meet a split node of product i in a decision tree of AODF$(k+1)$. We assume that the marginal revenue of the products follows $\rho_1 = \ldots = \rho_n = 0$ and $\rho_{n+1} = M$, where M is the number of clauses in the Max 1-in-EkSAT problem.

For each clause C^m of the Max 1-in-EkSAT problem, we introduce a decision tree t_m with depth $k+1$ and $O(k^2)$ leaf nodes that mimics the structure of C^m. We set the probability of tree t_m to be $1/M$, so that $\lambda_{t_m} = 1/M$.

We then construct each tree as follows. Set the root node of t_m to be a split node involving product $n+1$. The right child of the root node is set to be a leaf node with the no-purchase option 0. We then construct a subtree by invoking the function `GrowTree` on C_m (formalized as Algorithm 4 below), and root this subtree, `GrowTree`(C_m), at the left child of the root node of t_m; see Figure 4a. The function call `GrowTree`(C^m) constructs a subtree according to the clause C^m and the resulting tree t_m will have the following property: an assortment will lead to a purchase decision of $n+1$ in tree t_m if and only if we make exactly one of the literals true in C^m.

Note that Algorithm 4 will iteratively call a subroutine, `GrowVine` (formalized as Algorithm 3), to build a subtree that corresponds to subclauses of clause C^m. To better explain the two algorithms, we introduce the following notation. We use C^m_j to denote the jth literal of clause C^m and $C^m_{j:k}$ to denote the subclause that consists of literals $C^m_j, C^m_{j+1}, \ldots, C^m_k$. We further write $V(C^m_j)$ as the variable that appears in the literal C^m_j.

For example, if $C^m = \{x_1, \neg x_2, x_3, \neg x_4\}$, then $C^m_2 = \neg x_2$, $C^m_{2:4} = \{\neg x_2, x_3, \neg x_4\}$, and $V(C^m_2) = 2$. Note that $C^m_{j:k}$ is simply an empty set if $j > k$. In the algorithms, we will introduce dummy variables x_0 and x_{n+1} for convenience; however, they do not contribute to any real decision in the Max 1-in-EkSAT problem. With a slight abuse of notation, we use $v(s)$ to denote the product associated with a node s in a decision tree.

With this notation in hand, we now explain the mechanics of the function `GrowVine`. The function `GrowVine` constructs a substructure, called a *vine*, in a way that it only grows the split nodes in the *opposite* directions of the literals. We design the vine to have the following property: one makes the purchase decision of $n+1$ in the vine if and only if one makes every literal in the subclause *false*. Figure 4b shows an example of a vine constructed according to subclause $\{\neg x_2, x_3, \neg x_4\}$. The vine first places 2 at its root, which the variable for the first literal $\neg x_2$ of the subclause. Since x_2 appears in negated form in the subclause, the vine branches to the left, which is the "positive" direction of the tree, to grow a split node for the next literal. The right child node of 2 is set to a leaf node that is assigned to the no-purchase option. Now, we further proceed with the child split node of 2, which will be associated with the next literal, x_3. We place 3 at the node. Since x_3 appears in the non-negated form in the subclause, the vine branches to the right, which the "negative" direction of the tree, to grow a split node for the next literal. The left child node of 3 is set to a leaf node that is assigned to the no-purchase option. The process repeats until we use all the literals in the subclause and place the purchase decision $n+1$ for the leaf which is reached if all the literals are false. Algorithm 3 summarizes the detailed implementation in a pseudocode.

We now turn our attention back to `GrowTree`. Recall that Algorithm 4 grows a subtree, `GrowTree`(C^m), according to the clause C^m, and that this subtree should be such that one reaches a leaf with the (profitable) purchase decision of $n+1$ if and only if one makes exactly one of the literals in the clause C^m true. Figure 4 shows an example of subtree `GrowTree`(C^m) constructed according to the clause $C^m = \{x_1, \neg x_2, x_3, \neg x_4\}$. The algorithm first places 1 at the root split node, which is the variable for the first literal x_1. The tree now grows in both directions. The left branch from the node with 1 corresponds to the condition that literal x_1 is already true, and so we construct a subtree rooted at the left child such that the profitable purchase node $n+1$ is reached if and only if the remaining literals $\neg x_2, x_3$, and $\neg x_4$ are all made false. To this end, we use Algorithm 3, which takes the remaining literals as an input, to produce a vine `GrowVine`$(\{\neg x_2, x_3, \neg x_4\})$ as the left descendant of 1. This vine is exactly the one in Figure 4b. Now, we proceed to the right child of the split node with 1. We set this node to a split node and place 2 on this child node, which is the the variable for the second literal $\neg x_2$. We again grow the tree both to the left and right of the split node with 2. For the right branch, which corresponds to the condition that in clause C_m the first literal x_1 is already false and the second literal $\neg x_2$ is already true, we call Algorithm 3 and place vine `GrowVine`$(\{x_3, \neg x_4\})$. We then move on to the left child node of 2 and repeat the process until we use all the literals. Figure 5a shows how we repeatedly call Algorithm 3 in Algorithm 4 and Figure 5b shows the complete subtree. Figure 6 shows the structure of the final tree t_m.

We note that the construction of the tree t_m takes $O(k^2)$ runtimes, as we call out Algorithm 3 at most k times in Algorithm 4 and each Algorithm 3 uses $O(k)$ steps. One can easily show that the resulting tree t_m has $2 + k(k+1)/2$ leaf nodes. Therefore, whenever $k \geq 2$, the tree t_m has at most k^2 leaf nodes.

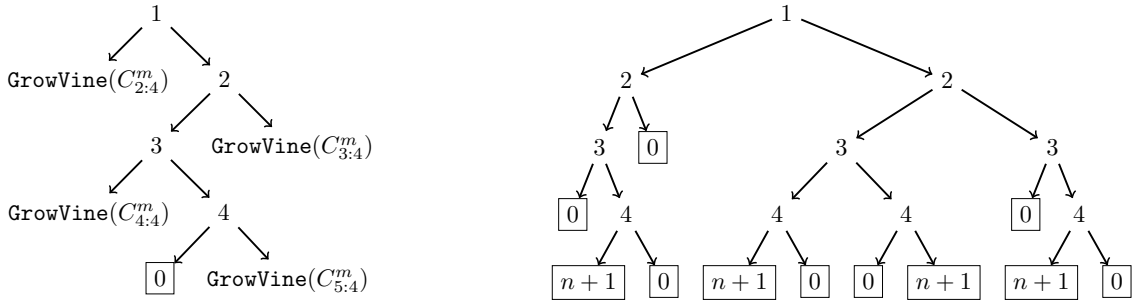
(a) Tree constructed for clause C^m. (b) GrowVine$(\{\neg x_2, x_3, \neg x_4\})$

Figure 4 **The construction of the tree t_m and an example of the GrowVine procedure (Algorithm 3).**

Algorithm 3 GrowVine

Require: subclause C^{sub}

$C^{\text{sub}} \leftarrow C^{\text{sub}} \cup \{x_{n+1}\}$, added from behind.
$s \leftarrow$ root node
$v(s) \leftarrow V\left(C_1^{\text{sub}}\right)$.
for $q = 1, \ldots, |C^{\text{sub}}| - 1$ **do**
  $(s_{\text{L}}, s_{\text{R}}) \leftarrow$ left and right child nodes of s.
  if C_q^{sub} is a negation **then**
    $\left(v(s_{\text{L}}), v(s_{\text{R}})\right) \leftarrow \left(V\left(C_{q+1}^{\text{sub}}\right), 0\right)$.
    $s \leftarrow s_{\text{L}}$
  else
    $\left(v(s_{\text{L}}), v(s_{\text{R}})\right) \leftarrow \left(0, V\left(C_{q+1}^{\text{sub}}\right)\right)$.
    $s \leftarrow s_{\text{R}}$
Set all nodes with 0 and $n+1$ as leaf nodes



(a) The Growing of GrowTree(C^m) (b) The resulting subtree GrowTree(C^m)

Figure 5 **Example of the GrowTree procedure (Algorithm 4) for the clause $C^m = \{x_1, \neg x_2, x_3, \neg x_4\}$.**

Finally, to complete the proof, it suffices to prove the following two claims. First, for any assignment $\mathbf{x}$ to the Max 1-in-EkSAT problem with z^{SAT} clauses satisfied, we can construct an assortment $S_{\mathbf{x}}$ which has an expected revenue z^{SAT} in the constructed instance of the assortment problem AODF$(k+1)$. Second, for any assortment S of expected revenue $z^{\text{AODF}} > 0$ in the constructed instance of the assortment problem AODF$(k+1)$, we can construct an assignment $\mathbf{x}_S$ of variables to the the Max 1-in-EkSAT problem with z^{AODF} clauses satisfied. To prove the first claim, we simply construct an assortment $S_{\mathbf{x}}$ that includes product $n+1$ and product i if $x_i = \texttt{true}$ in the assignment $\mathbf{x}$, for all $i = 1, \ldots, n$. Since tree t_m contributes 1 to the

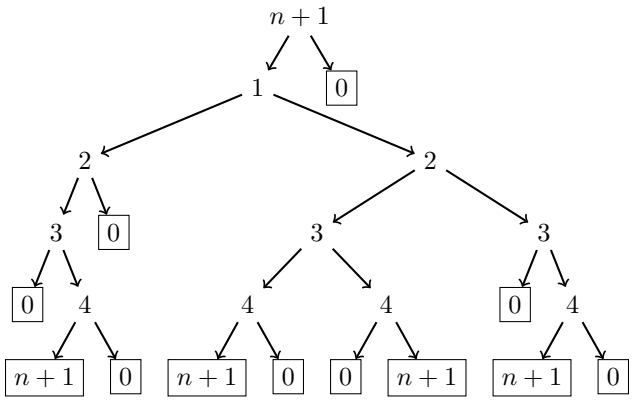


Figure 6 **Final tree t_m corresponding to clause $C^m = \{x_1, \neg x_2, x_3, \neg x_4\}$.**

Algorithm 4 GrowTree

Require: Clause C and $k = |C|$
  $C \leftarrow C \cup \{x_0\}$, added from behind.
  $s \leftarrow$ root node
  $v(s) \leftarrow V(C_1)$.
  for $q = 1, \ldots, k$ **do**
    $(s_{\mathrm{L}}, s_{\mathrm{R}}) \leftarrow$ left and right child nodes of s.
    if C_q is a negation **then**
      Replace node s_{R} by subtree `GrowVine`$(C_{q+1:k})$ via Algorithm 3.
      $v(s_{\mathrm{L}}) \leftarrow V(C_{q+1})$
      $s \leftarrow s_{\mathrm{L}}$
    else
      Replace node s_{L} by subtree `GrowVine`$(C_{q+1:k})$ via Algorithm 3.
      $v(s_{\mathrm{R}}) \leftarrow V(C_{q+1})$
      $s \leftarrow s_{\mathrm{R}}$
  Set all nodes with 0 and $n+1$ as leaf nodes

total expected revenue if and only if the claim C_m is satisfied by $\mathbf{x}$, we know that the expected revenue of the assortment is z^{SAT}.

To prove the second claim, we first note that assortment S must have included product $n+1$, otherwise its expected revenue would be zero. We construct the assignment $\mathbf{x}_S$ of variables to the Max 1-in-EkSAT problem by setting $x_i = \texttt{true}$ if product i is offered in S. Furthermore, since each tree contributes either expected revenue 1 or 0 to the total expected revenue, there must be z^{AODF} trees that contributes to the revenue. For each such tree, the corresponding clause must be satisfied according to the tree construction. Therefore, the assignment $\mathbf{x}$ satisfies z^{AODF} clauses. □

A.2. Proof of Proposition 1

Let Z^* be the optimal objective value of the assortment problem AODF(d) and let $r^t_{\max}$ be the revenue of most expensive product in the leaf nodes of tree t. Call this leaf node $\ell^t_{\max}$; if there are multiple leaf nodes with the maximal revenue in tree t, we choose the leftmost one. Then we naturally have $Z^* \leq \sum_t \lambda_t \cdot r^t_{\max}$.

We construct a random assortment S_X through the following procedure. First, we independently draw values for $X_1, X_2, \ldots, X_n$, where X_i is an IID Bernoulli random variable with success rate 1/2. We include

product i in the assortment S_X if $X_i = 1$. Notice that given a decision tree of depth at most d, the probability of node $\ell^t_{\max}$ being chosen when S_X is offered is

$$\prod_{s \in \mathbf{LS}(\ell^t_{\max})} P\left(X_{v(s)} = 1\right) \cdot \prod_{s \in \mathbf{RS}(\ell^t_{\max})} P\left(X_{v(s)} = 0\right) = \prod_{s \in \mathbf{LS}(\ell^t_{\max})} \frac{1}{2} \cdot \prod_{s \in \mathbf{RS}(\ell^t_{\max})} \frac{1}{2} \geq \frac{1}{2^d},$$

where $\mathbf{LS}(\ell)$ and $\mathbf{RS}(\ell)$ are the sets of split nodes that consider leaf node ℓ as its left descendant and right descendant, respectively. The last inequality holds since the leaf node $\ell^t_{\max}$ would not have more than d ancestors given that the tree t is of depth at most d. Therefore, the expected revenue of assortment S_X is

$$E_X\left[\sum_t \lambda_t \cdot R_t(S_X)\right] = \sum_t \lambda_t \cdot E_X\left[R_t(S_X)\right] \geq \sum_t \lambda_t \cdot P_{\ell^t_{\max}} \cdot r^t_{\max} \geq \sum_t \lambda_t \cdot \frac{r^t_{\max}}{2^d} \geq \frac{Z^*}{2^d}$$

where $R_t(S_X)$ is the revenue of tree t under assortment S_X and $P_{\ell^t_{\max}}$ is the probability that the tree t chooses the leaf node $\ell^t_{\max}$ under assortment S_X. By applying the derandomization method, we can retrieve a deterministic assortment that approximates the optimal objective value Z^* within factor 2^d. □

A.3. Proof of Proposition 2

Proof of the sufficient condition: When $|F| = 1$, we can write the feasible region $\mathcal{F}_{\text{SplitMIO}}$ of the relaxation of SplitMIO in standard form as follows:

$$\sum_{\ell \in \mathbf{left}(s)} y_\ell - x_{v(s)} + w_{\mathbf{left},s} = 0, \quad \forall\ s \in \mathbf{splits}, \tag{20}$$

$$\sum_{\ell \in \mathbf{right}(s)} y_\ell + x_{v(s)} + w_{\mathbf{right},s} = 1, \quad \forall\ s \in \mathbf{splits}, \tag{21}$$

$$\sum_{\ell \in \mathbf{leaves}} y_\ell = 1, \quad \forall\ \ell \in \mathbf{leaves}, \tag{22}$$

$$x_i + z_i = 1, \quad \forall\ i \in [n]. \tag{23}$$

$$y_\ell \geq 0, \quad \forall\ \ell \in \mathbf{leaves}, \tag{24}$$

$$w_{\mathbf{left},s}, w_{\mathbf{right},s} \geq 0, \quad \forall\ s \in \mathbf{splits}, \tag{25}$$

$$x_i, z_i \geq 0, \quad \forall\ i \in [n], \tag{26}$$

where $w_{\mathbf{left},s}$, $w_{\mathbf{right},s}$ and z_i are slack variables. We let $\mathbf{w}$ be the vector of $w_{\mathbf{left},s}$ and $w_{\mathbf{right}}$ slack variables and $\mathbf{z}$ be the vector of z_i slack variables. When we concatenate all of the decision variables in a vector $(\mathbf{y}, \mathbf{x}, \mathbf{w}, \mathbf{z})$, the constraint coefficient matrix can be written in block form using the matrices $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ as

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{27}$$

where $\mathbf{I}$ is used to denote an appropriately-sized identity matrix and $\mathbf{0}$ is an appropriately sized matrix with all entries equal to zero. The first row in the block form corresponds to constraints (20) - (22), and the second row in the block form corresponds to (23). We assume that the columns in $\mathbf{B}$ are ordered so as to follow the order of the leaves in the tree topology, i.e., the left-most leaf in the tree topology appears as the first column, followed by the second left-most leaf, and so on, until the right-most leaf which would be the final column in $\mathbf{B}$.

Note that the columns of the matrix $\mathbf{D}$ are simply columns of an appropriately sized identity matrix. Since appending columns or rows from the identity matrix to a matrix preserves the total unimodularity of that

matrix, the above matrix is totally unimodular if and only if the matrix [**B C**] is totally unimodular. Thus, we focus on establishing that [**B C**] is totally unimodular.

Consider now a square submatrix $\mathbf{A}'$ of [**B C**]. To show that [**B C**] is totally unimodular, we need to verify that $\det \mathbf{A}' \in \{+1, -1, 0\}$. There are two cases to consider:

Case 1. If $\mathbf{A}'$ is a submatrix of only **B**, i.e., $\mathbf{A}' = [\mathbf{B}']$ for some square submatrix $\mathbf{B}'$ of **B**, then the determinant of $\mathbf{A}'$ must be +1, -1, or 0. This is because the matrix **B** satisfies the consecutive ones property – each row is of the form $[0, \dots, 0, 1, \dots, 1, 0, \dots, 0]$ – which is guaranteed because the columns of **B** are assumed to follow the order of the leaves in the tree topology. Since **B** satisfies the consecutive ones property and is an interval matrix, it is totally unimodular, and any square submatrix has determinant +1, -1, or 0 (Fulkerson and Gross 1965; see also Theorem 3.3(c) in Bertsimas and Weismantel 2005).

Case 2. If $\mathbf{A}'$ a submatrix of both **B** and **C**, i.e., $\mathbf{A}' = [\mathbf{B}' \ \mathbf{C}']$ for appropriate submatrices $\mathbf{B}'$ of **B** and $\mathbf{C}'$ of **C**, write $\mathbf{C}'$ as $\mathbf{C}' = [\mathbf{C}'_1 \ \dots \mathbf{C}'_k]$, where each $\mathbf{C}'_j$ is a column of the submatrix $\mathbf{C}'$. For each column $\mathbf{C}'_j$, there are four possibilities: either the column contains all zeros; the column contains one +1, and all other entries are zero; the column contains one -1, and all other entries are zero; or the column contains one +1, one -1, and all other entries are zero. Note that it is not possible for a +1 to appear more than once, or a -1 to appear more than once, because of the hypothesis that each product appears at most once in the splits of the tree.

For each column that falls into the first three categories, do nothing. For each column that falls into the fourth category, perform an elementary row operation as follows: take the row in which the +1 occurs, and add it to the row in which the -1 occurs. These two rows have the form

$$[\mathbf{f}_{\mathbf{left}(s)} \ \ 0 \ \dots \ 0 \ -1 \ 0 \ \dots 0] \qquad \text{(row with -1)} \tag{28}$$

$$[\mathbf{f}_{\mathbf{right}(s)} \ 0 \ \dots \ 0 \ +1 \ 0 \ \dots 0] \qquad \text{(row with +1)} \tag{29}$$

for some split s, where $\mathbf{f}_{\mathcal{U}}$ is the incidence vector of the leaf set $\mathcal{U}$, i.e., a 0-1 vector where a 1 occurs for each column in the submatrix **B** if the corresponding leaf is in the set $\mathcal{U}$, and a 0 occurs otherwise. Observe that when we add them, the +1 and -1 cancel out, and the row in which the -1 occurs becomes

$$[\mathbf{f}_{\mathbf{left}(s)} + \mathbf{f}_{\mathbf{right}(s)} \ 0 \ \dots \ 0] \tag{30}$$

which is equivalent to

$$[\ \mathbf{f}_{\mathbf{left}(s) \cup \mathbf{right}(s)} \ \ 0 \ \dots \ 0], \tag{31}$$

because $\mathbf{left}(s)$ and $\mathbf{right}(s)$ do not intersect. Note also that for any split s, the vector $\mathbf{f}_{\mathbf{left}(s) \cup \mathbf{right}(s)}$ satisfies the consecutive ones property, again by the assumption that the columns of **B** are ordered in the same way as the leaves in the tree topology.

Thus, by performing this operation – adding the row with a -1 to the row with the +1 – for each column $\mathbf{C}'_j$ with one +1 and one -1, we obtain a new matrix, $[\tilde{\mathbf{B}} \ \tilde{\mathbf{C}}]$, where each column of $\tilde{\mathbf{C}}$ is either a vector of

zeros, a vector of zeros with one +1 (i.e., a column of an appropriately sized identity matrix), or a vector of zeros with one -1 (i.e., the negative of a column of an appropriately sized identity matrix). Additionally, each row of $\tilde{\mathbf{B}}$ still satisfies the consecutive ones property. This follows because the modified row satisfies the consecutive ones property, as explained in the previous paragraph, and all other unmodified rows also satisfy it, as discussed for Case 1. Note also that because each product appears at most once in each split, any row that is modified by the prescribed elementary row operation is only modified once, and cannot be modified multiple times. Thus, the square matrix $[\tilde{\mathbf{B}}\ \tilde{\mathbf{C}}]$ is totally unimodular, and its determinant therefore must be +1, -1 or 0. Since this matrix was obtained by elementary row operations from the submatrix $[\mathbf{B}'\ \mathbf{C}']$, it follows that this original submatrix has the same determinant as $[\tilde{\mathbf{B}}\ \tilde{\mathbf{C}}]$, and therefore its determinant is in +1, -1, or 0.

Thus, since $\det \mathbf{A}' \in \{+1, -1, 0\}$, it follows that $[\mathbf{B}\ \mathbf{C}]$ is totally unimodular, and that the overall constraint matrix $\mathbf{A}$ is totally unimodular. By standard integer programming results (Hoffman and Kruskal 1956; see also Theorem 3.1(a) in Bertsimas and Weismantel 2005), it follows that all extreme points of the polyhedron described by (20) - (26) are integral. Lastly, since the polyhedron (20) - (26) is isomorphic to the polyhedron described by constraints (3b) - (3f) when $|F| = 1$, each extreme point of (20) - (26) is an extreme point of $\mathcal{F}_{\text{SPLITMIO}}$ (see Exercise 2.5(b) in Bertsimas and Tsitsiklis 1997), and thus each extreme point of $\mathcal{F}_{\text{SPLITMIO}}$ is integral.

Proof of the necessary condition: To show this direction, we will prove the contrapositive. Thus, suppose that there exists a product i^* such that for two different splits, s_1, s_2, it is the case that $v(s_1) = i^*$ and $v(s_2) = i^*$ (note that for convenience, we drop t whenever it appears). We now need to show the existence of a non-integral extreme point of $\mathcal{F}_{\text{SPLITMIO}}$.

Let $\ell_1 \in \mathbf{left}(s_1)$ be any leaf left of s_1, and $\ell_2 \in \mathbf{left}(s_2)$ be any leaf left of s_2. Let $\nu_1^1, \ldots, \nu_{k_1}^1$ be the set of split nodes traversed on the path from the root node to leaf ℓ_1, where ν_1^1 denotes the root node. Similarly, let $\nu_1^2, \ldots, \nu_{k_2}^2$ be the set of split nodes traversed on the path from the root node to leaf ℓ_2, where again ν_1^2 denotes the root node. Let k^* be defined as

$$k^* = \max\{1 \leq k \leq \min\{k_1, k_2\} \mid \nu_k^1 = \nu_k^2\}. \tag{32}$$

Observe that k^* denotes the position of the node where the path from the root node to ℓ_1 and the path from the root node to ℓ_2 diverge. Let $s^* = \nu_{k^*}^1 = \nu_{k^*}^2$ denote this split node. Without loss of generality, let us assume that $\ell_1 \in \mathbf{left}(s^*)$ and $\ell_2 \in \mathbf{right}(s^*)$. See an illustration in Figure 7.

Next, we will define subsets of split nodes. We define the sets $\mathcal{S}^{0,\mathbf{left}}, \mathcal{S}^{0,\mathbf{right}}, \mathcal{S}^{1,\mathbf{left}}, \mathcal{S}^{1,\mathbf{right}}, \mathcal{S}^{2,\mathbf{left}}, \mathcal{S}^{2,\mathbf{right}}$ as

$$\mathcal{S}^{0,\mathbf{left}} = \mathbf{LS}(\ell_1) \cap \{\nu_1^1, \ldots, \nu_{k^*-1}^1\}, \tag{33}$$

$$\mathcal{S}^{0,\mathbf{right}} = \mathbf{RS}(\ell_1) \cap \{\nu_1^1, \ldots, \nu_{k^*-1}^1\}, \tag{34}$$

$$\mathcal{S}^{1,\mathbf{left}} = \mathbf{LS}(\ell_1) \cap \{\nu_{k^*}^1, \nu_{k^*+1}^1, \ldots, \nu_{k_1}^1\}, \tag{35}$$

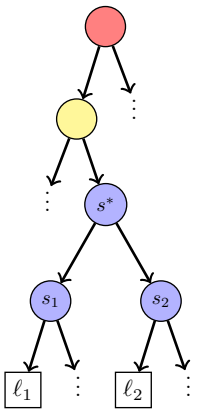


Figure 7 **An illustration supporting the proof of the necessary condition in Proposition 2. Nodes are labeled according to the definitions in the proof. Unspecified parts of the tree are represented by vertical dots. Split nodes are colored blue, red, and yellow, corresponding to x_i values of 0.5, 1, and 0, respectively, as defined in equations (42) - (44). Additionally, $y_{\ell_1} = y_{\ell_2} = 0.5$, while the y_ℓ values for all other leaf nodes are 0.**

$$\mathcal{S}^{1,\textbf{right}} = \textbf{RS}(\ell_1) \cap \{\nu^1_{k^*}, \nu^1_{k^*+1}, \ldots, \nu^1_{k_1}\}, \tag{36}$$

$$\mathcal{S}^{2,\textbf{left}} = \textbf{LS}(\ell_2) \cap \{\nu^2_{k^*}, \nu^2_{k^*+1}, \ldots, \nu^2_{k_2}\}, \tag{37}$$

$$\mathcal{S}^{2,\textbf{right}} = \textbf{RS}(\ell_2) \cap \{\nu^2_{k^*}, \nu^2_{k^*+1}, \ldots, \nu^2_{k_2}\}. \tag{38}$$

In words, $\mathcal{S}^{0,\textbf{left}}$ and $\mathcal{S}^{0,\textbf{right}}$ are the sets of splits that we follow to the left and to the right, respectively, to reach s^*. From s^*, $\mathcal{S}^{1,\textbf{left}}$ and $\mathcal{S}^{1,\textbf{right}}$ are the sets of splits that we follow to the left and to the right, respectively, to reach ℓ_1. Finally, from s^*, $\mathcal{S}^{2,\textbf{left}}$ and $\mathcal{S}^{2,\textbf{right}}$ are the sets of splits that we follow to the left and to the right, respectively, to reach ℓ_2.

Next, we define sets of product indices. We let $\mathcal{I} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \textbf{splits}\}$ be the overall set of products that appear in the splits of the tree. We define additional sets as follows:

$$\mathcal{I}^{0,\textbf{left}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{0,\textbf{left}}\}$$
$$\mathcal{I}^{0,\textbf{right}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{0,\textbf{right}}\}$$
$$\mathcal{I}^{1,\textbf{left}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{1,\textbf{left}}\}$$
$$\mathcal{I}^{1,\textbf{right}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{1,\textbf{right}}\}$$
$$\mathcal{I}^{2,\textbf{left}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{2,\textbf{left}}\}$$
$$\mathcal{I}^{2,\textbf{right}} = \{i \in \mathcal{N} \mid v(s) = i \text{ for some } s \in \mathcal{S}^{2,\textbf{right}}\}$$

We now construct a fractional solution, as follows:

$$y_\ell = 0, \quad \forall \ell \in \textbf{leaves} \setminus \{\ell_1, \ell_2\}, \tag{39}$$

$$y_{\ell_1} = 0.5, \tag{40}$$

$$y_{\ell_2} = 0.5, \tag{41}$$

$$x_i = 1, \quad \forall i \in \mathcal{I}^{0,\textbf{left}}, \tag{42}$$

$$x_i = 0, \quad \forall i \in \mathcal{I}^{0,\textbf{right}}, \tag{43}$$

$$x_i = 0.5, \quad \forall i \in \mathcal{I}^{1,\textbf{left}} \cup \mathcal{I}^{2,\textbf{left}} \cup \mathcal{I}^{1,\textbf{right}} \cup \mathcal{I}^{2,\textbf{right}}, \tag{44}$$

$$x_i = 0, \quad \forall i \in \mathcal{N} \setminus (\mathcal{I}^{0,\textbf{left}} \cup \mathcal{I}^{0,\textbf{right}} \mathcal{I}^{1,\textbf{left}} \cup \mathcal{I}^{2,\textbf{left}} \cup \mathcal{I}^{1,\textbf{right}} \cup \mathcal{I}^{2,\textbf{right}}) \tag{45}$$

This construction of the solution $(\mathbf{x}, \mathbf{y})$ is well-defined because $(\mathcal{I}^{1,\textbf{left}} \cup \mathcal{I}^{2,\textbf{left}} \cup \mathcal{I}^{1,\textbf{right}} \cup \mathcal{I}^{2,\textbf{right}})$, $\mathcal{I}^{0,\textbf{left}}$, and $\mathcal{I}^{1,\textbf{right}}$ are mutually disjoint by their definitions and in accordance with Assumption 1. We illustrate this constructed solution in Figure 7.

It is also not difficult to verify that this solution is a feasible solution of $\mathcal{F}_{\text{SplitMIO}}$. Additionally, one can verify that this is the unique solution of the following system of active constraints:

$$y_\ell = 0, \quad \forall\ \ell \in \textbf{leaves} \setminus \{\ell_1, \ell_2\} \tag{46a}$$

$$\sum_{\ell \in \textbf{left}(s_1)} y_\ell = x_{v(s_1)}, \tag{46b}$$

$$\sum_{\ell \in \textbf{left}(s_2)} y_\ell = x_{v(s_2)}, \tag{46c}$$

$$\sum_{\ell \in \textbf{leaves}} y_\ell = 1, \tag{46d}$$

$$\sum_{\ell \in \textbf{left}(s)} y_\ell = x_{v(s)}, \quad \forall s \in \mathcal{S}^{0,\textbf{left}}, \tag{46e}$$

$$\sum_{\ell \in \textbf{right}(s)} y_\ell = 1 - x_{v(s)}, \quad \forall s \in \mathcal{S}^{0,\textbf{right}} \tag{46f}$$

$$\sum_{\ell \in \textbf{left}(s)} y_\ell = x_{v(s)}, \quad \forall s \in \mathcal{S}^{1,\textbf{left}} \cup \mathcal{S}^{2,\textbf{left}}, \tag{46g}$$

$$\sum_{\ell \in \textbf{right}(s)} y_\ell = 1 - x_{v(s)}, \quad \forall s \in \mathcal{S}^{1,\textbf{right}} \cup \mathcal{S}^{2,\textbf{right}}, \tag{46h}$$

$$x_i = 0, \quad \forall i \in \mathcal{N} \setminus (\mathcal{I}^{0,\textbf{left}} \cup \mathcal{I}^{0,\textbf{right}} \mathcal{I}^{1,\textbf{left}} \cup \mathcal{I}^{2,\textbf{left}} \cup \mathcal{I}^{1,\textbf{right}} \cup \mathcal{I}^{2,\textbf{right}}) \tag{46i}$$

Since the proposed $(\mathbf{x}, \mathbf{y})$ is a feasible solution of $\mathcal{F}_{\text{SplitMIO}}$ and is uniquely determined by the above set of active constraints, it follows that there are $n + |\textbf{leaves}|$ linearly independent active constraints. This means that $(\mathbf{x}, \mathbf{y})$ is a basic feasible solution (and hence an extreme point) of $\mathcal{F}_{\text{SplitMIO}}$, which implies that $\mathcal{F}_{\text{SplitMIO}}$ is not integral. □

A.4. Example of a Non-TU ProductMIO Constraint Matrix

We present an instance of the ProductMIO formulation in which the constraint matrix is not totally unimodular. Consider a decision forest model consisting of a single tree, denoted by t. This tree has a depth of three and contains seven split nodes. The root split corresponds to product 1, the two splits at depth two correspond to product 2, and the four splits at depth three correspond to product 3. These split nodes follow the structure depicted in the left panel of Figure 2. Note that the products associated with the leaf nodes do not need to be specified, as $r_{t,\ell}$ appears only in the objective function of ProductMIO and does not influence the constraint matrix.

We now explicitly construct the constraint matrix. Since there is only one decision tree, we omit the subscript t. Ordering the variables as $(x_1, x_2, x_3, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$, the constraint matrix is given by

$$\begin{bmatrix} & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & & & 1 & 1 & 1 & 1 & & & & \\ & -1 & & 1 & 1 & & & 1 & 1 & & \\ & & -1 & 1 & & 1 & & 1 & & 1 & \\ 1 & & & & & & & 1 & 1 & 1 & 1 \\ & 1 & & & & & 1 & 1 & & 1 & 1 \\ & & 1 & & 1 & & 1 & & 1 & & 1 \end{bmatrix}, \tag{47}$$

where we omit the zeros. To demonstrate that this matrix is not totally unimodular, consider the submatrix formed by the second, third, and fourth rows and the columns corresponding to variables y_2, y_3, and y_5. This submatrix has the determinant

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} = -2.$$

Recall that a matrix is totally unimodular matrix if every square submatrix has a determinant of 0, 1 or -1. Therefore, the constraint matrix (47) is not totally unimodular.

A.5. Proof of Proposition 4

Define $\Delta^{\mathbf{leaves}} = \{\mathbf{y} \in \mathbb{R}^{|\mathbf{leaves}|} \mid \sum_{\ell \in \mathbf{leaves}} y_\ell = 1; y_\ell \geq 0, \forall\, \ell \in \mathbf{leaves}\}$ to be the $(|\mathbf{leaves}| - 1)$-dimensional unit simplex. In addition, for any $S \subseteq \mathbf{leaves}$, define $Q(S) = \{\mathbf{y} \in \Delta^{\mathbf{leaves}} \mid y_\ell \leq 0 \text{ for } \ell \in \mathbf{leaves} \setminus S\}$. We write the combinatorial disjunctive constraint over the ground set **leaves** as $\text{CDC}(\mathbf{leaves}) = \bigcup_{\ell \in \mathbf{leaves}} Q(\{\ell\})$. Consider now the optimization problem

$$\underset{\mathbf{y}}{\text{maximize}} \left\{ \sum_{\ell \in \mathbf{leaves}} r_\ell y_\ell \;\middle|\; \mathbf{y} \in \text{CDC}(\mathbf{leaves}) \right\}. \tag{48}$$

We will re-formulate this problem into a mixed-integer optimization problem. To do this, we claim that $\text{CDC}(\mathbf{leaves})$ can be written as the following pairwise independent branching scheme:

$$\bigcup_{\ell \in \mathbf{leaves}} Q(\{\ell\}) = \bigcup_{i=1}^{n} \left(Q(L_i) \cup Q(R_i) \right), \tag{49}$$

where $L_i = \{\ell \in \mathbf{leaves} \mid \ell \in \mathbf{left}(s) \text{ for some } s \text{ with } v(s) = i\}$ and $R_i = \{\ell \in \mathbf{leaves} \mid \ell \in \mathbf{right}(s) \text{ for some } s \text{ with } v(s) = i\}$. Note that (49) is equivalent to the statement

$$\bigcup_{\ell \in \mathbf{leaves}} \{\ell\} = \bigcup_{i=1}^{n} \left((\mathbf{leaves} \setminus L_i) \cup (\mathbf{leaves} \setminus R_i) \right). \tag{50}$$

To establish (50), it is sufficient to prove the following equivalence:

$$\{\ell\} = \bigcap_{i \in I(\ell)} (\mathbf{leaves} \setminus R_i) \cap \bigcap_{i \in E(\ell)} (\mathbf{leaves} \setminus L_i) \cap \bigcap_{\substack{i \in \mathcal{N}: \\ i \notin I(\ell) \cup E(\ell)}} (\mathbf{leaves} \setminus L_i), \tag{51}$$

where $I(\ell) = \{i \in \mathcal{N} \mid \ell \in \bigcup_{s: v(s) = i} \mathbf{left}(s)\}$ and $E(\ell) = \{i \in \mathcal{N} \mid \ell \in \bigcup_{s: v(s) = i} \mathbf{right}(s)\}$.

We now prove (51).

Equation (51)*, $\subseteq$ direction:* For $i \in I(\ell)$, we have that $\ell \in \bigcup_{s: v(s) = i} \mathbf{left}(s)$. This means that there exists $\bar{s}$ such that $\ell \in \mathbf{left}(\bar{s})$ and $v(\bar{s}) = i$. Since $\ell \in \mathbf{left}(\bar{s})$, this means that $\ell \notin \mathbf{right}(\bar{s})$ (a leaf cannot be to the

left and to the right of any split). Moreover, ℓ cannot be in $\mathbf{right}(s)$ for any other s with $v(s)=i$, because this would mean that product i appears more than once along the path to leaf ℓ, violating Assumption 1. Therefore, $\ell \in \mathbf{leaves} \setminus R_i$ for any $i \in I(\ell)$.

For $i \in E(\ell)$, we have that $\ell \in \bigcup_{s:v(s)=i} \mathbf{right}(s)$. This means that there exists a split $\bar{s}$ such that $\ell \in \mathbf{right}(\bar{s})$ and $v(\bar{s})=i$. Since $\ell \in \mathbf{right}(\bar{s})$, we have that $\ell \notin \mathbf{left}(\bar{s})$. In addition, ℓ cannot be in $\mathbf{left}(s)$ for any other s with $v(s)=i$. Therefore, $\ell \in \mathbf{leaves} \setminus L_i$ for any $i \in E(\ell)$.

Lastly, for any $i \notin I(\ell) \cup E(\ell)$, note that product i does not appear in any split along the path from the root of the tree to leaf ℓ. Therefore, for any s with $v(s)=i$, it will follow that either $\mathbf{left}(s) \subseteq \mathbf{left}(s')$ for some s' for which $\ell \in \mathbf{right}(s')$, or $\mathbf{left}(s) \subseteq \mathbf{right}(s')$ for some s' for which $\ell \in \mathbf{left}(s')$ – in other words, there is a split s' such that every leaf in $\mathbf{left}(s)$ is to one side of s' and ℓ is on the other side of s'. This means that ℓ' cannot be in $\mathbf{left}(s)$ for any s with $v(s)=i$, or equivalently, $\ell \in \mathbf{leaves} \setminus L_i$ for any $i \notin I(\ell) \cup E(\ell)$.

Equation (51)*, $\supseteq$ direction:* We will prove the contrapositive $\{\ell' \in \mathbf{leaves} \mid \ell' \neq \ell\} \subseteq \bigcup_{i \in I(\ell)} R_i \cup \bigcup_{i \in E(\ell)} L_i \cup \bigcup_{i \notin I(\ell) \cup E(\ell)} L_i$. A straightforward result (see Lemma EC.1 from Mišić 2020) is that $\{\ell' \in \mathbf{leaves} \mid \ell' \neq \ell\} = \bigcup_{s:\ell \in \mathbf{left}(s)} \mathbf{right}(s) \cup \bigcup_{s:\ell \in \mathbf{right}(s)} \mathbf{left}(s)$. Thus, if $\ell' \neq \ell$, then we have that $\ell' \in \mathbf{right}(s)$ for some s such that $\ell \in \mathbf{left}(s)$, or $\ell' \in \mathbf{left}(s)$ for some s such that $\ell \in \mathbf{right}(s)$. Let $i^* = v(s)$. In the first case, since $\ell \in \mathbf{left}(s)$, we have that $i^* \in I(\ell)$, and we thus have

$$\mathbf{right}(s) \subseteq \bigcup_{s':v(s')=i^*} \mathbf{right}(s') = R_{i^*} \subseteq \bigcup_{i \in I(\ell)} R_i \cup \bigcup_{i \in E(\ell)} L_i \cup \bigcup_{\substack{i' \in \mathcal{N}: \\ i' \notin I(\ell) \cup E(\ell)}} L_i.$$

In the second case, since $\ell \in \mathbf{right}(s)$, we have that $i^* \in E(\ell)$, and thus we have

$$\mathbf{left}(s) \subseteq \bigcup_{s':v(s')=i^*} \mathbf{left}(s') = L_{i^*} \subseteq \bigcup_{i \in I(\ell)} R_i \cup \bigcup_{i \in E(\ell)} L_i \cup \bigcup_{\substack{i' \in \mathcal{N}: \\ i' \notin I(\ell) \cup E(\ell)}} L_i.$$

This establishes the validity of the pairwise independent branching scheme (50). Thus, a valid formulation for problem (48) (see formulation (9) in Vielma et al. 2010, formulation (14) in Vielma and Nemhauser 2011 and formulation (13) in Huchette and Vielma 2019) is

$$\begin{aligned}
\underset{\mathbf{x},\mathbf{y}}{\text{maximize}} \quad & \sum_{\ell \in \mathbf{leaves}} r_\ell y_\ell && \text{(52a)}\\
\text{subject to} \quad & \sum_{\ell \in \mathbf{leaves}} y_\ell = 1, && \text{(52b)}\\
& \sum_{\ell \in L_i} y_\ell \leq x_i, \quad \forall\, i \in \mathcal{N}, && \text{(52c)}\\
& \sum_{\ell \in R_i} y_\ell \leq 1 - x_i, \quad \forall\, i \in \mathcal{N}, && \text{(52d)}\\
& y_\ell \geq 0, \quad \forall\, \ell \in \mathbf{leaves}, && \text{(52e)}\\
& x_i \in \{0,1\}, \quad \forall\, i \in \mathcal{N}. && \text{(52f)}
\end{aligned}$$

Observe that, by the definition of L_i and R_i, formulation (52) is identical to ProductMIO when $|F| = 1$. By invoking Theorem 1 from Vielma et al. (2010) with appropriate modifications, we can assert that formulation (52) is integral. Therefore, when $|F| = 1$, formulation ProductMIO is always integral. □

A.6. Proof of Proposition 5

Assume that $|F| = M$, where M is a constant, i.e., $M = O(1)$. Let $F = \{t_1, \ldots, t_M\}$, and let $\Omega = \prod_{m=1}^{M} \textbf{leaves}(t_m)$. Thus, each element $\boldsymbol{\omega} \in \Omega$ is a tuple $\boldsymbol{\omega} = (\tilde{\ell}_1, \ldots, \tilde{\ell}_M)$ of length M, where $\tilde{\ell}_m \in \textbf{leaves}(t_m)$ for $m = 1, 2, \ldots, M$.

For each m and each leaf $\ell_m \in \textbf{leaves}(t_m)$, we define the sets $I_m(\ell_m)$ and $E_m(\ell_m)$ as

$$I_m(\ell_m) = \{i \in \mathcal{N} \mid v(t_m, s) = i \text{ for some } s \in \textbf{splits}(t_m) \text{ such that } \ell_m \in \textbf{left}(s)\},$$
$$E_m(\ell_m) = \{i \in \mathcal{N} \mid v(t_m, s) = i \text{ for some } s \in \textbf{splits}(t_m) \text{ such that } \ell_m \in \textbf{right}(s)\}.$$

In words, $I_m(\ell_m)$ is the set of products that must be *i*ncluded in the assortment for leaf ℓ_m to be reached in the tree, while $E_m(\ell_m)$ is the set of products that must be *e*xcluded from the assortment for leaf ℓ_m to be reached in the tree. Note that for a single tree t_m, $I_m(\ell_m)$ and $E_m(\ell_m)$ will not intersect; this follows by Assumption 1. Note also that computing $I_m(\ell_m)$ and $E_m(\ell_m)$ can be done by traversing the tree backwards from the leaf ℓ_m to the root node of tree t_m. Since the number of leaves is polynomial in n, so is the number of splits (because $|\textbf{splits}(t)| = |\textbf{leaves}(t)| - 1$), and hence $I_m(\ell_m)$ and $E_m(\ell_m)$ can be computed in time polynomial in n.

Now, for every tuple of leaves $\boldsymbol{\omega} = (\tilde{\ell}_1, \ldots, \tilde{\ell}_M)$, define $I(\boldsymbol{\omega})$ and $E(\boldsymbol{\omega})$ as $I(\boldsymbol{\omega}) = \cup_{m=1}^{M} I_m(\tilde{\ell}_m)$, $E(\boldsymbol{\omega}) = \cup_{m=1}^{M} E_m(\tilde{\ell}_m)$. In words, $I(\boldsymbol{\omega})$ is the set of products that must be included in the assortment, and $E(\boldsymbol{\omega})$ is the set of products that must be excluded from the assortment, in order for each tree t_m maps the assortment to $\tilde{\ell}_m$. When $I(\boldsymbol{\omega})$ and $E(\boldsymbol{\omega})$ do not overlap, i.e., $I(\boldsymbol{\omega}) \cap E(\boldsymbol{\omega}) = \emptyset$, we say that $\boldsymbol{\omega}$ is feasible, because it is possible to select an assortment S such that the leaf tuple $\boldsymbol{\omega}$ is realized (any S such that $I(\boldsymbol{\omega}) \subseteq S$ and $S \cap E(\boldsymbol{\omega}) = \emptyset$ will suffice; a simple choice is $S = I(\boldsymbol{\omega})$). On the other hand, when $I(\boldsymbol{\omega})$ and $E(\boldsymbol{\omega})$ do overlap, i.e., $I(\boldsymbol{\omega}) \cap E(\boldsymbol{\omega}) \neq \emptyset$, we say that $\boldsymbol{\omega}$ is infeasible, because there exists a product that must be both included and excluded in the assortment in order for the trees to map the assortment to $\boldsymbol{\omega}$, which clearly cannot be accomplished. Given $I_m(\tilde{\ell}_m)$ and $E_m(\tilde{\ell}_m)$ for each m, computing $I(\boldsymbol{\omega})$ and $E(\boldsymbol{\omega})$ can be done in time polynomial in n and M, and testing whether $I(\boldsymbol{\omega}) \cap E(\boldsymbol{\omega}) = \emptyset$ can also be done in time polynomial in n.

We define the objective value $Z(\boldsymbol{\omega})$ of $\boldsymbol{\omega}$ be defined as

$$Z(\boldsymbol{\omega}) = \begin{cases} \sum_{m=1}^{M} \lambda_{t_m} r_{t_m, \tilde{\ell}_m} & \text{if } \boldsymbol{\omega} \text{ is feasible,} \\ -\infty & \text{if } \boldsymbol{\omega} \text{ is infeasible} \end{cases} \tag{53}$$

We now describe our algorithm. For each $\boldsymbol{\omega}$, compute $Z(\boldsymbol{\omega})$. Let $\boldsymbol{\omega}^* \in \arg\max Z(\boldsymbol{\omega})$ be any maximizer of $Z(\cdot)$ and Z^* be the corresponding maximum. Note that Z^* is finite, because any fixed assortment S maps to some leaf tuple $\boldsymbol{\omega}' \in \Omega$ that must have a finite value of Z. Let S be any assortment such that $S \supseteq I(\boldsymbol{\omega}^*)$ and $S \cap E(\boldsymbol{\omega}^*) = \emptyset$; as before, we can simply set $S = I(\boldsymbol{\omega}^*)$. Then S is an optimal solution to the assortment problem.

We now analyze the runtime. Observe that $|\Omega|$ is polynomial in n, since $|\textbf{leaves}(t_m)|$ is polynomial in n and $M = O(1)$. Additionally, for each $\boldsymbol{\omega}$, computing the function $Z(\omega)$ requires summing M numbers and determining the feasibility of $\boldsymbol{\omega}$, which we previously noted can be done in time polynomial in n. Thus, it follows that the total runtime of this algorithm is polynomial in n. $\square$

A.7. Proof of Theorem 2

Proof of part (a) (feasibility): By definition, the solution produced by Algorithm 1 produces a solution $\mathbf{y}_t$ that satisfies the left and right split constraints (10c) and (10d). With regard to the nonnegativity constraint (10e), we can see that at each stage of Algorithm 1, the quantities $x_{v(t,s)} - \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell}$, $1 - x_{v(t,s)} - \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell}$ and $1 - \sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell}$ never become negative; thus, the solution $\mathbf{y}_t$ produced upon termination satisfies the nonnegativity constraint (10e).

The only constraint that remains to be verified is constraint (10b), which requires that $\mathbf{y}_t$ adds up to 1. Observe that it is sufficient for a C event to occur during the execution of Algorithm 1 to ensure that constraint (10b) is satisfied. We will show that a C event must occur during the execution of Algorithm 1.

We proceed by contradiction. For the sake of a contradiction, let us suppose that a C event does not occur during the execution of the algorithm. Note that under this assumption, for any split s, it is impossible that the solution $\mathbf{y}_t$ produced by Algorithm 1 satisfies $x_{v(t,s)} = \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell}$ and $1 - x_{v(t,s)} = \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell}$, as this would imply that $\sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} + \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell} = x_{v(t,s)} + 1 - x_{v(t,s)} = 1$; by the definition of Algorithm 1, this would have triggered a C event at one of the leaves in $\mathbf{left}(s) \cup \mathbf{right}(s)$.

Thus, this means that at every split, either $\sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} < x_{v(t,s)}$ or $\sum_{\ell \in \mathbf{right}(s)} < 1 - x_{v(t,s)}$. Using this property, let us identify a leaf ℓ^* using the following procedure:

Procedure 1:
1. Set $j \leftarrow \mathbf{root}(t)$.
2. If $j \in \mathbf{leaves}(t)$, terminate with $\ell^* = j$; otherwise, proceed to Step 3.
3. If $\sum_{\ell \in \mathbf{left}(j)} y_{t,\ell} < x_{v(t,j)}$, set $j \leftarrow \mathbf{leftchild}(j)$; otherwise, set $j \leftarrow \mathbf{rightchild}(j)$.
4. Repeat Step 2.

Note that by our observation that at most one of the left or right split constraints can be satisfied at equality for any split s, Procedure 1 above is guaranteed to terminate with a leaf ℓ^* such that:

$$y_{t,\ell^*} \leq \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} < x_{v(t,s)}, \quad \forall\ s \in \mathbf{splits}(t) \text{ such that } \ell^* \in \mathbf{left}(s),$$

$$y_{t,\ell^*} \leq \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell} < 1 - x_{v(t,s)}, \quad \forall\ s \in \mathbf{splits}(t) \text{ such that } \ell^* \in \mathbf{right}(s).$$

However, this is impossible, because Algorithm 1 always sets each leaf $y_{t,\ell}$ to the highest value it can be without violating any of the left or right split constraints; the above conditions imply that y_{t,ℓ^*} could have been set higher, which is not possible. We thus have a contradiction, and it must be the case that a C event occurs.

Proof of part (b) (extreme point): To show that $\mathbf{y}_t$ is an extreme point, let us assume that $\mathbf{y}_t$ is not an extreme point. Then, there exist feasible solutions $\mathbf{y}_t^1$ and $\mathbf{y}_t^2$ different from $\mathbf{y}_t$ and a weight $\theta \in (0,1)$ such that $\mathbf{y}_t = \theta \mathbf{y}_t^1 + (1-\theta)\mathbf{y}_t^2$. Let ℓ^* be the first leaf checked by Algorithm 1 at which $y_{t,\ell^*} \neq y_{t,\ell^*}^1$ and $y_{t,\ell^*} \neq y_{t,\ell^*}^2$. Such a leaf must exist because $\mathbf{y}_t \neq \mathbf{y}_t^1$ and $\mathbf{y}_t \neq \mathbf{y}_t^2$, and because $\mathbf{y}_t$ is the convex combination of $\mathbf{y}_t^1$ and $\mathbf{y}_t^2$. Without loss of generality, let us further assume that $y_{t,\ell^*}^1 < y_{t,\ell^*} < y_{t,\ell^*}^2$.

By definition, Algorithm 1 sets $y_{t,\ell}$ at each iteration to the largest it can be without violating the left split constraints (10c) and the right split constraints (10d), and ensuring that $\sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell}$ does not exceed 1. Since $y_{t,\ell^*}^2 > y_{t,\ell^*}$, and since $\mathbf{y}_t^2$ and $\mathbf{y}_t$ are equal for all leaves checked before ℓ^*, this implies that $\mathbf{y}_t^2$ either violates constraint (10c), violates constraint (10d), or is such that $\sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} > 1$. This implies that $\mathbf{y}_t^2$ cannot be a feasible solution, which contradicts the assumption that $\mathbf{y}_t^2$ is a feasible solution. □

A.8. Proof of Theorem 3

Proof of part (a) (feasibility): Before we prove the result, we first establish a helpful property of the events that are triggered during the execution of Algorithm 1.

LEMMA 1. *Let $s_1, s_2 \in \mathbf{splits}(t)$, $s_1 \neq s_2$, such that s_2 is a descendant of s_1. Suppose that $e_1 = A_{s_1}$ or $e_1 = B_{s_1}$, and that $e_2 = A_{s_2}$ or $e_2 = B_{s_2}$. If e_1 and e_2 occur during the execution of Algorithm 1, then $r_{t,f(e_1)} \leq r_{t,f(e_2)}$.*

Proof: We will prove this by contradiction. Suppose that we have two splits s_1 and s_2 and events e_1 and e_2 as in the statement of the lemma, and that $r_{t,f(e_2)} < r_{t,f(e_1)}$. This implies that leaf $f(e_1)$ is checked before leaf $f(e_2)$. When leaf $f(e_1)$ is checked, the event e_1 occurs, which implies that either the left split constraint (10c) becomes tight (if $e_1 = A_{s_1}$) or the right split constraint (10d) becomes tight (if $e_1 = B_{s_1}$) at s_1. In either case, since s_2 is a descendant of s_1, the leaf $f(e_2)$ must be contained in the left leaves of split s_1 (if $e = A_{s_1}$) or the right leaves of split s_1 (if $e_1 = B_{s_1}$). Thus, when leaf $f(e_2)$ is checked, the event e_2 cannot occur, because q_{s_1} in Algorithm 1 will be zero (implying that $q_{A,B} = 0$), and so s^* cannot be equal to s_2 because s_1 is a shallower split that attains the minimum of $q_{A,B} = 0$. □

To establish that $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is feasible for the SPLITMIO dual subproblem (11), we will first show that the $\alpha_{t,s}$ variables are nonnegative. Fix $s \in \mathbf{splits}(t)$. If $A_s \notin \mathcal{E}$, then $\alpha_{t,s} = 0$, and constraint (11c) is satisfied. If $A_s \in \mathcal{E}$, then consider the split $\tilde{s} = \arg\min_{s'} [\{d(s') \mid s' \in \mathbf{LS}(f(A_s)),\ d(s') < d,\ A_{s'} \in \mathcal{E}\} \cup \{d(s') \mid s' \in \mathbf{RS}(f(A_s)),\ d(s') < d,\ B_{s'} \in \mathcal{E}\}]$, where we recall that $d = d(s)$ is the depth of split s. In words, $\tilde{s}$ is the shallowest split (i.e., closest to the root) along the path of splits from the root node to split s such that either an $A_{\tilde{s}}$ event occurs or a $B_{\tilde{s}}$ event occurs for split $\tilde{s}$. There are three possible cases that can occur here, which we now handle.

Case 1: $\tilde{s} \in \mathbf{LS}(f(A_s))$. In this case, $A_{\tilde{s}} \in \mathcal{E}$, and we have

$$\begin{aligned}
\alpha_{t,s} &= r_{t,f(A_s)} - \left[\sum_{s' \in \mathbf{LS}(f(A_s)):\ d(s') < d,\ A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(A_s)):\ d(s') < d,\ B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - \left[\alpha_{t,\tilde{s}} + \sum_{s' \in \mathbf{LS}(f(A_s)):\ d(s') < d(\tilde{s}),\ A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(A_s)):\ d(s') < d(\tilde{s}),\ B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - \left[\alpha_{t,\tilde{s}} + \sum_{s' \in \mathbf{LS}(f(A_{\tilde{s}})):\ d(s') < d(\tilde{s}),\ A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(A_{\tilde{s}})):\ d(s') < d(\tilde{s}),\ B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - r_{t,f(A_{\tilde{s}})} \\
&\geq 0,
\end{aligned}$$

where the first step follows by the definition of $\alpha_{t,s}$ in Algorithm 2; the second step follows by the definition of $\alpha_{t,\tilde{s}}$ as the deepest split for which an A or B event occurs that is at a depth lower than s; the third step by the fact that the left splits and right splits of $f(A_{\tilde{s}})$ at a depth below $d(\tilde{s})$ are the same as the left and right splits of $f(A_s)$ at a depth below $d(\tilde{s})$; and the fourth step follows from the definition of $\alpha_{t,\tilde{s}}$ in Algorithm 2. The inequality follows by Lemma 1.

Case 2: $\tilde{s} \in \mathbf{RS}(f(A_s))$. In this case, $B_{\tilde{s}} \in \mathcal{E}$, and analogously to Case 1, we have:

$$\begin{aligned}
\alpha_{t,s} &= r_{t,f(A_s)} - \left[\sum_{s' \in \mathbf{LS}(f(A_s)):\, d(s') < d,\, A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(A_s)):\, d(s') < d, B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - \left[\beta_{t,\tilde{s}} + \sum_{s' \in \mathbf{LS}(f(A_s)):\, d(s') < d(\tilde{s}),\, A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(A_s)):\, d(s') < d(\tilde{s}),\, B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - \left[\beta_{t,\tilde{s}} + \sum_{s' \in \mathbf{LS}(f(B_{\tilde{s}})):\, d(s') < d(\tilde{s}),\, A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(f(B_{\tilde{s}})):\, d(s') < d(\tilde{s}),\, B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t \right] \\
&= r_{t,f(A_s)} - r_{t,f(B_{\tilde{s}})} \\
&\geq 0.
\end{aligned}$$

Case 3: $\tilde{s}$ is undefined because the underlying sets are empty. In this case, $\alpha_{t,s} = r_{t,f(A_s)} - \gamma_t$, and we have $\alpha_{t,s} = r_{t,f(A_s)} - \gamma_t = r_{t,f(A_s)} - r_{t,f(C)} \geq 0$, where the inequality follows because $f(C)$ is the last leaf to be checked before Algorithm 1 terminates, and thus it must be that $r_{t,f(A_s)} \geq r_{t,f(C)}$. This establishes that $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ satisfy constraint (11c). Constraint (11d) can be shown in an almost identical fashion; for brevity, we omit the steps. We thus only need to verify constraint (11b). Let $\ell \in \mathbf{leaves}(t)$. Here, there are four mutually exclusive and collectively exhaustive cases to consider.

Case 3.1: $r_{t,\ell} \leq r_{t,f(C)}$. In this case we have $\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t \geq \gamma_t = r_{t,f(C)} \geq r_{t,\ell}$.

Case 3.2: $r_{t,\ell} > r_{t,f(C)}$ and $\ell = f(A_s)$ for some $s \in \mathbf{splits}(t)$. In this case, we have

$$\sum_{s' \in \mathbf{LS}(\ell)} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(\ell)} \beta_{t,s'} + \gamma_t \geq \alpha_{t,s} + \sum_{\substack{s' \in \mathbf{LS}(\ell): \\ d(s') < d(s),\, A_{s'} \in \mathcal{E}}} \alpha_{t,s'} + \sum_{\substack{s' \in \mathbf{RS}(\ell): \\ d(s') < d(s),\, B_{s'} \in \mathcal{E}}} \beta_{t,s'} + \gamma_t = r_{t,f(A_s)} = r_{t,\ell},$$

where the first step follows by the nonnegativity of $\alpha_{t,s'}$ and $\beta_{t,s'}$ for all s', and the second step by the definition of $\alpha_{t,s}$ in Algorithm 2.

Case 3.3: $r_{t,\ell} > r_{t,f(C)}$ and $\ell = f(B_s)$ for some $s \in \mathbf{splits}(t)$. By similar logic as case 2, we have

$$\sum_{s' \in \mathbf{LS}(\ell)} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(\ell)} \beta_{t,s'} + \gamma_t \geq \beta_{t,s} + \sum_{\substack{s' \in \mathbf{LS}(\ell): \\ d(s') < d(s),\, A_{s'} \in \mathcal{E}}} \alpha_{t,s'} + \sum_{\substack{s' \in \mathbf{RS}(\ell): \\ d(s') < d(s),\, B_{s'} \in \mathcal{E}}} \beta_{t,s'} + \gamma_t = r_{t,f(B_s)} = r_{t,\ell}.$$

Case 3.4: $r_{t,\ell} > r_{t,f(C)}$ and ℓ is not equal to $f(A_s)$ or $f(B_s)$ for any split s. In this case, when leaf ℓ is checked by Algorithm 1, the algorithm reaches line 17 where s^* is determined and e is set to either A_{s^*} or B_{s^*}, and it turns out that e is already in $\mathcal{E}$. If $e = A_{s^*}$, then this means that leaf $f(A_{s^*})$ was checked before leaf ℓ, and that $r_{t,\ell} \leq r_{t,f(A_{s^*})}$. We thus have

$$\begin{aligned}
\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t &\geq \alpha_{t,s^*} + \sum_{s \in \mathbf{LS}(\ell):\, d(s) < d(s^*),\, A_s \in \mathcal{E}} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell):\, d(s) < d(s^*),\, B_s \in \mathcal{E}} \beta_{t,s} + \gamma_t \\
&= \alpha_{t,s^*} + \sum_{\substack{s \in \mathbf{LS}(f(A_{s^*})): \\ d(s) < d(s^*),\, A_s \in \mathcal{E}}} \alpha_{t,s} + \sum_{\substack{s \in \mathbf{RS}(f(A_{s^*})): \\ d(s) < d(s^*),\, B_s \in \mathcal{E}}} \beta_{t,s} + \gamma_t \\
&= r_{t,f(A_{s^*})} \\
&\geq r_{t,\ell},
\end{aligned}$$

where the first equality follows because ℓ and $f(A_{s^*})$, by virtue of being to the left of s^*, share the same left and right splits at depths lower than $d(s^*)$. Similarly, if $e = B_{s^*}$, then the leaf $f(B_{s^*})$ was checked before ℓ, which means that $r_{t,\ell} \leq r_{t,f(B_{s^*})}$; in this case, we have

$$\begin{aligned}
\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t &\geq \beta_{t,s^*} + \sum_{s \in \mathbf{LS}(\ell):\ d(s) < d(s^*),\ A_s \in \mathcal{E}} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell):\ d(s) < d(s^*),\ B_s \in \mathcal{E}} \beta_{t,s} + \gamma_t \\
&\geq \beta_{t,s^*} + \sum_{\substack{s \in \mathbf{LS}(f(B_{s^*})): \\ d(s) < d(s^*),\ A_s \in \mathcal{E}}} \alpha_{t,s} + \sum_{\substack{s \in \mathbf{RS}(f(B_{s^*})): \\ d(s) < d(s^*),\ B_s \in \mathcal{E}}} \beta_{t,s} + \gamma_t \\
&= r_{t,f(B_{s^*})} \\
&\geq r_{t,\ell}.
\end{aligned}$$

We have thus shown that $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a feasible solution to the SplitMIO dual subproblem (11).

Proof of part (b) (extreme point): To prove this, we will use the equivalence between extreme points and basic feasible solutions, and show that $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a basic feasible solution of problem (11).

Define the sets $L_A = \{\ell \in \mathbf{leaves}(t) \mid \ell = f(A_s) \text{ for some } s \in \mathbf{splits}(t)\}$ and $L_B = \{\ell \in \mathbf{leaves}(t) \mid \ell = f(B_s) \text{ for some } s \in \mathbf{splits}(t)\}$. Consider the following system of equations:

$$\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t = r_{t,\ell}, \quad \forall \ell \in L_A, \tag{54}$$

$$\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t = r_{t,\ell}, \quad \forall \ell \in L_B, \tag{55}$$

$$\sum_{s \in \mathbf{LS}(f(C))} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(f(C))} \beta_{t,s} + \gamma_t = r_{t,f(C)}, \tag{56}$$

$$\alpha_{t,s} = 0, \quad \forall\ s \text{ such that } A_s \notin \mathcal{E}, \tag{57}$$

$$\beta_{t,s} = 0, \quad \forall\ s \text{ such that } B_s \notin \mathcal{E}. \tag{58}$$

Observe that each equation corresponds to a constraint from problem (11) made to hold at equality. In addition, we note that there are $|L_A| + |L_B| + 1 + (|\mathbf{splits}(t)| - |L_A|) + (|\mathbf{splits}(t)| - |L_B|) = 2|\mathbf{splits}(t)| + 1$ equations, which is exactly the number of variables. We will show that the unique solution implied by this system of equations is exactly the solution $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ that is produced by Algorithm 2.

In order to establish this, we first establish a couple of useful results.

Lemma 2. *Suppose that $e \in \mathcal{E}$, $\ell = f(e)$ and $e = A_s$ or $e = B_s$ for some $s \in \mathbf{splits}(t)$. Then: (a) $A_{s'} \notin \mathcal{E}$ for all $s' \in \mathbf{LS}(\ell)$ such that $d(s') > d(s)$; and (b) $B_{s'} \notin \mathcal{E}$ for all $s' \in \mathbf{RS}(\ell)$ such that $d(s') > d(s)$.*

Proof of Lemma 2: We will prove this by contradiction. Without loss of generality, let us suppose that there exists an $A_{s'}$ event in $\mathcal{E}$ where $s' \in \mathbf{LS}(\ell)$ and $d(s') > d(s)$. (The case where there exists an $B_{s'}$ event in $\mathcal{E}$ where $s' \in \mathbf{RS}(\ell)$ and $d(s') > d(s)$ can be shown almost identically.)

Since $A_{s'} \in \mathcal{E}$, consider the leaf $\ell' = f(A_{s'})$. There are now two possibilities for when Algorithm 1 checks leaf ℓ':

1. **Case 1**: Leaf ℓ' is checked after leaf ℓ. In this case, in the iteration of Algorithm 1 corresponding to leaf ℓ', it will be the case that $q_s = 0$ because the left constraint (10c) at split s (if $e = A_s$) or the right constraint (10d) at split s (if $e = B_s$) became tight when leaf ℓ was checked. As a result, $q_{A,B} = 0$ in the iteration for leaf ℓ'. This implies that s' cannot be the lowest depth split that attains the minimum q_s value of $q_{A,B}$, because $q_s = 0$, and s has a depth lower than s', which contradicts the fact that the $A_{s'}$ event occurred.

2. **Case 2**: Leaf ℓ' is checked before leaf ℓ. In this case, consider the value of q_s when leaf ℓ is checked in Algorithm 1.

If $q_s > 0$, then there is immediately a contradiction, because $q_{s'} = 0$ when leaf ℓ is checked (this is true because the left split constraint (10c) at s' became tight after leaf ℓ' was checked), and thus it is impossible that $s^* = s$. If $q_s = 0$, then this implies that $x_{v(t,s)} = 0$. This would imply that $q_s = 0$ when leaf ℓ' was checked, which would imply that s^* cannot be s' when leaf ℓ' is checked because s is at a lower depth than s'.

Thus, in either case, we arrive at a contradiction, which completes the proof. □

LEMMA 3. *Suppose $\ell = f(C)$. Then: (a) $A_{s'} \notin \mathcal{E}$ for all $s' \in \mathbf{LS}(\ell)$; and (b) $B_{s'} \notin \mathcal{E}$ for all $s' \in \mathbf{RS}(\ell)$.*

Proof of Lemma 3: We proceed by contradiction. Suppose that A_s occurs for some $s \in \mathbf{LS}(\ell)$ or that B_s occurs for some $s \in \mathbf{LS}(\ell)$; in the former case, let $e = A_s$, and in the latter case, let $e = B_s$. Let $\ell' = f(e)$. Then ℓ' must be checked before ℓ by Algorithm 1, since the algorithm always terminates after a C event occurs. Consider what happens when Algorithm 1 checks leaf ℓ:

1. **Case 1**: $q_C > 0$. This is impossible, because if e occurs, then q_s when leaf ℓ is checked would have to be 0, which would imply that $q_{A,B} < q_C$ and that a C event could not have occurred when ℓ was checked.

2. **Case 2**: $q_C = 0$. This is also impossible, because it implies that the unit sum constraint (10b) was satisfied at an earlier iteration, which would have triggered the C event at a leaf that was checked before ℓ.

We thus have that $A_{s'}$ does not occur for any $s' \in \mathbf{LS}(\ell)$ and $B_{s'}$ does not occur for any $s' \in \mathbf{RS}(\ell)$. □

With these two lemmas in hand, we now return to the proof of Theorem A.8 (b). Observe now that by using Lemmas 2 and 3 and using equations (59) and (60), the system of equations (54)-(58) is equivalent to

$$\alpha_{t,s} + \sum_{s' \in \mathbf{LS}(f(A_s)):\ d(s') < d(s),\ A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(\ell):\ d(s') < d(s),\ B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t = r_{t,f(A_s)}, \quad \forall s \text{ such that } A_s \in \mathcal{E},$$

$$\beta_{t,s} + \sum_{s' \in \mathbf{LS}(\ell):\ d(s') < d(s),\ A_{s'} \in \mathcal{E}} \alpha_{t,s'} + \sum_{s' \in \mathbf{RS}(\ell):\ d(s') < d(s),\ B_{s'} \in \mathcal{E}} \beta_{t,s'} + \gamma_t = r_{t,f(B_s)}, \quad \forall s \text{ such that } B_s \in \mathcal{E},$$

$$\gamma_t = r_{t,f(C)},$$

$$\alpha_{t,s} = 0, \quad \forall\ s \text{ such that } A_s \notin \mathcal{E}, \tag{59}$$

$$\beta_{t,s} = 0, \quad \forall\ s \text{ such that } B_s \notin \mathcal{E}. \tag{60}$$

Thus the solution implied by this system of equations is exactly the solution produced by Algorithm 2, establishing that $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is a basic feasible solution of problem (11), and thus an extreme point. □

A.9. Proof of Theorem 4

To prove that the $\mathbf{y}_t$ and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ produced by Algorithms 1 and 2 are optimal for their respective problems, we show that they satisfy complementary slackness for problems (10) and (11):

$$\alpha_{t,s} \cdot \left(x_{v(t,s)} - \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} \right) = 0, \quad \forall\ s \in \mathbf{splits}(t), \tag{61}$$

$$\beta_{t,s} \cdot \left(1 - x_{v(t,s)} - \sum_{\ell \in \mathbf{right}(s)} y_{t,\ell} \right) = 0, \quad \forall\ s \in \mathbf{splits}(t), \tag{62}$$

$$y_{t,\ell} \cdot \left(\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t - r_{t,\ell} \right) = 0, \quad \forall\ \ell \in \mathbf{leaves}(t). \tag{63}$$

Condition (61): If $\alpha_{t,s} = 0$, then the condition is trivially satisfied. If $\alpha_{t,s} > 0$, then this implies that $A_s \in \mathcal{E}$. This means that the left split constraint (10c) at s became tight after leaves $f(A_s)$ was checked, which implies that $\sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} = x_{v(t,s)}$ or equivalently, that $x_{v(t,s)} - \sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} = 0$, which again implies that the condition is satisfied.

Condition (62): This follows along similar logic to condition (61), only that we use the fact that $\beta_{t,s} > 0$ implies that a B_s event occurred and that the right split constraint (10d) at s became tight.

Condition (63): If $y_{t,\ell} = 0$, then the condition is trivially satisfied. If $y_{t,\ell} > 0$, then either $\ell = f(C)$, or $\ell = f(A_s)$ for some split $s \in \mathbf{LS}(\ell)$, or $\ell = f(B_s)$ for some split $s \in \mathbf{RS}(\ell)$. In any of these three cases, as shown in the proof of part (b) of Theorem 3, the dual constraint (11b) holds with equality for any such leaf ℓ. Thus, we have that $\sum_{s \in \mathbf{LS}(\ell)} \alpha_{t,s} + \sum_{s \in \mathbf{RS}(\ell)} \beta_{t,s} + \gamma_t - r_{t,\ell} = 0$, and the condition is again satisfied.

As complementary slackness holds, $\mathbf{y}_t$ is feasible for the primal problem (10) (Theorem A.7), and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ is feasible for the dual problem (11) (Theorem 3). Thus they are optimal for their respective problems. □

A.10. Proof of Theorem 5

Proof of part (a): Observe that by construction, $\mathbf{y}_t$ automatically satisfies the unit sum constraint (10b) and the nonnegativity constraint (10e). We thus need to verify constraints (10c) and (10d). For constraint (10c), observe that for any split $s \notin \mathbf{LS}(\ell^*)$, it must be that $\ell^* \notin \mathbf{left}(s)$. Thus, we will have $\sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} = 0$, which means that constraint (10c) is automatically satisfied, because the right hand side $x_{v(t,s)}$ is always at least 0. On the other hand, for any split $s \in \mathbf{LS}(\ell^*)$, we will have that $x_{v(t,s)} = 1$, and that $\sum_{\ell \in \mathbf{left}(s)} y_{t,\ell} = \sum_{\ell \in \mathbf{left}(s): \ell \neq \ell^*} y_{t,\ell} + y_{t,\ell^*} = 1$, which implies that constraint (10c) is satisfied. Similar reasoning can be used to establish that constraint (10d) holds. This establishes that $\mathbf{y}_t$ is indeed a feasible solution of problem (10).

Proof of part (b): By construction, $\alpha_{t,s} \geq 0$ and $\beta_{t,s} \geq 0$ for all $s \in \mathbf{splits}(t)$, so constraints (11c) and (11d) are satisfied. To verify constraint (11b), fix a leaf $\ell \in \mathbf{leaves}(t)$. If $\ell \neq \ell^*$, then either $\ell \in \mathbf{left}(s')$ for some $s' \in \mathbf{RS}(\ell^*)$ or $\ell \in \mathbf{right}(s')$ for some $s' \in \mathbf{LS}(\ell^*)$. If $\ell \in \mathbf{left}(s')$ for some $s' \in \mathbf{RS}(\ell^*)$, then

$$\sum_{s: \ell \in \mathbf{left}(s)} \alpha_{t,s} + \sum_{s: \ell \in \mathbf{right}(s)} \beta_{t,s} + \gamma_t \geq \alpha_{t,s'} + \gamma_t \geq \max_{\ell' \in \mathbf{left}(s')} r_{t,\ell'} - r_{t,\ell^*} + r_{t,\ell^*} \geq r_{t,\ell}$$

where the first inequality follows because $\ell \in \mathbf{left}(s')$ and the fact that all $\alpha_{t,s}$ and $\beta_{t,s}$ variables are nonnegative; the second follows by how the dual solution is defined in the statement of the theorem; and the third by the definition of the maximum. Similarly, if $\ell \in \mathbf{right}(s')$ for some $s' \in \mathbf{LS}(\ell^*)$, then we have

$$\sum_{s:\ell \in \mathbf{left}(s)} \alpha_{t,s} + \sum_{s:\ell \in \mathbf{right}(s)} \beta_{t,s} + \gamma_t \geq \beta_{t,s'} + \gamma_t \geq \max_{\ell' \in \mathbf{right}(s')} r_{t,\ell'} - r_{t,\ell^*} + r_{t,\ell^*} \geq r_\ell - r_{t,\ell^*} + r_{t,\ell^*} = r_{t,\ell}.$$

Lastly, if $\ell = \ell^*$, then we automatically have $\sum_{s:\ell^* \in \mathbf{left}(s)} \alpha_{t,s} + \sum_{s:\ell^* \in \mathbf{right}(s)} \beta_{t,s} + \gamma_t \geq \gamma_t = r_{t,\ell^*}$. Thus, we have established that constraint (11b) is satisfied for all leaves ℓ, and thus $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ as defined in the statement of the theorem is a feasible solution of the dual (11).

Proof of part (c): To establish that the two solutions are optimal, by weak duality it is sufficient to show that the two solutions attain the same objective values in their respective problems. For the primal solution $\mathbf{y}_t$, it is immediately clear that its objective is r_{t,ℓ^*}. For the dual solution $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$, we have

$$\sum_{s \in \mathbf{splits}(t)} \alpha_{t,s} x_{v(t,s)} + \sum_{s \in \mathbf{splits}(t)} \beta_{t,s}(1 - x_{v(t,s)}) + \gamma_t = \sum_{s \in \mathbf{RS}(\ell^*)} \alpha_{t,s} x_{v(t,s)} + \sum_{s \in \mathbf{LS}(\ell^*)} \beta_{t,s}(1 - x_{v(t,s)}) + \gamma_t = \gamma_t = r_{t,\ell^*},$$

where the first step follows because $\alpha_{t,s} = 0$ for $s \notin \mathbf{RS}(\ell^*)$ and $\beta_{t,s} = 0$ for $s \notin \mathbf{LS}(\ell^*)$; the second step follows by the fact that $x_{v(t,s)} = 0$ for $s \in \mathbf{RS}(\ell^*)$ and $x_{v(t,s)} = 1$ for $s \in \mathbf{LS}(\ell^*)$; and the final step follows just by the definition of γ_t. This establishes that $\mathbf{y}_t$ and $(\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \gamma_t)$ are optimal for their respective problems. □

A.11. Proof of Theorem 6

The proof follows a similar argument to the proof of Theorem 5. For brevity, we omit the proof.

Appendix B: Example of Benders algorithms for SplitMIO

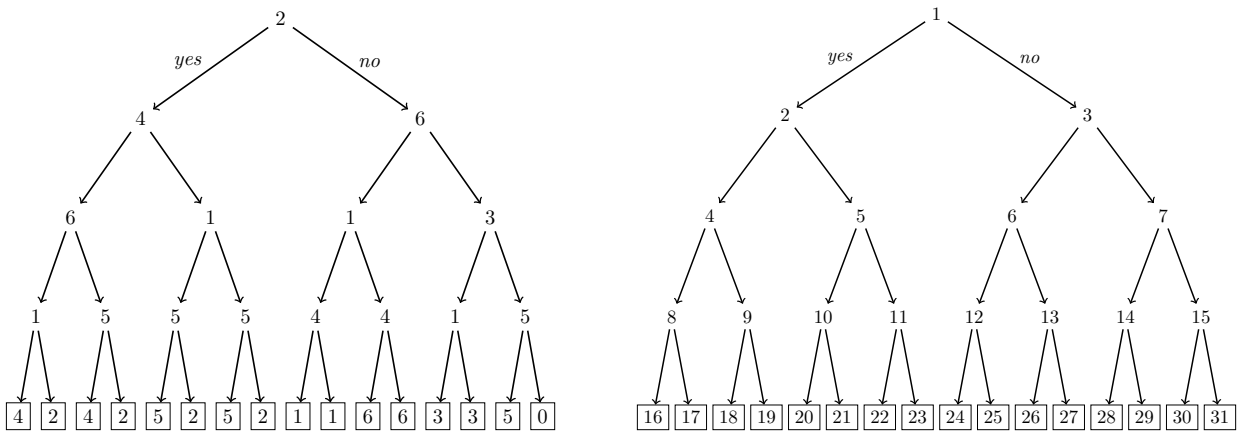
In this section, we provide a small example of the primal-dual procedure (Algorithms 1 and 2) for solving the SplitMIO subproblem. Suppose that $n = 6$, and that $\mathbf{x} = (x_1, \ldots, x_6) = (0.62, 0.45, 0.32, 0.86, 0.05, 0.35)$. Suppose that $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_6) = (97, 72, 89, 50, 100, 68)$. Suppose that the purchase decision tree t has the form given in Figure 8a; in addition, suppose that the splits and leaves are indexed as in Figure 8b. For example, in the latter case, 8 corresponds to the split node that is furthest to the bottom and to the left, while 30 corresponds to the second leaf from the right.

We first run Algorithm 1 on the problem, which carries out the steps shown below in Table 4. For this execution of the procedure, we assume that the following ordering of leaves (encoded by τ) is used:

$$20, 22, 30, 24, 25, 28, 29, 17, 19, 21, 23, 26, 27, 16, 18, 31.$$

After running the procedure, the primal solution $\mathbf{y} = (y_{16}, \ldots, y_{31})$ satisfies that $y_{17} = y_{24} = 0.35$, $y_{28} = 0.15$, $y_{20} = y_{22} = y_{30} = 0.05$ and all other components are zero. Here we drop the index t to simplify the notation. The event set is $\mathcal{E} = \{A_{10}, A_{11}, A_{15}, A_3, B_1, C\}$, and the function $f : \mathcal{E} \to \mathbf{leaves}$ is defined as $f(A_{10}) = 20$, $f(A_{11}) = 22$, $f(A_{15}) = 30$, $f(A_3) = 24$, $f(B_1) = 28$, and $f(C) = 17$.

We now run Algorithm 2, which carries out the steps shown in Table 5. We obtain a dual solution $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma)$, where $\gamma = 72$, $\alpha_{10} = \alpha_{11} = 28$, $\alpha_{15} = 11$, $\beta_1 = 17$, and all other components of $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{15})$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{15})$ are zero.



(a) Purchase decision tree. (b) Indexing of nodes in tree.

Figure 8 Tree used in example of SplitMIO primal-dual algorithms.

Iteration	Values of q_C and $q_{A,B}$	Steps
$\ell = 20$	$q_C = 1.0$, $q_{A,B} = 0.05$	Set $y_{20} \leftarrow 0.05$ A_{10} event
$\ell = 22$	$q_C = 0.95$, $q_{A,B} = 0.05$	Set $y_{22} \leftarrow 0.05$ A_{11} event
$\ell = 30$	$q_C = 0.90$, $q_{A,B} = 0.05$	Set $y_{30} \leftarrow 0.05$ A_{15} event
$\ell = 24$	$q_C = 0.85$, $q_{A,B} = 0.35$	Set $y_{24} \leftarrow 0.35$ A_3 event
$\ell = 25$	$q_C = 0.5$, $q_{A,B} = 0.0$	Set $y_{25} \leftarrow 0.0$
$\ell = 28$	$q_C = 0.5$, $q_{A,B} = 0.15$	Set $y_{28} \leftarrow 0.15$ B_1 event
$\ell = 29$	$q_C = 0.35$, $q_{A,B} = 0.0$	Set $y_{29} \leftarrow 0.0$
$\ell = 17$	$q_C = 0.35$, $q_{A,B} = 0.35$	Set $y_{17} \leftarrow 0.35$ C event **break**

Table 4 Steps of primal procedure (Algorithm 1).

Phase	Calculation
Initialization	$\alpha_s \leftarrow 0$, $\beta_s \leftarrow 0$ for all s
Set γ	$\gamma \leftarrow r_{17} = 72$
Loop: $d = 1$	$\beta_1 \leftarrow r_{28} - \gamma = 89 - 72 = 17$
Loop: $d = 2$	$\alpha_3 \leftarrow r_{24} - \gamma - \beta_1 = 97 - 72 - 17 = 8$
Loop: $d = 4$	$\alpha_{10} \leftarrow r_{20} - \gamma = 100 - 72 = 28$
	$\alpha_{11} \leftarrow r_{22} - \gamma = 100 - 72 = 28$
	$\alpha_{15} \leftarrow r_{30} - \gamma - \beta_1 = 100 - 72 - 11 = 11$

Table 5 Steps of dual procedure (Algorithm 2).

The feasibility of the dual solution is visualized in Figure 9. The colored bars correspond to the different dual variables; a colored bar appears multiple times when the variable participates in multiple dual constraints. The height of the black lines for each leaf indicates the value of r_ℓ, while the total height of the colored bars at a leaf corresponds to the value $\gamma + \sum_{s \in \mathbf{LS}(\ell)} \alpha_s + \sum_{s \in \mathbf{RS}(\ell)} \beta_s$ (the left hand side of the dual constraint (11b)). For each leaf, the total height of the colored bars exceeds the black line, which indicates that all dual constraints are satisfied.

The objective value of the primal solution is $\sum_{\ell \in \mathbf{leaves}} r_\ell \cdot y_\ell = r_{20} \times y_{20} + r_{22} \times y_{22} + r_{30} \times y_{30} + r_{24} \times y_{24} + r_{28} \times y_{28} + r_{17} \times y_{17} = 87.5$. The objective value of the dual solution is $\gamma + (1 - x_2) \times \beta_1 + x_6 \times \alpha_3 + x_5 \times \alpha_{10} + x_5 \times \alpha_{11} + x_5 \times \alpha_{15} = 72.0 + 0.55 \times 17 + 0.35 \times 8 + 0.05 \times 28 + 0.05 \times 28 + 0.05 \times 11 = 87.5$

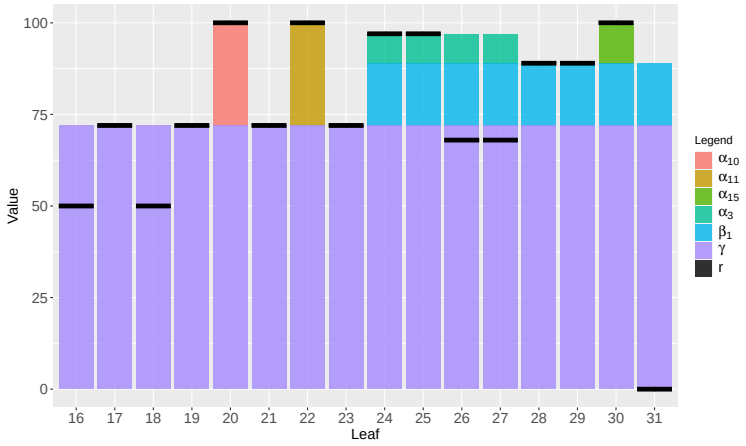


Figure 9 **Visualization of feasibility of dual solution.**

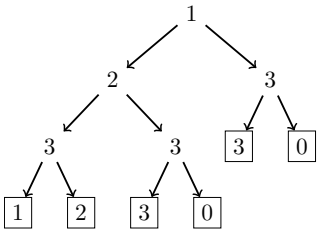


Figure 10 **Structure of tree for which the ProductMIO primal subproblem is not solvable via a greedy algorithm.**

Appendix C: Benders subproblem of the ProductMIO relaxation

We continue the discussion in Section 4.2 and show that Problem (13) cannot be solved with a greedy approach. We formalize this claim as the following proposition.

PROPOSITION 6. *There exists an* $\mathbf{x} \in [0,1]^n$*, a tree* t *and revenues* $\rho_1, \ldots, \rho_n$ *for which the greedy solution to problem* (13) *is not optimal.*

We prove this proposition by providing a counterexample. Consider an instance where $\mathcal{N} = \{1,2,3\}$ and $\mathbf{x} = (0.5, 0.5, 0.5)$. Assume the revenues of the products are $\rho_1 = 20$, $\rho_2 = 19$ and $\rho_3 = 18$. Consider the tree shown in Figure 10. We label the leaves as 1, 2, 3, 4, 5 and 6 from left to right. When we apply the greedy algorithm to solve this LO problem, we can see that there are multiple orderings of the leaves in decreasing revenue: (i) 1,2,3,5,4,6; (ii) 1,2,5,3,4,6; (iii) 1,2,3,5,6,4; and (iv) 1,2,5,3,6,4. For any of these orderings, the greedy solution will turn out to be $(y_1, y_2, y_3, y_4, y_5, y_6) = (0.5, 0, 0, 0, 0, 0.5)$, resulting in an objective value of 10. However, the actual optimal solution $G_t(\mathbf{x})$ turns out to be $(y_1^*, y_2^*, y_3^*, y_4^*, y_5^*, y_6^*) = (0, 0.5, 0, 0, 0.5, 0)$, for which the objective value is 18.5.

This counterexample shows that in general, the PRODUCTMIO primal subproblem cannot be solved to optimality via the same type of greedy algorithm as for SPLITMIO.

Appendix D: Additional Numerical Results Based on Synthetic Data

Our experiments were implemented in the Julia programming language, version 1.8.4 (Bezanson et al. 2017) and executed on Amazon Elastic Compute Cloud (EC2) using a single instance of type `m6a.48xlarge` (AMD EPYC 7R13 processor with 2.95GHz clock speed, 192 virtual CPUs and 768 GB memory). All LO and MIO formulations were solved using Gurobi version 10.0.1 and modeled using the `JuMP` package (Lubin and Dunning 2015), with a maximum of four cores per formulation.

D.1. Experiment: Tractability

In this experiment, we seek to understand the tractability of SplitMIO and ProductMIO when they are solved as integer problems (i.e., not as relaxations). For a given instance and a given formulation $\mathcal{M}$, we solve the integer version of formulation $\mathcal{M}$. Due to the large size of some of the problem instances, we impose a computation time limit of 1 hour for each formulation. We record $T_{\mathcal{M}}$, the computation time required for formulation $\mathcal{M}$, and we record $O_{\mathcal{M}}$ which is the final optimality gap, and is defined as $O_{\mathcal{M}} = 100\% \times (Z_{\text{UB},\mathcal{M}} - Z_{\text{LB},\mathcal{M}})/Z_{\text{UB},\mathcal{M}}$, where $Z_{\text{UB},\mathcal{M}}$ and $Z_{\text{LB},\mathcal{M}}$ are the best upper and lower bounds, respectively, obtained at the termination of formulation $\mathcal{M}$ for the instance. We test all of the T1, T2 and T3 instances with $n = 100$, $|F| \in \{50, 100, 200, 500\}$ and $|\mathbf{leaves}(t)| \in \{8, 16, 32, 64\}$. Table 6 displays the average computation time and average optimality gap of each formulation for each combination of n, $|F|$ and $|\mathbf{leaves}(t)|$. For ease of notation in the table, we abbreviate SplitMIO as S-MIO and ProductMIO as P-MIO, and write $N_L \equiv |\mathbf{leaves}(t)|$.

From this table, we can see that for the smaller instances, SplitMIO requires more time to solve than ProductMIO. For larger instances, where the computation time limit is exhausted, the average gap obtained by ProductMIO tends to be lower than that of SplitMIO. This is congruent with our theoretical findings (specifically Proposition 3). Similarly to our results on formulation strength in Section 5.2, ProductMIO's edge over SplitMIO is most pronounced for the T1 instances, which exhibit the largest degree of product repetition among the splits compared to the T2 and T3 instances. For the T2 and T3 instances, which exhibit a lower degree of product repetition, the two formulations behave similarly, and the edge of ProductMIO is smaller; this also makes sense, because as we note in Section 5.2, ProductMIO and SplitMIO coincide when a tree is such that no product is repeated.

D.2. Comparison to Heuristic Approaches

In this experiment, we compare the performance of the two formulations to three different heuristic approaches:

1. LS: A local search heuristic, which starts from the empty assortment, and in each iteration moves to the neighboring assortment which improves the expected revenue the most. The neighborhood of assortments consists of those assortments obtained by adding a new product to the current assortment, or removing one of the existing products from the assortment. The heuristic terminates when there is no assortment in the neighborhood of the current one that provides an improvement.

2. LS10: This heuristic involves running LS from ten randomly chosen starting assortments. Each assortment is chosen uniformly at random from the set of 2^n possible assortments. After the ten repetitions, the assortment with the best expected revenue is retained.

Type	$\|F\|$	N_{L}	$O_{\text{S-MIO}}$	$O_{\text{P-MIO}}$	$T_{\text{S-MIO}}$	$T_{\text{P-MIO}}$	Type	$\|F\|$	N_{L}	$O_{\text{S-MIO}}$	$O_{\text{P-MIO}}$	$T_{\text{S-MIO}}$	$T_{\text{P-MIO}}$
T1	50	8	0.0	0.0	0.0	0.0	T2	200	8	0.0	0.0	0.6	0.6
T1	50	16	0.0	0.0	0.1	0.0	T2	200	16	0.0	0.0	747.7	673.6
T1	50	32	0.0	0.0	0.5	0.1	T2	200	32	9.8	9.7	3600.0	3600.0
T1	50	64	0.0	0.0	2.4	0.5	T2	200	64	17.1	16.3	3600.1	3600.0
T1	100	8	0.0	0.0	0.0	0.0	T2	500	8	0.0	0.0	171.6	167.9
T1	100	16	0.0	0.0	0.8	0.1	T2	500	16	14.1	13.8	3600.0	3600.0
T1	100	32	0.0	0.0	13.5	1.6	T2	500	32	23.4	23.0	3600.1	3600.1
T1	100	64	0.0	0.0	479.0	65.1	T2	500	64	28.7	28.1	3600.1	3600.1
T1	200	8	0.0	0.0	0.1	0.0	T3	50	8	0.0	0.0	0.0	0.0
T1	200	16	0.0	0.0	25.0	3.0	T3	50	16	0.0	0.0	0.1	0.1
T1	200	32	0.7	0.0	2330.2	421.5	T3	50	32	0.0	0.0	0.6	0.6
T1	200	64	9.8	5.3	3600.1	3600.0	T3	50	64	0.0	0.0	4.1	3.7
T1	500	8	0.0	0.0	4.6	1.4	T3	100	8	0.0	0.0	0.0	0.0
T1	500	16	5.0	1.2	3600.0	2951.7	T3	100	16	0.0	0.0	0.8	0.7
T1	500	32	15.9	11.8	3600.1	3600.0	T3	100	32	0.0	0.0	29.0	26.4
T1	500	64	20.9	17.4	3600.1	3600.1	T3	100	64	0.9	0.5	2600.8	2206.5
T2	50	8	0.0	0.0	0.0	0.0	T3	200	8	0.0	0.0	0.4	0.4
T2	50	16	0.0	0.0	0.1	0.1	T3	200	16	0.0	0.0	76.8	79.1
T2	50	32	0.0	0.0	0.5	0.5	T3	200	32	5.3	5.1	3600.0	3600.0
T2	50	64	0.0	0.0	24.0	24.6	T3	200	64	13.5	12.5	3600.1	3600.0
T2	100	8	0.0	0.0	0.0	0.0	T3	500	8	0.0	0.0	75.9	72.9
T2	100	16	0.0	0.0	1.5	1.4	T3	500	16	8.8	8.8	3600.0	3600.0
T2	100	32	0.0	0.0	245.2	234.9	T3	500	32	21.0	20.3	3600.1	3600.0
T2	100	64	4.8	4.4	3600.0	3600.0	T3	500	64	28.9	27.9	3600.2	3600.1

Table 6 Comparison of final optimality gaps and computation times for SplitMIO and ProductMIO.

3. ROA: This heuristic involves finding the optimal revenue ordered assortment. More formally, we define $S_k = \{i_1, \ldots, i_k\}$, where $i_1, \ldots, i_n$ corresponds to an ordering of the products so that $r_{i_1} \geq r_{i_2} \geq \cdots \geq r_{i_n}$, and we find $\arg\max_{S \in \{S_1, \ldots, S_n\}} R^{(F,\boldsymbol{\lambda})}(S)$.

We compare these heuristics against the best integer solution obtained by each of our two MIO formulations, leading to a total of five methods for each instance. We measure the performance of the solution corresponding to approach $\mathcal{M}$ using the metric $G_{\mathcal{M}}$, which is defined as $G_{\mathcal{M}} = 100\% \times (Z' - Z_{\mathcal{M}})/Z'$, where Z' is the highest lower bound (i.e., integer solution) obtained from among the two MIO formulations and the three heuristics, and $Z_{\mathcal{M}}$ is the objective value of the solution returned by approach $\mathcal{M}$.

Table 7 shows the performance of the five approaches – SplitMIO, ProductMIO, LS, LS10 and ROA – for each family of instances. The gaps are averaged over the twenty instances for each combination of instance type, $|F|$ and $|\mathbf{leaves}(t)|$. Again, for ease of notation in the table, we abbreviate SplitMIO as S-MIO and ProductMIO as P-MIO, and write $N_L \equiv |\mathbf{leaves}(t)|$. We can see from this table that in general, for the small instances, the solutions obtained by the MIO formulations are either the best or close to the best out of the five approaches, while the solutions produced by the heuristic approaches are quite suboptimal. For cases where the gap is zero for the MIO formulations, the gap of LS ranges from 1.5% to 13.9%; the LS10 heuristic improves on this, due to its use of restarting and randomization, but still does not perform as well as the MIO solutions (gaps ranging from 0.3 to 8.1%). For the larger instances, where the gap of the MIO solutions is larger, LS and LS10 still tend to perform worse. Across all of the instances, ROA achieves much higher gaps than all of the other approaches (ranging from 13.8 to 34.9%). Overall,

Type	$\|F\|$	N_L	$G_{\text{S-MIO}}$	$G_{\text{P-MIO}}$	G_{LS}	G_{LS10}	G_{ROA}	Type	$\|F\|$	N_L	$G_{\text{S-MIO}}$	$G_{\text{P-MIO}}$	G_{LS}	G_{LS10}	G_{ROA}
T1	50	8	0.0	0.0	8.1	2.1	26.8	T2	200	8	0.0	0.0	3.8	0.8	23.1
T1	50	16	0.0	0.0	9.8	4.0	31.2	T2	200	16	0.0	0.0	5.5	2.8	24.2
T1	50	32	0.0	0.0	9.0	5.2	27.5	T2	200	32	0.2	0.2	7.0	4.5	24.7
T1	50	64	0.0	0.0	10.0	7.2	28.6	T2	200	64	1.0	0.4	8.2	5.5	23.5
T1	100	8	0.0	0.0	3.9	2.4	26.0	T2	500	8	0.0	0.0	2.0	0.5	15.6
T1	100	16	0.0	0.0	6.6	3.7	26.2	T2	500	16	0.1	0.1	3.8	1.5	16.3
T1	100	32	0.0	0.0	8.5	6.3	25.6	T2	500	32	0.5	0.4	4.9	1.8	15.1
T1	100	64	0.0	0.0	9.9	6.6	24.5	T2	500	64	1.1	0.9	4.5	1.9	14.3
T1	200	8	0.0	0.0	3.5	1.3	19.9	T3	50	8	0.0	0.0	13.2	3.8	33.1
T1	200	16	0.0	0.0	5.5	3.1	21.7	T3	50	16	0.0	0.0	14.4	5.2	34.9
T1	200	32	0.0	0.0	7.8	4.6	22.3	T3	50	32	0.0	0.0	12.6	5.0	33.7
T1	200	64	0.3	0.0	7.6	6.3	19.5	T3	50	64	0.0	0.0	13.9	8.1	33.0
T1	500	8	0.0	0.0	1.5	0.3	14.6	T3	100	8	0.0	0.0	8.0	1.9	30.2
T1	500	16	0.0	0.0	3.7	1.6	15.3	T3	100	16	0.0	0.0	10.0	3.1	32.6
T1	500	32	0.3	0.0	5.3	2.4	15.9	T3	100	32	0.0	0.0	10.4	3.8	32.0
T1	500	64	0.6	0.1	4.9	3.5	13.8	T3	100	64	0.0	0.0	10.0	4.5	31.0
T2	50	8	0.0	0.0	13.8	3.2	31.5	T3	200	8	0.0	0.0	4.0	1.1	25.8
T2	50	16	0.0	0.0	11.6	4.9	32.2	T3	200	16	0.0	0.0	6.5	2.5	26.5
T2	50	32	0.0	0.0	10.0	5.8	31.1	T3	200	32	0.1	0.1	7.6	3.3	26.9
T2	50	64	0.0	0.0	11.6	7.0	30.4	T3	200	64	0.3	0.1	9.2	4.1	24.1
T2	100	8	0.0	0.0	5.5	1.9	28.1	T3	500	8	0.0	0.0	2.6	0.4	15.8
T2	100	16	0.0	0.0	8.2	4.0	30.8	T3	500	16	0.0	0.0	4.3	1.1	16.5
T2	100	32	0.0	0.0	8.9	4.8	31.3	T3	500	32	0.5	0.5	5.3	1.3	16.0
T2	100	64	0.0	0.0	11.6	6.6	27.1	T3	500	64	0.9	0.4	5.5	1.3	16.3

Table 7 Comparison of SplitMIO and ProductMIO against the heuristics LS, LS10 and ROA.

these results suggest that *the very general structure of the decision forest model poses significant difficulty to standard heuristic approaches*, and highlight the value of using exact approaches over inexact/heuristic approaches to the assortment optimization problem.

D.3. Value of the Greedy-Based Benders Algorithm

In this subsection, we present results from a computational experiment to showcase the value of the greedy-based Benders approach for solving the linear relaxation of SplitMIO. We consider the same T3 instances from Section 5.1.

For each instance, we solve the linear relaxation of SplitMIO using the greedy-based Benders approach, where we solve the master problem using constraint generation, and the primal and dual subproblems are solved using Algorithms 1 and 2, respectively. We denote this solution method by `SG`. We also solve the linear relaxation of SplitMIO using constraint generation, where we solve the dual subproblem (11) directly using Gurobi, without using our greedy algorithms. We denote this solution method by `SD`. Lastly, we also solve the linear relaxation of ProductMIO using constraint generation, where again we solve the dual subproblem (15) in that formulation directly using Gurobi. We denote this solution method by `PD`. We impose a time limit of 2 hours on all three solution methods.

For each instance and for each method m we record the solution time, T_m. We additionally compute R_m as the reduction in computation time of using `SG` relative to method m; mathematically, it is defined as

$$R_m = 100\% \times (T_m - T_{\texttt{SG}}) / T_m. \tag{64}$$

n	b	$T_{\texttt{SG}}$	$T_{\texttt{SD}}$	$T_{\texttt{PD}}$	$R_{\texttt{SD}}$	$R_{\texttt{PD}}$	H
200	4	11.6	134.5	107.2	91.4	89.2	0.3
200	8	11.7	119.9	107.9	90.3	89.2	0.6
200	12	12.0	121.9	110.5	90.1	89.1	0.8
200	16	12.0	125.1	110.2	90.4	89.1	0.8
200	20	12.4	120.5	106.3	89.7	88.4	0.7
200	24	11.8	115.8	104.6	89.8	88.7	0.7
500	10	9.9	110.0	111.7	91.0	91.0	0.5
500	20	20.8	182.4	189.0	88.6	88.9	0.4
500	30	23.0	189.9	193.4	87.9	88.1	0.4
500	40	23.2	186.8	190.4	87.6	87.8	0.4
500	50	24.7	192.8	194.3	87.2	87.2	0.3
500	60	25.1	184.4	189.5	86.4	86.8	0.3
1000	20	11.4	121.3	170.7	90.6	93.4	0.1
1000	40	44.2	259.7	345.4	83.1	87.3	0.1
1000	60	55.5	285.4	412.2	80.6	86.6	0.2
1000	80	61.9	295.0	412.0	79.0	85.0	0.2
1000	100	70.2	303.0	415.4	76.8	83.1	0.2
1000	120	78.5	304.7	408.1	74.3	80.8	0.2
2000	40	11.0	144.3	239.5	92.3	95.4	0.0
2000	80	36.8	204.1	439.8	82.0	91.7	0.0
2000	120	122.5	399.3	746.4	69.7	83.8	0.1
2000	160	196.2	507.6	950.8	61.6	79.5	0.0
2000	200	288.9	627.3	1096.7	53.9	73.7	0.1
2000	240	385.0	741.5	1249.3	48.2	69.3	0.1
3000	60	11.6	163.2	361.6	92.9	96.7	0.0
3000	120	64.1	253.4	658.5	75.7	90.7	0.0
3000	180	299.4	632.2	1391.3	52.8	78.6	0.0
3000	240	619.7	1025.3	2079.3	39.6	70.2	0.0
3000	300	1009.8	1489.2	2609.9	32.2	61.3	0.0
3000	360	1524.0	2011.5	3258.1	24.2	53.2	0.0

Table 8 Comparison of the solution methods for the SplitMIO and ProductMIO relaxations.

We compute this metric for m equal to either SD or PD. Lastly, we also compute the relative gap, H, of the SplitMIO relaxation relative to the ProductMIO relaxation, as

$$H = 100\% \times (Z_{\texttt{SG}} - Z_{\texttt{PD}}) / Z_{\texttt{PD}}, \tag{65}$$

where $Z_{\texttt{SG}}$ and $Z_{\texttt{PD}}$ are the objective values from the SG and PD approaches, respectively. Recall that the bound from ProductMIO is always at least as tight as that of SplitMIO, and hence this H will always be a positive percentage.

In Table 8, we report the runtime T_m for $m \in \{\texttt{SG}, \texttt{SD}, \texttt{PD}\}$, along with the reduction in computation time of SG relative to SD and PD, denoted as $R_{\texttt{SD}}$ and $R_{\texttt{PD}}$, respectively. Additionally, we present the relative gap H of the SplitMIO relaxation relative to the ProductMIO relaxation. From this table, we obtain two important insights. The first is that SG is always faster than SD for solving the SplitMIO relaxation across all of the combinations of n and b. Comparing SG and SD, the reduction in solution time can be quite large in some cases. For example, for $n = 2000$ and $b = 40$, the average solution time for SG is 11.0 seconds, while for SD it is 144.3 seconds, which corresponds to a reduction of $R_{\texttt{SD}} = 92.3\%$. Thus, even focusing on SplitMIO, our proposed greedy-based method can be of significant utility in being able to solve the relaxation of this problem quickly at scale.

The second insight is that SG solves its relaxation faster than PD across all values of n and b. Interestingly, PD takes longer to solve than SD, and the reduction in computation time achieved by SG relative to PD is even more pronounced. For example, in the same instances where $n = 2000$ and $b = 40$, SG requires on average 11.0 seconds, whereas PD requires on average 239.5 seconds, which corresponds to a reduction in computation of $R_{\text{PD}} = 95.4\%$.

With regard to this second insight, a natural question is how the objective values of relaxed SplitMIO and ProductMIO compare for these instances: it could be the case that PD produces an appreciably tighter bound than SG, and hence the increase in computation time associated with PD could potentially be worth it. However, this turns out to not be the case. In general, the average value of H across all n and b values is no more than 0.8%. This is consistent with our findings in Section 5.2, where we show that the reduction in the integrality gap of ProductMIO is largest for the T1 instances, which exhibit a huge degree of product repetition across splits, and smallest for the T3 instances, where there is a smaller degree of repetition and a higher degree of flexibility in the tree structure.

Overall, these results highlight two synergistic aspects of the SplitMIO formulation: (1) for the same problem instance, the greedy Benders approach for solving the linear relaxation of SplitMIO is in general faster than the traditional "direct" Benders approaches for solving the linear relaxations of SplitMIO and ProductMIO; and (2) while SplitMIO is in theory not as tight as ProductMIO, the loss in bound quality is negligible in large-scale, randomly generated instances that do not exhibit a high degree of structure.

Appendix E: Additional Numerical Results Based on the Sushi Dataset

In this section, we include an additional numerical study to complement the results in Section 6. Specifically, we consider using the Sushi dataset (Kamishima 2003). Different from the IRI Dataset in Section 6, which is composed on real-world transaction records, the Sushi dataset consists of respondents' preferences over ten popular types of sushi. Consequently, the dataset immediately implies a ground-truth choice model based on rankings. Since the ranking-based model is subsumed by the decision forest model, this dataset enables us to understand how overfitting might degrade the performance of the decision forest model in terms of revenue performance.

E.1. Data Processing

Our data preprocessing process closely follows Berbeglia et al. (2022). The dataset consists of the sushi preferences for 5,000 respondents. In the survey, each respondent was shown $n = 10$ type of sushi: *ebi* (shrimp), *anago* (sea eel), *maguro* (tuna), *ika* (squid), *uni* (sean urchin), *ikura* (salmon roe), *tamago* (egg), *toro* (fatty tuna), *tekka maki* (tuna roll), and *kappa maki* (cucumber roll). The respondent then was asked to rank these ten sushi types. Note that in this experiment, the respondents were not asked to rank the no-purchase option among the sushi types. We therefore consider a setting which Berbeglia et al. (2022) referred as *Top 3*; see Section 4.2 of Berbeglia et al. (2022). In this setting, each respondent's ranking is trimmed to have only length three. In other words, when offered an assortment, each respondent would only buy a sushi if one of her top 3 sushi types is in the assortment. We believe that, in the context of discrete choice modeling, this is a more realistic setting than the untrimmed setting. Note that in the latter setting, the resulting no-purchase

probability stays the same for all offered assortments, i.e., $P(0 \mid S)$ is constant for all assortments $S \subseteq N$, which is unrealistic. We use $\sigma_1, \ldots, \sigma_{5000}$ to denote the resulting trimmed rankings.

We follow Berbeglia et al. (2022) to generate semisynthetic datasets via these trimmed rankings. For each transaction τ, we sample an assortment S_τ from the collection $\mathcal{S}_{\text{sample}} = \{S \subseteq N \mid 3 \leq |S| \leq 6\}$ uniformly at random. Next, with probability 1/2, we choose a trimmed ranking σ uniformly at randomly from $\{\sigma_1, \ldots, \sigma_{5000}\}$, and let $o_\tau \in N_+$ be the purchase decision of σ under assortment S_τ. With the remaining 1/2 probability, o_τ is simply no-purchase option, i.e., $o_\tau = 0$. In other words, the ground-truth choice model, denoted as `GT`, is a ranking-based model composed of 5001 rankings, where each σ_t is associated with weight $\lambda_t = 0.0001$, for $t = 1, \ldots, 5000$, and the 5001st ranking, which has weight $\lambda_{50001} = 0.5$, prefers the no-purchase option to all sushi types. We generate 6400 transactions and save these transactions $\mathcal{T}_1 = \{(S_\tau, o_\tau)\}_{\tau=1,\ldots,6400}$ as a dataset. We repeat the data generating process four more times and obtain $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$, and $\mathcal{T}_5$.

Before we present the numerical results, we comment on the difference between the setting in this section and the setting presented in Section 6. We argue that the juxtaposition of the two numerical studies can help us assess the optimal assortments returned by the decision forest model from two different angles. Recall that in Section 6, the IRI dataset offers real-world transaction data, which enable us to discuss how real consumer behavior and its inherit bias, such as the choice overload effect, influence the assortment decision; see Section 6.3. Meanwhile, the IRI dataset does not offer a ground truth model. Consequently, we were not able to *absolutely* evaluate the performance of the assortment S^{DF} returned by the decision forest model. The best we can do is to evaluate it *relatively*, i.e., with respect to another estimated choice model; see Table 3. In contrast, the Sushi dataset offers the ground truth choice model `GT`, which is a ranking-based model. This ground truth model enables us to evaluate the expected revenue of any assortment *absolutely* and measure the optimality gap. Meanwhile, since the Sushi dataset restricts the respondents to make a complete preference over sushi types, the resulting synthetic data eventually have to satisfy the regularity property (see Section 6.3) and thus rinse off any possible non-rational consumer choice.

E.2. Model Estimation and Performance Metrics

We follow Section 6 to estimate a decision forest model and a ranking-based model from dataset $\mathcal{T}_i$, $i = 1, \ldots, 5$. Note that the estimated ranking-based model from dataset $\mathcal{T}_i$ is not necessarily same as the ground truth model.

For the decision forest model, we again set the depth limit of the estimated trees as three. Similarly, we further restrict the length of the estimated rankings as three. In other words, each ranking can have at most three products preferred to the no-purchase option. We impose such length condition by including an additional constraint in the subproblem when using the column generation approach to estimate the ranking-based model from data (van Ryzin and Vulcano 2014)[1]. Observe that in this setting, the class of ranking-based models that we will learn from data exactly includes the ground truth model `GT`. Meanwhile, the class of decision forest models strictly subsumes the ground truth model `GT`, as the latter only consists of rankings of length three. This setting enables us to examine how overfitting, a potential byproduct of the decision forest model's universal representational power, might impair revenue performance.

[1] For example, adding a constraint $\sum_{j=1}^{n} x_{j0} \leq 3$ to Problem (12) of van Ryzin and Vulcano (2014).

To assess the performance of the learning results, we define two metrics. The first metric is the L^1-norm between a learned choice model $\mathcal{M}$ and the ground truth model $\texttt{GT}$. Specifically, it is defined as

$$L^1_{\mathcal{M}} = \frac{1}{2^n} \sum_{S \subseteq N} \left\| P_{\mathcal{M}}(\cdot \mid S) - P_{\texttt{GT}}(\cdot \mid S) \right\|_1, \tag{66}$$

which is the average of L^1 norm between the choice vector of model $\mathcal{M}$ and that of the ground truth $\texttt{GT}$ over all possible assortments. The metric measures how close the learned choice model $\mathcal{M}$ is to the ground truth model $\texttt{GT}$ in the space of choice probability. We remark that one can consider replacing the L^1 norm in the summation by another "distance" metric such as the L^2 norm or the KL divergence. Here we use the L^1 norm since it measures the deviation in a linear sense, making the L^1 norm more relevant to the assortment tasks. The second metric is the optimality gap of the optimal assortment $S^{\mathcal{M}}$ returned by the learned choice model $\mathcal{M}$. Let Z^* be the maximal revenue under the ground truth model $\texttt{GT}$ and $Z^{\mathcal{M}}$ be the expected revenue of assortment $S^{\mathcal{M}}$ under the ground truth model. The optimality gap is thus defined as

$$G_{\mathcal{M}} = 100\% \times \left(Z^* - Z^{\mathcal{M}} \right) / Z^*. \tag{67}$$

We remark that the price information of each sushi type needed to calculate the expected revenue is provided in the dataset.

To study the overfitting and its impact on assortment performance, we use subsamples to learn both the ranking-based model and the decision forest model. For each subsample size $n_{\mathcal{T}} \in \{200, 400, 800, 1600, 3200, 6400\}$, we use the first $n_{\mathcal{T}}$ transactions, i.e., $\{(S_\tau, o_\tau)_{\tau=1,\ldots,n_{\mathcal{T}}}\}$ to learn the ranking-based model and the decision forest model. We then measure the learned models by the two metrics. We repeat the aforementioned task for each dataset $\mathcal{T}_1, \ldots, \mathcal{T}_5$, and present the average result in Figure 11.

E.3. Results

Figure 11 compares the ranking-based model and the decision forest model based on the two metrics: the L^1 norm (left panel) and the optimality gap (right panel). In both panels, the x-axis is displayed on a logarithmic scale, with error bars representing standard deviations. The left panel focuses on how well each model recovers the ground truth as the sample size, $n_{\mathcal{T}}$, increases. The ranking-based model is shown in blue, and the decision forest model in red. Although the decision forest model has the ultimate representation power to fit the data, it struggles with overfitting when the sample size is small. For instance, with 800 transactions (approximately one transaction per assortment in $\mathcal{S}_{\text{sample}}$), its distance to the ground truth, as measured by the L^1 norm, is nearly double that of the estimated ranking-based model. However, as the dataset grows larger – such as when $n_{\mathcal{T}} = 6400$ (roughly eight transactions per assortment) – the overfitting issue diminishes, and the decision forest model's performance becomes comparable to that of the ranking-based model.

While overfitting can significantly hamper the decision forest model's ability to recover the ground truth, this does *not* translate into substantial losses when evaluating its performance based on the optimality gap of the assortment S^{DF} it returns. For instance, at $n_{\mathcal{T}} = 200$ and $n_{\mathcal{T}} = 800$, although the decision forest model deviates from the ground truth by nearly 100% more than the ranking-based model, the optimality gap of S^{DF} is only 10% and 30% higher than that of S^{RM}, the assortment generated by the ranking-based model.

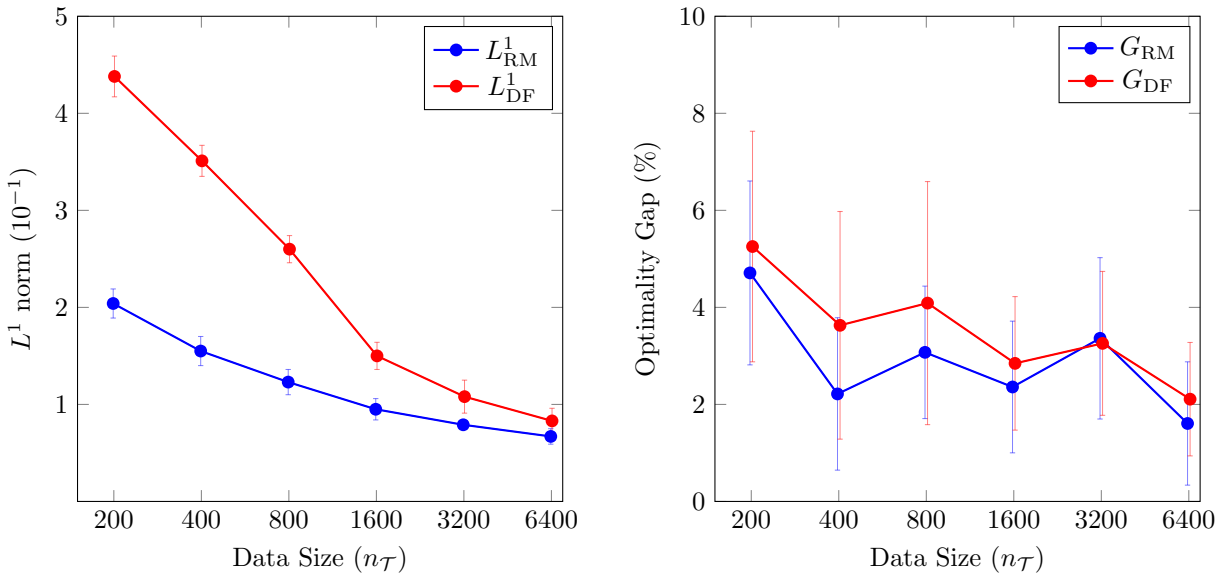


Figure 11 **The performance of the ranking-based model and the decision forest model measured by (i) the L^1-norm distance to the ground truth (left panel); and (ii) the optimality gap of the assortments they return (right panel).**

As the data size increases to $n_{\mathcal{T}} = 1600$, the two assortments begin to perform closely, with minor difference in their optimality gaps under relatively large standard deviations. By $n_{\mathcal{T}} = 6400$, both assortments are nearly optimal, with an optimality gap of just 2%. Interestingly, Figure 11 highlights that the operational performance and predictive accuracy of choice models can follow distinct patterns. Notably, the standard errors in the assortment tasks are significantly larger than those observed in the prediction tasks.

The key takeaway from this additional analysis using the Sushi Dataset is that even in the most extreme scenario, where the ground truth model consists solely of rankings and lacks any non-rational consumer choices, the decision forest model's operational performance might be only slightly inferior to that of the ranking-based model.