Abstract

We consider the assortment optimization problem under the classical two-level nested logit model. We establish a necessary and sufficient condition for the optimal assortment, and use this optimality condition to develop a simple greedy algorithm that iteratively removes at most one product from each nest, until the optimality condition is satisfied. Our algorithm also exploits the beautiful “lumpy” structure of the optimal solution, where in each nest, a certain set of “consecutive” products will always appear together in the optimal assortment. The algorithm is simple, intuitive, and extremely fast. For a problem with \( m \) nests, with each nest having \( n \) products, the running time is \( O(nm \log m) \). This is the fastest known running time for this problem.

1. Introduction

Since Talluri and van Ryzin (2004) have shown the importance of incorporating customer choice behavior in operations management, the assortment optimization problem has received much attention from researchers. In this problem, a firm wishes to determine the revenue-maximizing assortment of products, assuming that a customer chooses a product from an assortment according to a discrete choice model. As shown in an excellent survey by Kök et al. (2006), this problem has many applications in retail and revenue management.

Talluri and van Ryzin (2004) considered the assortment optimization problem under the multinomial logit (MNL) model, and gave a beautiful algorithm for computing the revenue-maximizing assortment; see also, Gallego et al. (2004); Kunnumkal and Topaloglu (2008); Farias et al. (2012). Although the MNL model admits a tractable solution, it suffers from the independence of irrelevant alternatives property, which results in biased estimates when alternatives are correlated (Train, 2003). The nested logit model generalizes the MNL model
by grouping similar alternatives into a nest and allowing differential substitution patterns within and across nests, thereby partially relaxing the independence of irrelevant alternatives restriction (Williams, 1977; McFadden, 1978). Under the nested logit model, a consumer first chooses a nest of products, and subsequently chooses a product from the chosen nest.

Davis et al. (2011) presented a novel solution to the assortment optimization problem under the two-level nested logit model. For the classical model\(^1\) with \(m\) nests, with each nest having \(n\) products, they transform the combinatorial optimization problem into an equivalent linear program with \(O(nm)\) constraints and \(O(m)\) variables\(^2\), enabling them to compute the optimal solution in \(O(nm^4)\) running time. The linear programming framework can also be used to obtain approximation algorithms for other NP-hard variants of the classical model.

In this paper, we present an alternative solution to the assortment optimization problem under the classical nested logit model. Our approach is based on a simple greedy algorithm that iteratively removes (at most) one product from each nest to increase the revenue. The greedy algorithm does not require any linear program, and it is extremely fast, with a running time of \(O(nm \log m)\). The algorithm makes use of the necessary and sufficient condition for an optimal assortment (Theorem 3), which to our knowledge, is the first such optimality condition for this problem. In addition, our analysis reveals a beautiful “lumpy” structure of the optimal assortment: surprisingly, within each nest, a certain set of “consecutive” products will always appear together in the optimal solution. Moreover, as shown in Theorem 4, by looking at a certain index of each product, we can determine in advance which products will appear together! Our greedy algorithm exploits this structure to obtain a fast running time.

**Problem Description:** We briefly review the setup for the classical two-level nested logit model, along with the corresponding optimization problem. We have \(m\) nests indexed by \(\{1, 2, \ldots, m\}\), and each nest \(i\) has \(n\) products indexed by \(\{1, 2, \ldots, n\}\). We denote the dissimilarity factor of nest \(i\) by \(\tau_i \in (0, 1)\). A product \(j\) in nest \(i\) is denoted by \(j_i\), and in settings where the nest index is clear from context, we simply refer to it as product \(j\). Each product \(j_i\) has a revenue \(r_{j_i} \geq 0\) and a preference weight \(v_{j_i} > 0\). Without loss of generality, we assume that the products in each nest are sorted in a descending order of revenues; that is, for each nest \(i = 1, 2, \ldots, m\),

\[
r_{1i} \geq r_{2i} \geq \cdots \geq r_{ni} \geq 0.
\]

We denote the preference weight of the no-purchase option by \(v_0 > 0\).

---

\(^1\)Under the classical two-level nested logit model (McFadden, 1978; Börch-Supan, 1990), the dissimilarity parameter of each nest is between 0 and 1, ensuring that the model is consistent with random utility maximization; see, Talluri and van Ryzin (2004) for an overview of the random utility theory. We will also assume that the no-purchase option appears as a single product in its own nest.

\(^2\)Solving a linear program with \(h\) constraints and \(k\) variables requires \(O(hk^3)\) operations (Potra and Wright, 2000).

\(^3\)All of the analysis easily extends to the case where some nests have fewer than \(n\) products.
An assortment $S = (S_1, \ldots, S_m)$ consists of products in each nest, with $S_i \subseteq \{1, 2, \ldots, n\}$ for all $i$. For any assortment $S = (S_1, \ldots, S_m)$, the revenue $\text{Rev}(S)$ is given by:

$$\text{Rev}(S) = \sum_{i=1}^{m} Q_i(S) \text{Rev}_i(S_i),$$

where $Q_i(S)$ denotes the probability that a customer chooses a product from nest $i$, and $\text{Rev}_i(S_i)$ denotes the expected revenue from assortment $S_i$ in nest $i$, and

$$Q_i(S) = \frac{\left(\sum_{\ell \in S_i} v_{\ell i}\right)^{\tau_i}}{v_0 + \sum_{k=1}^{m} \left(\sum_{\ell \in S_k} v_{\ell k}\right)^{\tau_k}} \quad \text{and} \quad \text{Rev}_i(S_i) = \frac{\sum_{\ell \in S_i} r_{\ell i} v_{\ell i}}{\sum_{\ell \in S_i} v_{\ell i}}.$$

The corresponding optimization problem is given by:

$$Z^* = \max_{S = (S_1, \ldots, S_m)} \text{Rev}(S)$$

and we denote an optimal assortment by $S^* = (S_1^*, \ldots, S_m^*)$ and let $Z^* = \text{Rev}(S^*)$ denote the optimal revenue. If there are ties, we can choose the optimal assortment according to any pre-determined tie-breaking rule$^4$.

2. Characterization of the Optimal Assortments

Let $\mathcal{N}_+ = \{\{1\}, \{1, 2\}, \ldots, \{1, 2, \ldots, n\}\}$ denote the collection of revenue-ordered subsets. As shown in the following lemma, at the optimal solution, if the nest is nonempty, then the nest is revenue-ordered. The proof follows from Davis et al. (2011), and we omit the details.

**Lemma 1** (Optimal Nest Structure). For $i = 1, \ldots, m$, if $S_i^* \neq \emptyset$, then $S_i^* \in \mathcal{N}_+$.

Although Lemma 1 shows that $S_i^* \in \{\emptyset\} \cup \mathcal{N}_+$ for all $i$, exhaustive search is not feasible because there are $(n + 1)^m$ possible configurations of revenue-ordered subsets among $m$ nests. To develop an efficient algorithm, we will establish a necessary and sufficient condition for an optimal assortment. For any integers $k_1$ and $k_2$, let

$$[k_1, k_2] = \begin{cases} \{k_1, k_1 + 1, \ldots, k_2\} & \text{if } k_1 \leq k_2, \\ \emptyset & \text{otherwise.} \end{cases}$$

$^4$For example, we can choose $S^*$ to be the minimal optimal assortment; that is, there is no other optimal assortment $B = (B_1, \ldots, B_m) \neq S^*$ such that $B_i \subseteq S_i^*$ for all $i$. All of our analysis applies to other tie-breaking rules.
Also, for each nest $i$ and $1 \leq k_1 \leq k_2 \leq n$, let $G_i(k_1, k_2)$ be defined by:

$$G_i(k_1, k_2) = \frac{\text{Rev}_i([k_1, k_2]) - \text{Rev}_i([1, k_2])}{f_i \left( 1 - \frac{\sum_{\ell \in [k_1, k_2]} v_{\ell i}}{\sum_{\ell \in [1, k_2]} v_{\ell i}} \right)} + \text{Rev}_i([1, k_2]),$$

where $f_i : [0, 1] \rightarrow [0, 1]$ is defined by:

$$f_i(x) = \begin{cases} 
  x^{1 - \tau_i} - x & \text{if } x \in [0, 1), \\
  1 - x & \text{if } x = 1 
\end{cases},$$

and it is easy to verify that $f_i(\cdot)$ is a continuous, strictly increasing function, with $f_i(0) = 0$. Also, we define $0/0 = 0$, and thus $G_i(1, k_2) = \text{Rev}_i([1, k_2])$ for all $k_2$.

The following lemma shows that we can use the index function $G_i(\cdot, \cdot)$ to determine if a subset can be removed from $S_i$ to improve the revenue. The proof is given in Appendix A.

**Lemma 2** (When is Removing Subsets Beneficial?). For any assortment $S = (S_1, \ldots, S_m)$, a collection of subsets $\{A_i \subseteq S_i : i = 1, 2, \ldots, m\}$ can be removed from $S$ to achieve a greater revenue if and only if

$$\frac{\sum_{i=1}^m [V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(S_i \setminus A_i)^{\tau_i} \text{Rev}_i(S_i \setminus A_i)]}{\sum_{i=1}^m [V(S_i)^{\tau_i} - V(S_i \setminus A_i)^{\tau_i}]} < \text{Rev}(S),$$

where for any set $X$, $V(X)$ denotes the sum of the preference weights of the products in $X$. Moreover, if $S_i = [1, p_i]$ with $p_i \geq 1$, $A_i = [k_i, p_i]$ with $k_i \leq p_i$, and $A_\ell = \emptyset$ for all $\ell \neq i$, then removing $A_i$ from $S_i$ leads to a greater revenue if and only if $G_i(k_i, p_i) < \text{Rev}(S)$.

The main result of this section is the following theorem, which establishes a necessary and sufficient condition for an optimal assortment. This theorem forms the basis for our greedy algorithm given in Section 3.

**Theorem 3** (Optimality Condition). Consider any assortment $S = ([1, p_1], \ldots, [1, p_m])$ such that $S_i^* \subseteq [1, p_i]$ for all $i$. Then, $S$ is optimal if and only if for every nest $i$ such that $p_i \geq 1$,

$$\min_{j=1, \ldots, p_i} G_i(j, p_i) \geq \text{Rev}(S).$$

**Proof.** Suppose that $S = (S_1, \ldots, S_m)$ is an optimal solution, where $S_i = [1, p_i]$ for all $i$. If $p_i \geq 1$, then $S_i \neq \emptyset$. Since $S$ is optimal, removing a subset $[k_i, p_i]$ from $S_i$ cannot improve the revenue, and thus, it follows from Lemma 2 that $G_i(k_i, p_i) \geq \text{Rev}(S)$ for all $k_i \leq p_i$, which is the desired result.

To establish sufficiency, consider any assortment $S = (S_1, \ldots, S_m)$ such that $S_i^* \subseteq S_i = [1, p_i]$ for all $i$, and $S$ satisfies the condition of the theorem. We will show that $\text{Rev}(S) = Z^*$. For
each i, let \( E_i = S_i \setminus S_i^* = [1, p_i] \setminus [k_i, p_i] \) for some index \( k_i \). By our hypothesis, for every nest \( i \) such that \( E_i \neq \emptyset \), \( G_i(k_i, p_i) \geq \text{Rev}(S) \), which means that removing \( E_i \) from \( S_i \) does not increase the revenue. It then follows from the first part of Lemma 2 that

\[
\text{Rev}(S) \leq \frac{V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(S_i \setminus E_i)^{\tau_i} \text{Rev}_i(S_i \setminus E_i)}{V(S_i)^{\tau_i} - V(S_i \setminus E_i)^{\tau_i}},
\]

where for any set \( X \), \( V(X) \) is the sum of the preference weights of the products in \( X \). Thus,

\[
\text{Rev}(S) \leq \min_{i : E_i \neq \emptyset} \frac{V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(S_i \setminus E_i)^{\tau_i} \text{Rev}_i(S_i \setminus E_i)}{V(S_i)^{\tau_i} - V(S_i \setminus E_i)^{\tau_i}} \leq \frac{\sum_{i : E_i \neq \emptyset} [V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(S_i \setminus E_i)^{\tau_i} \text{Rev}_i(S_i \setminus E_i)]}{\sum_{i : E_i \neq \emptyset} [V(S_i)^{\tau_i} - V(S_i \setminus E_i)^{\tau_i}]},
\]

where the second inequality follows because for any \( x \in \mathbb{R}^k \) and \( y \in \mathbb{R}^k_+ \),

\[
\frac{\sum_{i=1}^{k} x_i - y_i}{\sum_{i=1}^{k} x_i} \geq \min_{i=1, \ldots, k} \frac{x_i}{y_i},
\]

and \( V(S_i)^{\tau_i} - V(S_i \setminus E_i)^{\tau_i} \geq 0 \) for all \( i \). Note that if \( E_i = \emptyset \), then \( S_i = S_i^* \). It then follows from the first part of Lemma 2 that

\[
\text{Rev}(S) \geq \text{Rev}(S_1 \setminus E_1, S_2 \setminus E_2, \ldots, S_m \setminus E_m) = \text{Rev}(S_1^*, \ldots, S_m^*) = Z^*,
\]

which is the desired result. \( \square \)

Surprisingly, as shown in the following theorem, the optimal assortment in each nest is “lumpy”, with certain consecutive products always appearing together. Moreover, for each nest \( i \), by looking at the index \( G_i(j, j) \) of each product \( j \), we can determine in advance which products will appear together!

**Theorem 4** (Lumpiness of the Optimal Assortments). For every nest \( i \), if there exist products \( j \) and \( k \) such that \( j < k \) and

\[
G_i(j, j) < G_i(j + 1, j + 1) < \cdots < G_i(k, k),
\]

then either \( \{j, j + 1, \ldots, k\} \subseteq S_i^* \), or \( \{j, j + 1, \ldots, k\} \cap S_i^* = \emptyset \).

**Proof.** It suffices to prove the result when \( k = j + 1 \); that is, we need to show that if \( G_i(j, j) < G_i(j + 1, j + 1) \), then either \( \{j, j + 1\} \subseteq S_i^* \), or \( \{j, j + 1\} \cap S_i^* = \emptyset \). There are two cases.

**Case 1:** \( j + 1 \in S_i^* \). In this case, \( \{j, j + 1\} \subseteq S_i^* \) because \( S_i^* \in \mathcal{N}_+ \) by Lemma 1.

**Case 2:** \( j + 1 \notin S_i^* \). In this case, we will show that \( j \notin S_i^* \). Suppose on contrary that \( j \in S_i^* \). Then, we have that \( S_i^* = [1, j] \). Let \( \hat{S} = (S_1^*, \ldots, [1, j + 1], \ldots, S_m^*) \). By definition, \( \text{Rev}(\hat{S}) \leq \text{Rev}(S^*) \), and since \( S^* \) is obtained from \( \hat{S} \) by removing product \( j + 1 \) from nest \( i \), it
follows from Lemma 2 that $G_i(j + 1, j + 1) \leq \text{Rev}(\hat{S})$. So, we have that

$$G_i(j, j) < G_i(j + 1, j + 1) \leq \text{Rev}(\hat{S}) \leq \text{Rev}(S^*) ,$$

which implies that $\text{Rev}(S_1^*, \ldots, S_{i-1}^* \setminus \{j\}, \ldots, S_m^*) > \text{Rev}(S^*)$. This contradicts the optimality of $S^*$. Therefore, $j \notin S_i^*$.

\[ \square \]

3. A Greedy Algorithm

Our proposed greedy algorithm generates a sequence of assortments $\{S^t : t = 0, 1, \ldots\}$, terminating with an assortment that satisfies the optimality condition in Theorem 3. In Stage 1 of the algorithm, we exploit the lumpiness structure of the optimal assortments shown in Theorem 4, by combining products that will always appear together into a single group. A formal description of the algorithm is given as follows.

**Greedy Algorithm**

**Stage 1 (Lumping):** Compute the index $G_i(j, j)$ for every product $j$ in every nest $i$. For each nest $i$, if $G_i(j, j) < G_i(j + 1, j + 1)$, then it follows from Theorem 4 that either $\{j, j + 1\} \subseteq S_i^*$ or $\{j, j + 1\} \cap S_i^* = \emptyset$. Thus, we can combine the two products together, and replace products $j$ and $j + 1$ with a single “new” product with a revenue $\text{Rev}_i([j, j + 1]) = \frac{v_{ji}r_{ji} + v_{j+1,i}r_{j+1,i}}{v_{ji} + v_{j+1,i}}$ and a preference weight $v_{ji} + v_{j+1,i}$. Assign the new product the index $j$, and calculate the new $G_i(j, j)$. Repeat this process until we obtain a list of indices $G_i(j, j)$ that is non-increasing in $j$. Without loss of generality, assume that at the end of Stage 1, each nest $i$ has $n$ products such that

$$G_i(1, 1) \geq G_i(2, 2) \geq \cdots \geq G_i(n, n).$$

**Stage 2 (Removal):** Let $S^1 = ([1, n], \ldots, [1, n])$. For each iteration $t \geq 1$, given $S^t = ([1, J^t_1], [1, J^t_2], \ldots, [1, J^t_m])$, let

$$\text{Min}^t = \min_{i: J^t_i \geq 1} G_i(J^t_i, J^t_i) \quad \text{and} \quad \text{Index}^t = \arg\min_{i: J^t_i \geq 1} G_i(J^t_i, J^t_i).$$

If $\text{Min}^t \geq \text{Rev}(S^t)$, the algorithm terminates and outputs $S^t$. Otherwise, if $\text{Min}^t < \text{Rev}(S^t)$, then the algorithm generates a new assortment $S^{t+1} = (S^{t+1}_1, \ldots, S^{t+1}_m)$ as follows:

$$S^{t+1}_i = \begin{cases} S^t_i & \text{if } i \neq \text{Index}^t \\ S^t_i \setminus \{J^t_i\} = [1, J^t_i - 1] & \text{if } i = \text{Index}^t \end{cases}$$

Thus, in each iteration, we remove $J^t_i$ from $S^t_i$ if $G_i(J^t_i, J^t_i) = \text{Min}^t$ and the product $J^t_i$ violates the optimality condition given in Theorem 3, that is, $G_i(J^t_i, J^t_i) < \text{Rev}(S^t)$. 6
The following lemma shows that $S^t$ always contains the optimal assortment.

**Lemma 5** (Containment). For all $t$, $S^* \subseteq S^t$.

*Proof.* We will prove this result by induction. It’s true for $t = 1$ by our construction. Suppose that the lemma is true for $t$; that is, $S^* \subseteq S^t$. We wish to show that $S^*_i \subseteq S^{t+1}_i$ for all $i$. Consider an arbitrary nest $i$. There are two cases to consider: $S^*_i = S^t_i$ and $S^*_i \subset S^t_i$.

**Case 1:** Suppose that $S^*_i = S^t_i = [1, J^t_i]$. If $J^t_i = 0$, then $i \not= \text{index}^t$ and $S^{t+1}_i = S^t_i$ by our construction. So, suppose that $J^t_i \geq 1$. Since $\text{Rev}(S^*) \geq \text{Rev}(S^*_1, \ldots, S^*_m)$, it follows from Lemma 2 that $G_i(J^t_i, J^t_i) \geq \text{Rev}(S^*) \geq \text{Rev}(S^t)$, where the last inequality follows from the optimality of $S^*$. Since $G_i(J^t_i, J^t_i) \geq \text{Rev}(S^t)$, $S^{t+1}_i = S^t_i$, which is the desired result.

**Case 2:** Suppose that $S^*_i \subset S^t_i = [1, J^t_i]$. Then, $J^t_i \not= \text{index}^t$ and $J^t_i \geq 1$. Since $S^*_i \in \mathcal{N}_i$, we have that $S^*_i \subset [1, J^t_i - 1]$. By our construction, we will either remove $J^t_i$ from $S^t_i$ or do nothing, and in both cases, we have $S^*_i \subseteq S^{t+1}_i$, which is the desired result. \( \square \)

The main result of this section is stated in the following theorem.

**Theorem 6** (Correctness). The GREEDY ALGORITHM terminates with an optimal assortment, with a running time of $O(nm \log m)$.

*Proof.* We will first establish the running time of the algorithm. By definition,

$$G_i(j, j) = \frac{r_{ji} - \text{Rev}_i([1, j])}{f_i \left(1 - \frac{v_{ji}}{\sum_{t=1}^m v_{ti}}\right)} + \text{Rev}_i([1, j]),$$

and note that $\text{Rev}_i([1, j])$ is a convex combination of $\text{Rev}_i([1, j - 1])$ and $r_{ji}$. For simplicity, assume that we can compute the function $f_i(x)$ for each $x$ in constant time, and thus, by starting from $j = 1$, we can compute the index $G_i(j, j)$ for each product $j$ in each nest $i$ in constant time. Moreover, for each nest $i$, it is easy to verify that the “lumping” process in Stage 1 requires computations of $O(n)$ indices, and at most $O(n)$ comparisons. With $m$ nests, the total running time for Stage 1 is thus $O(nm)$. For a detailed proof, see Appendix B.

During the removal in Stage 2, the algorithm will continue to run as long as a single product is removed from any nest. There are $nm$ products; thus, the algorithm will terminate in $O(nm)$ iterations. We will now show that each iteration takes $O(\log m)$ operations.

At the beginning of Stage 2, we create a self-balancing binary search tree (SB-BST) with $m$ nodes, where for $i = 1, 2, \ldots, m$, node $i$ in the tree corresponds to the index $G_i(n, n)$. This takes $O(m \log m)$ operations; see Chapter 6 in Knuth (1998) for more details. We use the SB-BST as our data structure because such a tree always maintains a height of $O(\log m)$, allowing for an efficient search operation. In each iteration $t \geq 1$, we perform the following 3 operations:

\begin{itemize}
  \item \text{Search}($G_i(n, n)$)
  \item \text{Remove}($G_i(n, n)$)
  \item \text{Find}($G_i(n, n)$)
\end{itemize}
SEARCH: The algorithm searches for the nest with the minimum index $G_i(J_i^1, J_i^1)$. This can be done in $O(\log m)$ operations under the SB-BST.

DELETE: If the minimum index $G_i(J_i^1, J_i^1)$ is greater than or equal to the revenue of the current assortment, the algorithm terminates. Otherwise, the algorithm removes the product $J_i^1$ from $S_i^1$ in nest $i$, and also removes node $i$ from the tree. The removal is done in $O(\log m)$ operations because the tree may need to re-balance its heights.

INSERT: If the product $J_i^1$ in nest $i$ is removed and $J_i^1 > 1$, the algorithm adds a new node with a corresponding index $G_i(J_i^1 - 1, J_i^1 - 1)$ into the tree. Again, the insertion of a new node in SB-BST takes $O(\log m)$ operations.

Therefore, each iteration in Stage 2 takes $O(\log m)$ operations. As we have $O(nm)$ iterations, the total running time is $O(nm \log m)$.

Let $S = ([1, p_1], \ldots, [1, p_m])$ denote the assortment at the termination of the greedy algorithm. By our construction, for every nest $i$ such that $p_i \geq 1$, we have $G_i(p_i, p_i) \geq \text{Rev}(S)$. To complete the proof, we will show that $S$ satisfies the optimality condition in Theorem 3; that is, for every nest $i$ such that $p_i \geq 1$,

$$\min_{j=1,2,\ldots,p_i} G_i(j, p_i) \geq \text{Rev}(S).$$

We will prove this by contradiction. Suppose on the contrary that the optimality condition is violated, and $\min_{j=1,2,\ldots,p_i} G_i(j, p_i) < \text{Rev}(S)$. Let $k_i \in \{1, 2, \ldots, p_i - 1\}$ denote the largest index where the optimality condition is violated; that is,

$$G_i(k_i, p_i) < \text{Rev}(S) \leq G_i(k_i + 1, p_i),$$

and it follows from Lemma 2 that

$$\text{Rev}(S_1, \ldots, S_i \setminus [k_i, p_i], \ldots, S_m) > \text{Rev}(S) \geq \text{Rev}(S_1, \ldots, S_i \setminus [k_i + 1, p_i], \ldots, S_m).$$

Note that $S_i \setminus [k_i, p_i] = [1, k_i - 1]$ and $S_i \setminus [k_i + 1, p_i] = [1, k_i]$. Thus, removing product $k_i$ from $[1, k_i]$ in nest $i$ increases the revenue, and it follows from Lemma 2 that

$$G_i(k_i, k_i) < \text{Rev}(S_1, \ldots, S_i \setminus [k_i + 1, p_i], \ldots, S_m) \leq \text{Rev}(S).$$

Since we lump the products in Stage 1 so that $G_i(1, 1) \geq \cdots \geq G_i(p_i, p_i)$ and $k_i < p_i$, we have

$$G_i(p_i, p_i) \leq G_i(k_i, k_i) < \text{Rev}(S),$$

which contradicts our hypothesis on $p_i$. Therefore, $\min_{j=1,2,\ldots,p_i} G_i(j, p_i) \geq \text{Rev}(S)$. □

Numerical Results: We conduct small numerical experiments to compare the running time of our greedy algorithm with the linear programming formulation of Davis et al. (2011).
We consider 9 problem classes, where each problem class is characterized by the number of nests $m$, and the number of products $n$ in each nest, with $(m, n) \in \{10, 50, 100\} \times \{10, 50, 100\}$.

For each problem class $(m, n)$, we generate 1000 independent problem instances, and for each instance, $r_{ij}$ are sampled from a uniform distribution on the interval $[0, 20]$, $v_{ij}$ are uniformly distributed on $[0, 10]$, $v_0$ is uniformly distributed on $[0, 5]$, and finally, the dissimilarity parameter $\tau_i$ for each nest $i$ is drawn from a uniform distribution on $[0, 1]$. Table 1 shows the average running time for each problem class. To solve the linear program, we use the MATLAB LP Solver, and it turns out that both the interior point and simplex methods have roughly comparable running times, so we report only the running time from the interior point method.

For the greedy algorithm, to simplify our implementation, we use a standard array of size $m$ to implement Stage 2, instead of the self-balancing binary search tree. Even with this simple implementation, our greedy algorithm still outperforms the LP in all problem instances.

In columns 3 and 4 of Table 1, we show the average running time in seconds for each problem class. We observe that the greedy algorithm is, on average, faster than the LP by an order of magnitude. We also want to compare the running time on every problem instance. So, for each problem instance, we compute the ratio between the running time of LP and the running time of greedy, and report the minimum, median, and maximum of these ratios in columns 5, 6, and 7. Since the minimum is at least 2.2, we see that the greedy is always at least twice as fast as the LP, in every problem instance. In fact, for large problem instances, the greedy can be up to 99 times faster than the LP.

<table>
<thead>
<tr>
<th>Problem Class</th>
<th>Avg. Running Time (in seconds)</th>
<th>Ratio Between the LP and Greedy Running Times for Each Problem Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>$n$</td>
<td>LP</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.037</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.074</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.137</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>0.199</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>2.041</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>1.848</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>0.782</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
<td>6.368</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>19.635</td>
</tr>
</tbody>
</table>

Table 1: Columns 3 and 4 show the average running time in seconds for the LP and greedy algorithms. All numbers are statistically significant at 5%. We also consider the ratio between the LP and greedy running times for each problem instance, and the statistics of these ratios are reported in columns 5, 6, and 7.

\[5\text{By using the standard array of size } m, \text{ the running time of the greedy algorithm increases to } O(nm^2) \text{ because in each iteration of Stage 2, searching for the minimum index requires } O(m) \text{ operations, instead of } O(\log m) \text{ under the self-balancing binary search tree.}\]
4. Conclusion

We present a simple and fast greedy algorithm for the assortment optimization problem under the classical two-level nested logit model. Our algorithm makes use of the optimality condition, and exploit the “lumpy” structure of the optimal assortment. As a direction for future research, it would be interesting to explore whether the optimality condition can be established for more general choice models, or if a similar lumpy structure exists for other choice models.

Acknowledgement

We would like to thank Huseyin Topaloglu for introducing us to the surprising connection between the assortment optimization problem under the two-level nested logit model and its equivalent linear programming formulation. His work has inspired us to consider the problem in this paper.

References


A. Proof of Lemma 2

Proof. For \( i = 1, \ldots, m \), let \( S_i = S_1 \setminus A_i \). Assume that \( A_i \neq \emptyset \) for some \( i \); otherwise, the result is trivially true. Since \( v_{ji} > 0 \) for all \( ji \), \( \sum_{i=1}^{m} V(\hat{S}_i) \sum_{i=1}^{m} V(S_i) \). By definition,

\[
\text{Rev}(S) = \frac{\sum_{i=1}^{m} V(S_i)\text{Rev}(S_i)}{v_0 + \sum_{i=1}^{m} V(S_i)\tau_i} = \left( \frac{v_0 + \sum_{i=1}^{m} V(\hat{S}_i)\tau_i}{v_0 + \sum_{i=1}^{m} V(S_i)\tau_i} \times \frac{\sum_{i=1}^{m} V(\hat{S}_i)\tau_i \text{Rev}(\hat{S}_i)}{v_0 + \sum_{i=1}^{m} V(S_i)\tau_i} \right) + \left( \frac{\sum_{i=1}^{m} V(S_i)\tau_i \text{Rev}(S_i) - V(\hat{S}_i)\tau_i \text{Rev}(\hat{S}_i)}{v_0 + \sum_{i=1}^{m} V(S_i)\tau_i} \right)
\]

Thus, \( \text{Rev}(S) \) is a convex combination of \( \text{Rev}(\hat{S}) \) and \( \sum_{i=1}^{m} \left[ V(S_i)\tau_i \text{Rev}(S_i) - V(\hat{S}_i)\tau_i \text{Rev}(\hat{S}_i) \right] \), which gives the desired result.

Suppose that \( S_i = [1, p_i] \), \( A_i = [k_i, p_i] \neq \emptyset \), and \( A_k \neq \emptyset \) for all \( k \neq i \). Using the fact that \( \text{Rev}_i(\hat{S}_i) = \frac{V(S_i)}{V(\hat{S}_i)} \text{Rev}(S_i) + \left( 1 - \frac{V(S_i)}{V(\hat{S}_i)} \right) \text{Rev}_i(A_i) \), and that

\[
f_i \left( 1 - \frac{V(A_i)}{V(S_i)} \right) = f_i \left( \frac{V(S_i)}{V(\hat{S}_i)} \right) = \frac{\left( \frac{V(S_i)}{V(\hat{S}_i)} \right)^{1-\tau_i} - \frac{V(\hat{S}_i)}{V(S_i)}}{1 - \frac{V(\hat{S}_i)}{V(S_i)}} = \frac{V(S_i)^{\tau_i} - V(\hat{S}_i)^{\tau_i}}{V(\hat{S}_i)^{\tau_i-1}V(S_i)} \frac{V(\hat{S}_i)^{\tau_i-1}V(S_i) - V(\hat{S}_i)^{\tau_i}}{V(S_i)^{\tau_i-1}V(\hat{S}_i) - V(\hat{S}_i)^{\tau_i}} \frac{V(S_i)^{\tau_i} - V(\hat{S}_i)^{\tau_i}}{V(\hat{S}_i)^{\tau_i-1}V(S_i) - V(\hat{S}_i)^{\tau_i}} \frac{\text{Rev}(S_i)}{V(S_i)} \frac{1}{V(\hat{S}_i)} \text{Rev}_i(S_i)
\]

we have

\[
V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(\hat{S}_i)^{\tau_i} \text{Rev}_i(\hat{S}_i)
\]

Therefore

\[
\frac{V(S_i)^{\tau_i} \text{Rev}_i(S_i) - V(\hat{S}_i)^{\tau_i} \text{Rev}_i(\hat{S}_i)}{V(S_i)^{\tau_i} - V(\hat{S}_i)^{\tau_i}} < \text{Rev}(S)
\]

if and only if \( G_i(k_i, p_i) < \text{Rev}(S) \). 

\( \square \)
B. Complexity of the “Lumping” Step (Stage 1)

Proof. For each nest $i$, it suffices to show that the lumping process in Stage 1 takes $O(n)$ operations, in order to create a list of indices $G_i(j,j)$ that is non-increasing in $j$. The lumping process requires two types of operations: 1) comparison between two indices, and 2) merging of two products, which requires the computation of the new value of $G_i(\cdot,\cdot)$. For simplicity, we assume that computing the index $G_i(\cdot,\cdot)$ takes constant time. Whenever we merge two nodes to create a new one, the total number of nodes is reduced by one. Since we start with $n$ nodes, the number of mergings is at most $n - 1$.

To complete the proof, we will show that the number of comparisons is at most $O(n)$, by creating a directed graph that keeps track of the total number of comparisons. Create a node $j$ for each product $j \in \{1,2,\ldots,n\}$ in nest $i$. Place these nodes from left to right in an increasing order of index, so node 1 will be the leftmost node, followed by node 2, and node $n$ will be the rightmost node. We refer to these $n$ nodes as the “original” nodes.

For each node $j$, let $j_\ell$ and $j_r$ represent the adjacent nodes, on the left and right of node $j$, respectively. Whenever we compare $G_i(j,j)$ with $G_i(k,k)$, we add a directed edge $(j,k)$ from node $j$ to node $k$. We say that a violation occurs if $G_i(j,j) < G_i(j_r,j_r)$.

We start from the left most node (node 1). When we are at the “original” node $j$, we compare $G_i(j,j)$ with $G_i(j+1,j+1)$, and add an edge $(j,j+1)$. If no violation occurs, then $G_i(j,j) \geq G_i(j+1,j+1)$, and we proceed forward to node $j + 1$. On the other hand, if a violation occurs, then we must merge products $j$ and $j + 1$ together. This is done by removing both nodes $j$ and $j + 1$, and replace them with a “new” node $j'$, and we calculate the new index $G_i(j',j')$.

Whenever we create a “new” node $j'$, to ensure the correct ordering of indices, we need to compare $G_i(j',j')$ with the value of its left adjacent node, denoted by $j'_\ell$. So, when a “new” node $j'$ is created, we add a backward edge $(j',j'_\ell)$. If a violation occurs, we replace both nodes $j'$ and $j'_\ell$ with a new node. Otherwise, we have the correct ordering, and we compare node $j'$ with its right neighbor $j'_r$, and of course, add an edge $(j',j'_r)$ to keep track of this comparison.

The process stops when all existing nodes are connected and no violation occurs. Each of the $n$ “original” nodes (except for the rightmost node) has at most one outgoing edge to its right neighbor. Moreover, each newly created node has at most two outgoing edges (one to its left neighbor and another to its right). Since a new node is created only when a merging occurs, there are at most $n - 1$ new nodes. Therefore, the total number of edges is bounded by $(n - 1) + 2(n - 1) = 3(n - 1)$, and thus, the total number of comparisons is $O(n)$. $\square$