Electronic Companion: Assortment Optimization under the Decision Forest Model

(Yi-Chun Akchen and Velibor V. Mišić)

This electronic companion covers Sections A–E, which contain additional numerical analyses and implementation details. Theoretical results, including omitted proofs (Section F) and further analysis (Section G), are not included in this electronic companion due to page limits. These materials are instead available at `https://arxiv.org/abs/2103.14067`.

Appendix A: Solving the SplitMIO Relaxation via a Greedy Algorithm

As noted in Section 4.1, each primal subproblem (9) of the SplitMIO relaxation can be solved using a greedy algorithm. While the algorithm was briefly outlined in Section 4.1, we provide additional implementation details here, particularly regarding the events A_s, B_s, and C, through the following pseudocode (Algorthm 2).

First, we define an ordering τ of the leaves in nondecreasing revenue. In particular, we require a bijection $\tau : \{1, \ldots, |\mathbf{leaves}(t)|\} \to \mathbf{leaves}(t)$ such that $r_{t,\tau(1)} \geq r_{t,\tau(2)} \geq \cdots \geq r_{t,\tau(|\mathbf{leaves}(t)|)}$, i.e., an ordering of leaves in nondecreasing revenue. In addition, in the definition of Algorithm 2, 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 $\ell \in \mathbf{right}(s)$.

The algorithm processes the leaves in decreasing order of revenue and sets the variable $y_{t,\ell}$ for each leaf ℓ to the largest value that does not violate the left and right split constraints (9c) and (9d), nor the constraint $\sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} \leq 1$. At each iteration, the algorithm also records which constraint becomes tight by maintaining the event set $\mathcal{E}$. An A_s event indicates that the left split constraint (9c) for split s has become tight; a B_s event indicates that the right split constraint (9d) for split s has become tight; and a C event indicates that the constraint $\sum_{\ell \in \mathbf{leaves}(t)} y_{t,\ell} \leq 1$ has become tight. If a C event is not triggered, Algorithm 2 identifies the split with the smallest remaining capacity (line 15). If the arg min is not unique and multiple splits are tied, ties are broken by selecting the split s with the smallest depth $d(s)$ (i.e., the split closest to the root node of the tree).

The function f records which leaf ℓ is being processed when an A_s, B_s, or C event occurs. Although neither $\mathcal{E}$ nor f is required to compute the primal solution, they are essential for determining the dual solution in the dual procedure (Algorithm 1).

Appendix B: Example of Benders algorithms for SplitMIO

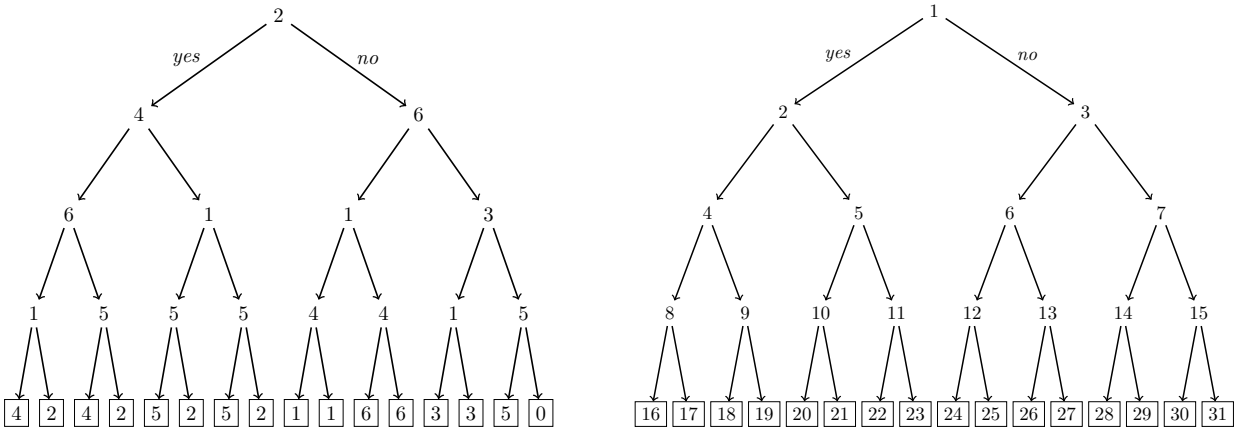
In this section, we provide a small example of the primal-dual procedure (Algorithms 2 and 1) for solving the SplitMIO subproblem. Suppose that $n = 6$, and that $\mathbf{x} = (x_1, \ldots, x_6) = (0.62, 0.45, 0.32, 0.86, 0.05, 0.35)$. Suppose that $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_6) = (97, 72, 89, 50, 100, 68)$. Suppose that the purchase decision tree t has the form

Algorithm 2 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 $f(e) = \tau(i)$.

given in Figure 4a; in addition, suppose that the splits and leaves are indexed as in Figure 4b. 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.



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

Figure 4 Tree used in example of SplitMIO primal-dual algorithms.

We first run Algorithm 2 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:

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

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 2).

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 1).

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 $f : \mathcal{E} \to \textbf{leaves}$ 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 1, 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.

The feasibility of the dual solution is visualized in Figure 5. 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 (10b)). 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 $\sum_{\ell \in \textbf{leaves}} r_\ell \cdot y_\ell = r_{20} \times y_{20} + r_{22} \times y_{22} + r_{30} \times y_{30} + r_{24} \times y_{24} + r_{28} \times y_{28} + r_{17} \times y_{17} = 87.5$. The objective value of the dual solution is $\gamma + (1 - x_2) \times \beta_1 + x_6 \times \alpha_3 + x_5 \times \alpha_{10} + x_5 \times \alpha_{11} + x_5 \times \alpha_{15} = 72.0 + 0.55 \times 17 + 0.35 \times 8 + 0.05 \times 28 + 0.05 \times 28 + 0.05 \times 11 = 87.5$.

Appendix C: Benders subproblem of the ProductMIO relaxation

We continue the discussion in Section 4.2 and show that Problem (12) cannot be solved with a greedy approach. We formalize this claim as the following proposition.

PROPOSITION 3. *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* (12) *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

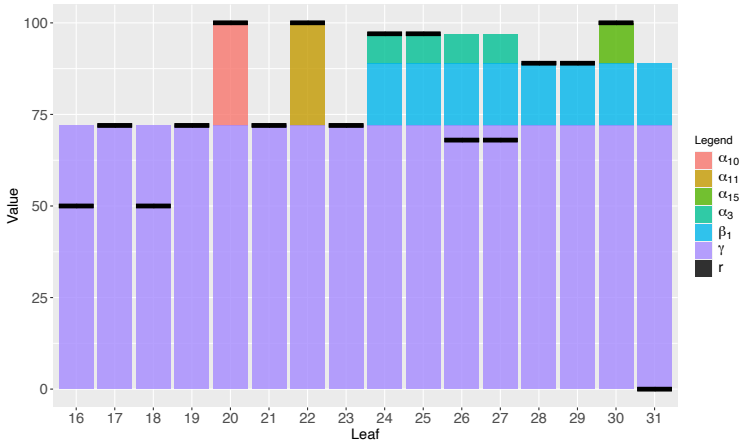


Figure 5 **Visualization of feasibility of dual solution.**

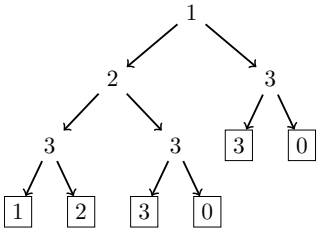


Figure 6 **Structure of tree for which the ProductMIO primal subproblem is not solvable via a greedy algorithm.**

shown in Figure 6. 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. Further Discussion on the T1, T2, and T3 instances

In Section D.1, we introduced the T1, T2, and T3 instances, which represent decision trees with distinct topologies. In this section, we provide additional discussion and empirical evidence related to these instances.

First, the trees in the T1 instances are highly symmetrical and structured. Thus, they are unlikely to arise in decision forest models estimated from real-world data. By contrast, trees in both the T2 and T3 instances trees are commonly observed in estimated decision forest models. Their difference lies in whether the trees are balanced; however, as shown in Table 1, SplitMIO and ProductMIO perform comparably on these instances, with ProductMIO exhibiting modest advantages in terms of integrality gap, optimality gap, and runtime.

In what follows, we use the IRI Dataset to illustrate the types of tree structures that actually appear in estimated decision forest models. We first define two indices for a given set of estimated trees F in the decision forest model. The first, I_{FU} (Fraction of Unbalanced Trees), is defined as

$$I_{\text{FU}} = \sum_{t \in F} \mathbb{I}\left[\text{tree } t \text{ is unbalanced}\right] / |F|,$$

which measures the fraction of trees in F that are unbalanced. The second, I_{LU} (Level of Unbalancedness), quantifies how unbalanced the leaf depths are, averaged across trees. For a leaf ℓ in tree t, let $\text{depth}(\ell)$ denote its depth. Recall from Section 4.1 that the root split node is defined to be at depth 1. Consequently, $\text{depth}(\ell) \geq 2$ for all $\ell \in \textbf{leaves}(t)$. With a slight abuse of notation, define $d^t_{\max} \equiv \max\{\text{depth}(\ell) \mid \ell \in \textbf{leaves}(t)\}$ and $d^t_{\min} \equiv \min\{\text{depth}(\ell) \mid \ell \in \textbf{leaves}(t)\}$ as the maximum and minimal leaf depths of tree t, respectively. The level of unbalancedness of tree t is then

$$\text{LU}(t) = \frac{d^t_{\max} - d^t_{\min}}{d^t_{\max} - 2},$$

which takes value in $[0, 1]$. By construction, $\text{LU}(t) = 0$ if and only if tree t is perfectly balanced, while $\text{LU}(t) = 1$ for maximally unbalanced trees (for example, a tree whose split nodes form a single chain; note that this class of trees includes rankings). We then define the overall index for a collection of trees as

$$I_{\text{LU}} = \frac{1}{|F|} \sum_{t \in F} \text{LU}(t) = \frac{1}{|F|} \sum_{t \in F} \frac{d^t_{\max} - d^t_{\min}}{d^t_{\max} - 2}.$$

Table 6 reports the two indices for decision forest models estimated from each product category of the IRI Dataset. Both metrics show that the estimated trees are predominantly unbalanced. In particular, I_{FU} indicates that up to 91% of trees are unbalanced in some categories, and the overall average is 83% across all categories. These results suggest that the trees we observe in practice resemble the T3 instances much more closely than the T2 instances. Importantly, we did not observe T1-type trees in these estimated decision forest models, supporting our point that the T1 instances in Section 5.1 were constructed more for theoretical illustration rather than practical relevance.

D.2. 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 $O_{\mathcal{M}} = 100\% \times \left(Z_{\text{UB},\mathcal{M}} - Z_{\text{LB},\mathcal{M}}\right) / Z_{\text{UB},\mathcal{M}}$, where $Z_{\text{UB},\mathcal{M}}$ and $Z_{\text{LB},\mathcal{M}}$ are the best upper and lower bounds, respectively,

Category	I_{FU}	I_{LU}	Category	I_{FU}	I_{LU}
Beer	0.46	0.41	Mayonnaise	0.82	0.78
Blades	0.82	0.64	Milk	0.87	0.77
Carbonated Beverages	0.86	0.82	Mustard / Ketchup	0.91	0.84
Cigarettes	0.88	0.82	Paper Towels	0.86	0.85
Coffee	0.86	0.74	Peanut Butter	0.90	0.87
Cold Cereal	0.88	0.86	Photo	0.68	0.65
Deodorant	0.86	0.74	Salty Snacks	0.79	0.64
Diapers	0.88	0.71	Shampoo	0.57	0.40
Facial Tissue	0.82	0.80	Soup	0.86	0.80
Frozen Dinners	0.87	0.75	Spaghetti / Italian Sauce	0.85	0.51
Frozen Pizza	0.75	0.65	Sugar Substitutes	0.89	0.80
Household Cleaners	0.88	0.87	Toilet Tissue	0.90	0.88
Hotdogs	0.60	0.55	Toothbrush	0.9	0.90
Laundry Detergent	0.84	0.79	Toothpaste	0.91	0.86
Margarine / Butter	0.87	0.81	Yogurt	0.89	0.75
			Average over all categories	0.83	0.74

Table 6 The fraction of unbalanced trees and level of unbalancedness of the estimated decision forest model for each product category of the IRI Academic Dataset

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 7 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 in this set of experiments, 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 2). 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.

We note that Gurobi is able to process the full formulations of both SplitMIO and ProductMIO for all instances in this experiment, though the larger instances are not solved to optimality within the one-hour time limit.

D.3. 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.

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

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.

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 8 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

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

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, 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.4. Value of the Relaxation Phase in the SplitMIO Benders Decomposition

In this subsection, we present results from a computational experiment designed to understand the value of solving the linear relaxation of the SplitMIO master problem (11) before solving its integer version. We compare the performance of (i) our original Benders approach, which includes both the relaxation phase (phase 1) and the integer phase (phase 2) (see Section 4.4), and (ii) an alternative Benders approach that consists only of phase 2. In the latter approach, we solve the SplitMIO master problem (13) using lazy constraint generation to separate constraints (13b) at integer solutions, without initializing the problem with cuts obtained from solving the LP relaxation.

We denote the original Benders approach (with phases 1 and 2) by the subscript "Benders", and the alternative version that only executes phase 2 by "Benders-2-only." To ensure a fair comparison, we impose a two-hour time limit on the Benders-2-only approach, matching the total time allocated to the original Benders approach (one hour for phase 1 and one hour for phase 2). Table 9 reports the final optimality gap metric, $O_{\mathcal{M}} = (Z_{\text{UB},\mathcal{M}} - Z_{\text{LB},\mathcal{M}})/Z_{\text{UB},\mathcal{M}} \times 100\%$, where $Z_{\text{UB},\mathcal{M}}$ and $Z_{\text{LB},\mathcal{M}}$ are the best upper and lower

n	ρ	b	O_{Benders}	$O_{\text{Benders-2-only}}$
200	0.02	4	0.00	0.00
200	0.04	8	0.00	0.00
200	0.06	12	8.23	5.28
200	0.08	16	17.38	17.46
200	0.10	20	22.13	24.16
200	0.12	24	27.27	28.82
500	0.02	10	0.00	0.00
500	0.04	20	0.48	1.66
500	0.06	30	8.10	12.15
500	0.08	40	14.26	19.09
500	0.10	50	18.26	25.84
500	0.12	60	22.94	29.13
1000	0.02	20	0.00	0.00
1000	0.04	40	1.05	6.80
1000	0.06	60	6.16	13.98
1000	0.08	80	10.07	17.77
1000	0.10	100	13.60	20.78
1000	0.12	120	15.77	25.98
2000	0.02	40	0.00	0.20
2000	0.04	80	0.21	6.43
2000	0.06	120	2.42	9.70
2000	0.08	160	5.21	13.30
2000	0.10	200	7.16	15.83
2000	0.12	240	11.91	17.01
3000	0.02	60	0.00	0.46
3000	0.04	120	0.08	6.22
3000	0.06	180	1.81	10.44
3000	0.08	240	4.23	14.84
3000	0.10	300	6.17	14.07
3000	0.12	360	9.83	34.78

Table 9 Comparison of the full Benders approach (phases 1 and 2) with the phase-2-only Benders approach.

bounds, respectively, obtained from either the original Benders approach ($\mathcal{M} = \text{Benders}$) or the phase-2-only approach ($\mathcal{M} = \text{Benders-2-only}$) after the computation time limit is exhausted. The same instances from Table 2 are used for consistency.

Table 9 summarizes the results and shows that the relaxation phase substantially improves performance across all instances. The reduction in optimality gap is particularly pronounced for larger problems ($n = 2000$ or 3000), where the gap is reduced by roughly half—or even more. For example, for the $(n, \rho) = (2000, 0.06)$ instances, the original Benders approach achieves an average gap of about 2.5%, while the phase-2-only approach yields a gap of roughly 10%. Similarly, for $(n, \rho) = (3000, 0.12)$, the original Benders approach reaches an optimality gap of around 10%, compared to approximately 35% for the phase-2-only approach. This improvement occurs because the cuts from phase 1 provide a tighter upper bound and a more accurate piecewise-linear approximation of the functions $G_t(\cdot)_{t \in F}$, enabling more effective pruning in the branch-and-bound process and resulting in better bounds within the limited computation time. In addition to the gap performance, we also observed that the integer solutions obtained by the original Benders approach are generally better than those from the phase-2-only approach for the larger instances.

D.5. 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 2 and 1, 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 (10) 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 (14) 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

$$R_m = 100\% \times (T_m - T_{\texttt{SG}}) / T_m. \tag{19}$$

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

$$H = 100\% \times (Z_{\texttt{SG}} - Z_{\texttt{PD}}) / Z_{\texttt{PD}}, \tag{20}$$

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 10, 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 $R_{\texttt{PD}} = 95.4\%$.

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

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 10 Comparison of the solution methods for the SplitMIO and ProductMIO relaxations.

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 evaluate the end-to-end performance of the decision forest model in a data-to-decision setting, using a semi-synthetic experiment based on the Sushi dataset (Kamishima 2003). Our results demonstrate that when the data exhibits non-rational choice behavior that violates the regularity property, the decision forest model can yield better assortment decisions than rational choice models.

E.1. Data Processing

Our data preprocessing process closely follows Berbeglia et al. (2022). The Sushi 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 use $\sigma_1, \dots, \sigma_{5000}$ to denote the resulting trimmed rankings. We assume each ranking σ_k, $k = 1, \dots, 5000$, represent a customer type in the market with probability weight $\lambda_k = 1/5000$. Note that each ranking $\sigma_k : \mathcal{N} \cup \{0\} \to \mathcal{N} \cup \{0\}$ is a bijection that assigns each option to a rank. Particularly, we use $\sigma_k(i)$ to denote the rank of option $i \in \mathcal{N} \cup \{0\}$, and $\sigma_k(i) < \sigma_k(j)$ indicates that i is more preferred to j under ranking σ_k. We use $\sigma_k[S]$ to denote the most preferred option when assortment S is offered, i.e., $\sigma_k[S] \equiv \arg\min_{i \in S \cup \{0\}} \sigma_k(i)$.

E.2. Ground-Truth Choice Model with Choice Overload

We study the performance of the decision forest model in a data-to-decision setting where customer choices may deviate from the regularity property. To this end, we construct a ground-truth choice model using rankings from the Sushi dataset, augmented with a behavioral component that captures choice overload (see Section 6.3 for an overview). The idea is that as the assortment size increases, customers experience greater mental fatigue when evaluating options, making them more likely to leave without making a purchase.

For each $k \in \{1, \dots, 5000\}$, let $\tilde{\sigma}_k[S]$ denote the purchase decision of customer type k facing assortment S. We assume

$$\tilde{\sigma}_k[S] = \begin{cases} \sigma_k[S], & \text{with probability } 1 - \pi(|S|), \\ 0, & \text{with probability } \pi(|S|), \end{cases}$$

where $\pi(|S|)$ is the baseline no-purchase probability. If $\pi(|S|)$ is fixed – e.g., following Berbeglia et al. (2022), where $\pi(|S|) = 0.5$ for all S – the resulting ground-truth choice model

$$P_{\texttt{GT}}(i \mid S) = \sum_{i=1}^{5000} \frac{1}{5000} \cdot P\left(\tilde{\sigma}_k[S] = i\right) \tag{21}$$

is a ranking-based model and thus consistent with the regularity property. However, if $\pi(|S|)$ increases with assortment size, the model may violate regularity and enter a non-rational choice regime.

In our experiment, we let the baseline no-purchase probability grow linearly with the assortment size:

$$\pi(|S|) = 0.5 + \alpha_{\text{OL}} \cdot (|S| - 3)_+, \tag{22}$$

where $(x)_+ = \max\{x, 0\}$ and α_{OL} measures the intensity of choice overload. When the assortment has three or fewer products, customers can still easily evaluate all options, form preferences, and make decisions without significant fatigue. Beyond this threshold, larger assortments increase cognitive load, making customers more likely to opt out of purchasing. Specifically, each additional product raises the no-purchase probability by α_{OL}. Finally, the constant 0.5 in (22) has no impact on the optimal assortment decision, as it uniformly scales all choice probabilities. We retain it so that when $\alpha_{\text{OL}} = 0$, our setting coincides with that of Berbeglia et al. (2022).

E.3. Sample Transactions using the Ground-Truth Choice Model

We generate transaction data from the ground-truth model as follows. First, we sample a set of historical assortments $\mathcal{S}_{\text{sample}} = \{S_1, \ldots, S_{n_{\text{asst}}}\}$ uniformly without replacement from the set of all possible assortments. At each time period $\tau = 1, \ldots, n_{\text{tran}}$, an assortment S_τ is drawn uniformly at random from $\mathcal{S}_{\text{sample}}$, and a customer makes a purchase decision $o_\tau \in S_\tau \cup 0$. Specifically, the arriving customer is assumed to belong to type k with probability $1/5000$ and makes a choice according to $\tilde{\sigma}_k$. The purchase decision is then sampled as $o_\tau \sim \tilde{\sigma}_k[S_\tau]$. The resulting set of transactions is $\mathcal{T}_1 \equiv \{(S_\tau, o_\tau)_{\tau=1,\ldots,n_{\text{tran}}}\}$.

We set $n_{\text{asst}} = 150$, which is roughly one-seventh of the total number of possible assortments ($2^n = 1024$). This design ensures that solving the data-to-decision problem requires the choice model to generalize effectively to unobserved assortments outside $\mathcal{S}_{\text{sample}}$. We also set $n_{\text{tran}} = 5000$, corresponding to five thousand transactions. To obtain multiple datasets, we repeat this data-generation procedure nine additional times, yielding ten transaction sets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{10}$.

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 numerical study in this section is built around a ground-truth choice model $P_{\texttt{GT}}(\cdot \mid \cdot)$. Having access to $P_{\texttt{GT}}(\cdot \mid \cdot)$ allows us to evaluate the expected revenue of any assortment *absolutely*, rather than only in relative terms, and to quantify the optimality gap directly. Moreover, because $P_{\texttt{GT}}(\cdot \mid \cdot)$ is explicitly constructed to incorporate both ranking-based preferences and choice overload, the synthetic datasets $\mathcal{T}_1, \ldots, \mathcal{T}_{10}$ capture consumer non-rationality solely through the choice overload effect, without additional confounding behaviors.

E.4. Model Estimation and Performance Metrics

Given a dataset $\mathcal{T}_i$, we estimate the MNL, ranking-based, and decision forest models. The MNL model is estimated by the standard maximum likelihood estimation (Train 2009). For the latter two, we follow the estimation procedures described in Section 6. In particular, for the decision forest model we again restrict the tree depth to three, and for the ranking-based model we limit the length of estimated rankings to at most three products above the no-purchase option. This restriction is imposed by adding an extra constraint to the column-generation subproblem when estimating the ranking-based model from data (van Ryzin and Vulcano 2014)[1]. Under this setting, the class of ranking-based models we learn from data exactly contains the ground-truth model $P_{\texttt{GT}}(\cdot \mid \cdot)$ when $\alpha_{\text{OL}} = 0$. Our empirical experience suggests that without imposing

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

this length restriction, the ranking-based models lead to substantial performance degradation in assortment tasks examined in this section – sometimes performing as poorly as the MNL model.

To assess the performance of each choice model in the assortment task, we propose a performance metric based on 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 $P_{\texttt{GT}}$ 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

$$G_{\mathcal{M}} = 100\% \times \left(Z^* - Z^{\mathcal{M}}\right)/Z^*. \tag{23}$$

Note that the price information ρ_i of each sushi type $i \in \mathcal{N}$ needed to calculate the expected revenue is provided in the original dataset (Kamishima 2003)

Before presenting the experimental results, we make additional remarks on the optimality gap metric (23). This metric can be interpreted as a measure of a model's *out-of-sample* performance in the data-to-decision assortment task. Recall that the observed assortments $\mathcal{S}_{\text{sample}}$ cover only about one-seventh of all possible assortments. Consequently, a choice model must be able to generalize its revenue predictions to unobserved assortments in order to make good decisions. More precisely, a choice model $\mathcal{M}$ must extend the choice probability vector $P_{\mathcal{M}}(\cdot \mid S)$ beyond the assortments in $\mathcal{S}_{\text{sample}}$, be able to compute the expected revenue of every possible assortment, and then identify the revenue-maximizing assortment $S^{\mathcal{M}}$. In this sense, finding $S^{\mathcal{M}}$ requires substantial generalizability beyond the training data.

E.5. Results

Figure 7 compares the out-of-sample revenue performance of the MNL, ranking-based, and decision forest models, as measured by the optimality gap (23). The x-axis shows the intensity of the choice overload effect $\alpha_{\text{OL}} \in \{0.005, 0.01, 0.015, 0.02, 0.025, 0.03\}$ tested in this experiment. The y-axis shows the optimality gap $G_{\mathcal{M}}$, with $\mathcal{M}$ representing the MNL (MNL), ranking-based (RM), and decision forest (DF) models. Recall that for each α_{OL}, we generate ten transaction datasets $\mathcal{T}_i$, $i = 1, \ldots, 10$. Each dot in the figure represents the average $G_{\mathcal{M}}$ across these datasets, with error bars indicating one standard deviation.

Among the three choice models, Figure 7 shows that the MNL model performs the worst. Its optimality gap consistently increases with the intensity of choice overload, reaching roughly twice that of the ranking-based model. At the same time, the optimal assortment S^{MNL} is remarkably stable compared to the other models, exhibiting zero standard deviation across the ten trials except when $\alpha_{\text{OL}} = 0.005$. This illustrates that, while the simplicity of the MNL model limits its ability to produce high-quality assortments, it offers an advantage in terms of stability for downstream operational tasks.

When the choice overload is slight (i.e., $\alpha_{\text{OL}} = 0.005$), the ranking-based model performs best among the three. This is expected because, when $\alpha_{\text{OL}} = 0$, the ground-truth model (21) is a ranking-based model with length-three rankings – exactly the class of model we estimate from the data (see Section E.4). The small increase to $\alpha_{\text{OL}} = 0.005$ does not materially affect its performance. The sub-optimality of the ranking-based model in this setting, around 4.5%, can be viewed as a baseline for this experiment: it reflects the challenge of generalizing the assortment decision when the model observes only a small fraction (roughly one-seventh) of all assortments in the dataset.

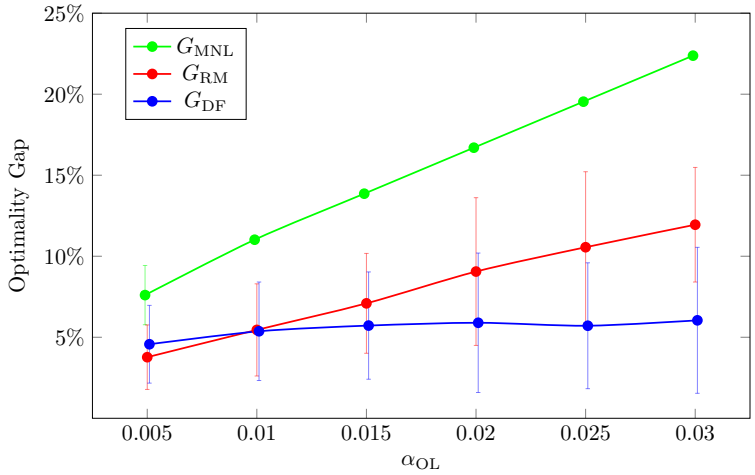


Figure 7 **The performance of the MNL model, ranking-based model (RM) and the decision forest model (DF) measured by the optimality gap of the assortments they return.**

When α_{OL} is further increased to 0.02, the ranking-based model's performance noticeably degrades, and the decision forest model surpasses it. This illustrates that even a mild choice overload, where customers are only 2% more likely to opt out of purchasing as the assortment size increases, can cause rational choice models, such as the ranking-based and MNL models, to suffer substantial sub-optimality, in this case at least 8%. In contrast, the decision forest model remains robust: as α_{OL} increases from 0.01 to 0.03, its optimality gap stays around 5–6%, while the gaps for the ranking-based and MNL models rise to 13% and 22%, respectively.

These results highlight the flexibility of the decision forest model in general, non-rational choice regimes. When the ground-truth model is close to purely rational, more restricted models like the ranking-based model can outperform the decision forest model, as the latter's flexibility may lead to overfitting and worse out-of-sample performance. However, this flexibility becomes an advantage under non-rational behavior: behavioral effects such as choice overload can significantly degrade the performance of rational choice models, whereas the decision forest model maintains consistent performance.

Sections F and G are not included in this electronic companion due to page limits. These materials are available at `https://arxiv.org/abs/2103.14067`.

References for E-Companion

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.

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

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.

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

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

G. van Ryzin and G. Vulcano. A market discovery algorithm to estimate a general class of nonparametric choice models. *Management Science*, 61(2):281–300, 2014.