Abstract

We revisit the problem of assortment optimization under the multinomial logit choice model with general constraints and propose new efficient optimization algorithms. Our algorithms do not make any assumptions on the structure of the feasible sets and in turn do not require a compact representation of constraints describing them. For the case of cardinality constraints, we specialize our algorithms and achieve time complexity sub-quadratic in the number of products in the assortment (existing methods are quadratic or worse). Empirical validations using the billion prices dataset and several retail transaction datasets show that our algorithms are competitive even when the number of items is $\sim 10^5$ and beyond (100x larger instances than previously studied), supporting their practicality in data driven revenue management applications.

1 Introduction

Assortment optimization [11] is the problem of showing a right subset (assortment) of items to a buyer taking into account their purchase (choice) behavior, and is a key problem studied in the revenue management literature. Both the offline [3] and online [1] optimization settings have a wide variety of applications in retail, airline, hotel and transportation industries among others. There are essentially two aspects to this problem: (a) the purchase behavior of the buyer, and (b) the metric that the seller wishes to optimize. Intuitively, the subset shown to the buyer impacts their purchase behavior, which in turn impacts the seller’s desired metric such as conversion or revenue. In this work, we focus on a popular parametric behavior model called the Multinomial Logit (MNL) model, and assume that the seller’s objective is to maximize expected revenue by choosing the right assortment (prices are assumed known and fixed). The algorithms that we propose for generating these optimal assortments (namely,
Assort-MNL, Assort-MNL(Approx) and Assort-MNL(BZ)) build on (noisy) binary search and make use of efficient data structures for similarity search, both of which have not been used for assortment optimization before. The consequence of these choices is that we can solve problem instances with extremely general specification of candidate assortments.

In fact, we motivate that frequent itemsets [4] discovered using transaction logs can readily give us candidate assortments, that can lead to large scale assortment optimization problem instances. There is a rich history of frequent itemset mining, both in research and in practice. Although, no previous work has connected them to assortment planning, we believe they are a natural fit and aligned with the seller’s objective of maximizing expected revenue. Often these sets are very large in number and cannot be easily represented using a compact (integer) polytope. This latter property is key for efficient algorithms in previous works. In particular, if assortment planning is carried over the polytope of all itemsets up to a fixed size, then under the MNL model, efficient algorithms proposed in [15, 10] can be used (generalizations to unimodular polytopes have also been studied). In reality, we would like to specify arbitrary/data-driven itemsets as candidate assortments in our optimization problem. For instance, store managers can suggest arbitrary feasible assortments based on domain knowledge and other business constraints. Unfortunately, no existing algorithms (including integer programming) work when these sets are not compactly represented. On the other hand, by leveraging recent advances in similarity search, we show that we can solve fairly large instances (∼10^5 items) within reasonable computation times for such hard instances.

Another significant motivation for focusing on the computational efficiency of assortment optimization is the following. Choice behavior varies across buyers as well as across time for the same buyer, and thus showing a single optimal assortment may not be ideal. In fact, if the seller assumes a single static purchase behavior model for her target population and does not personalize, she may drastically lose out on potential sales (especially of items not in the single assortment being used). As new information about the buyer (for instance, through click logs or some other form of feedback) updates their choice model, a real-time optimization of the most suitable assortment given current models of the choice behavior is highly desirable. In particular, such an optimization scheme should be scalable and extremely efficient to make personalization practical. Note that although one could potentially look at continuous estimation and optimization of assortments in an online learning framework [1] that is scalable, decoupling these two operations brings in a lot of flexibility for both steps, especially because the optimization approaches can be much better tuned for performance. Even when one is interested in an online learning scheme, efficient algorithms, such as the ones we propose, can easily be used as subroutines to regret minimizing online assortment optimization schemes.

While no methods other than integer programming exist when when the constraints on the feasible set are not unimodular, the special case of capacity constrained setting (where all assortments with size less than a threshold are feasible) has been well studied algorithmically. The key approaches are: (a) an linear programming (LP) based approach [8] (time complexity O(n^{3.5}) if based on interior point methods), (b) the STATIC-MNL algorithm (time complexity O(n^2 log n) [15]), and (c) ADXOpt (time complexity O(n^2 b C) where C is the maximum size of the assortment and b = min\{C, n - C + 1\} for the MNL choice model) [10]. Note that ADXOpt is a local search search heuristic that is not guaranteed to find the optimal when a general collection of feasible assortments are considered. It is not clear how to extend these
methods when the feasible assortments cannot be compactly represented. In contrast, not only do our methods become more time (subquadratic in the number of items) and space efficient in the capacity-constrained assortment setting, they are also better empirically as shown in our experiments. Even in the setting with general assortments, our algorithms compute (nearly) optimal assortments for instances of size 50000 within 5 seconds on average. These experiments are carried out using prices from the Billion Prices dataset [7] as well as using frequent itemsets mined from transaction logs [4].

A key feature of our algorithms is that we decompose the optimization problem into a sequence of inner product search problems and employ efficient similarity search routines and data structures to speed up the searches. Its iterative nature allows for trading off accuracy with the number of iterations, which can be desirable when the problem instances are very large. While, some researchers [3] have tried to improve empirical performance while using richer choice models (e.g., the distribution over rankings model, Markov chain model etc.) and general feasible assortments, their modeling approach (integer programming) is not scalable and is limited to instances where the sets can be described efficiently using a polytope. On the other hand, we fix the choice model to be the MNL model and design scalable algorithms applicable for large scale instances that arise in practice, especially in the Internet retail/e-commerce settings.

In Section 2, we describe some preliminary concepts. Our proposed algorithms are in Section 3, whose performance we empirically validate in Section 4. Finally, Section 5 presents some concluding remarks and avenues for future work.

2 Preliminaries

2.1 Assortment Planning

Without loss of generality, let the items be indexed from 1 to n in the decreasing order of their prices, i.e., \( p_1 \geq p_2 \geq \cdots p_n \). The assortment planning problem concerns with choosing the assortment that maximizes the expected revenue (note that we use price and revenue interchangeably throughout the paper). Let \( f(A) = \sum_{l \in A} p_l \mathbb{P}(l|A) \) denote the revenue of the assortment \( A \subseteq \{1, \ldots, n\} \). Here \( \mathbb{P}(l|A) \) in the choice model that represents the probability that a user selects item \( l \) when assortment \( A \) is shown to them. The expected revenue assortment optimization problem is: \( \max_A f(A) \). There are many choice models that have been proposed in the literature [16] including the Mixture of MNLs model, the nested logit model, the distribution over rankings model and the Markov chain model. Some of these are more expressive than the MNL choice model that we consider, but are harder to estimate reliably from data as well as make the corresponding optimization problems NP-hard.

Typically, in addition to choice models, the compactness of representation of feasible assortments also influences the computational tractability of the problem. In Section 3, we develop efficient algorithms for assortment optimization that do not require any structural assumption on the feasible sets, nor do they require a compact representation. The algorithms only require a set \( S \) of all the feasible assortments as input. Furthermore, when some structure is present, for instance, when we want to optimize over all assortments of a given size (i.e. constraints of the type \( c \leq |A| \leq C \)), we can improve the time and space complexity of our
2.2 The Multinomial Logit Model

Let the MNL choice model \[12\] parameters be represented by a vector \( \mathbf{v} = (v_0, v_1, \cdots v_n) \) with \( 0 \leq v_i \leq 1 \ \forall i \). Parameter \( v_i, 1 \leq i \leq n \), captures the preference of the user for purchasing product \( i \). To be precise, \( \log(v_i) \) is the mean utility derived from product \( i \) (MNL is an instance of the random utility maximization framework where utilities are random variables that are Gumbel distributed). Similarly, parameter \( v_0 \) captures the preference for not making any purchase or selection. For this model, \( \mathbb{P}(l|A) = \frac{v_l}{v_0 + \sum_{i \in A} v_i} \). Intuitively, the probability with which a buyer will pick an item increases when the item is shown with other items with lower mean utilities, and vice versa. Owing to its structure, the MNL model leads to tractable assortment optimization problems when the feasible sets can be represented compactly. For instance, in [3, Section 5, Table 11], the authors show that capacity-constrained assortment optimization under the MNL model can be solved fairly quickly (more than 5 times faster than a proposed integer programming formulation), with relatively small gap from the true optimal revenue (a Mixed Multinomial Logit model was used as the ground truth). Coupled with the fact that estimating MNL parameters is relatively easy, this evens out some of its shortcomings such as: (a) under-fitting the data, and (b) satisfying the independence of irrelevant alternatives property.

2.3 Maximum Inner Product Search (MIPS)

The MIPS problem is that of finding the vector in a given set of points which has the highest inner product with a query vector. Precisely, for a query vector \( q \) and set of points \( X \) (which may not be compactly represented), the optimization problem is: \( \max_{x \in X} q \cdot x \) (the ‘\( \cdot \)’ operation stands for inner product). In the case where there is no structure on \( X \), one can solve for the optimal via a linear scan that can be quite slow for large instances. To get around this, we can also solve approximately using methods \[13\] based on Locality Sensitive Hashing (LSH) and variants. Such approximate methods have also been proposed to related problems such as the Jaccard Similarity (JS) search and the Nearest-Neighbor (NN) search in the information retrieval literature. MIPS is a key part of all three of our algorithms, and we make use of a particular approximate method in one of our algorithms, namely ASSORT-MNL(Approx). Another algorithm, ASSORT-MNL, relies on a subroutine that solves the MIPS problem exactly.

2.4 Locality Sensitive Hashing (LSH)

LSH \[2\] is a technique for finding vectors from a known set of vectors, that are (approximately) ‘similar’ (according to some metric) to a given query vector. It uses hash functions such that points (vectors) which are similar are more likely to have the same hash value as compared to points which are not. The similarity metric that arises as a part of our algorithms is the inner product metric, and a variation of LSH proposed in \[13\] is one way to compute an approximate solution fast. The procedure is as follows: for \( x \in \mathbb{R}^n, \|x\|_2 \leq 1 \), we first
define a preprocessing transformation \( P : \mathbb{R}^n \rightarrow \mathbb{R}^{n+1} \) as \( P(x) = [x; \sqrt{1-||x||^2}] \). To hash the processed points, we use a spherical random vector \( a \sim \mathcal{N}(0, I) \) and define the hash function as \( h_a(x) = \text{sign}(a \cdot x) \). Without loss of generality, assume that a given query \( y \) have \( ||y||_2 = 1 \). We first transform the given query using \( P(\cdot) \), and then check for collisions of its hashes with the indexed points. The following guarantee can be shown for the probability of collision of two hashes:

\[
P[h_a(P(x)) = h_a(P(y))] = 1 - \frac{\cos^{-1}(x \cdot y)}{\pi},
\]

which is a decreasing function of the inner product \( x \cdot y \). For any chosen threshold value \( S \) and \( c < 1 \), we consequently get the following:

- If \( x \cdot y \geq D \), then \( P[h_a(P(x)) = h_a(P(y))] \geq 1 - \frac{\cos^{-1}(D)}{\pi} \).
- If \( x \cdot y \leq cD \), then \( P[h_a(P(x)) = h_a(P(y))] \leq 1 - \frac{\cos^{-1}(cD)}{\pi} \).

In practice, as described in [2], we use multiple hash functions (of the type \( h_a(\cdot) \)) to further increase confidence in the reported near neighbors (defined in terms of the similarity metric). The number of such hash functions is determined by parameters \( L_1 \) and \( L_2 \). In particular, we can choose \( L_2 \) functions of dimension \( L_1 \), denoted as \( g_j(q) = (h_{1,j}(q), h_{2,j}(q), \ldots, h_{L_1,j}(q)) \), where \( h_{t,j} \) with \( 1 \leq t \leq L_1, 1 \leq j \leq L_2 \) are chosen independently and uniformly at random from the family of hash functions. The data structure for searching for points with high inner product is constructed by taking each point \( x \) in the search space and storing it in the bucket \( g_j(x), 1 \leq j \leq L_2 \). When a new query point \( q \) is received, \( g_j(q), 1 \leq j \leq L_2 \) are calculated and all the points from the search space in the buckets \( g_j(q), 1 \leq j \leq L_2 \) are retrieved. We then compute the inner product of these points with the query vector and return the ones which have inner product greater than the specified threshold \( D \).

Let \( P_1 = 1 - \frac{\cos^{-1}(D)}{\pi} \). The probability that any point \( x \) such that \( q \cdot x \geq D \) is retrieved is at least \( 1 - (1 - P_1)^{L_2} \). Thus for any desired error probability \( \delta > 0 \), we can choose \( L_1 \) and \( L_2 \) such that \( x \) is returned by the algorithm with probability at least \( 1 - \delta \).

3 New Algorithms for Assortment Planning

In this section, we propose three algorithms: ASSORT-MNL, ASSORT-MNL\textit{(Approx)} and ASSORT-MNL\textit{(BZ)}. ASSORT-MNL aims to find an assortment with revenue within \( \epsilon \) of the optimal assortment. ASSORT-MNL\textit{(Approx)} is an approximate extension of ASSORT-MNL. And ASSORT-MNL\textit{(BZ)} extends ASSORT-MNL by providing error guarantees for its approximate solution.

3.1 First Algorithm: ASSORT-MNL

This algorithm (outlined in Algorithm 1) aims to find an \( \epsilon \)-optimal assortment i.e. an assortment with revenue within a small interval (defined by tolerance parameter \( \epsilon \)) of the optimal assortment’s revenue.
We search for the revenue maximizing assortment as follows. First, the optimal assortment is initialized to the set \( \{1\} \). This initial assortment will be returned as the optimal only when all the assortments have revenue less than \( \epsilon \) and in that case this assortment is approximately optimal. The search space starts with the lower and upper bounds \( (L_1, U_1) \) as 0 and \( p_1 \) respectively, where \( p_1 \) is the highest price among all items. In every iteration of the algorithm, we narrow down the search space for the optimal revenue by a binary search update of the lower and upper bounds.

The strength of the algorithm is in its ability to perform the calculation \( K \leq \max_{S \in S} f(S, v) \) very quickly for any pre-specified constant \( K \). We refer to this step as the COMPARE-STEP (see line 4). In particular, this decision problem is transformed into a MIPS problem as follows. In every iteration, we need to check if there exists a set \( S \in S \) such that

\[
K \leq \frac{\sum_{i \in S} P_i v_i}{v_0 + \sum_{i \in S} v_i} \iff K \leq \frac{1}{v_0} \sum_{i \in S} v_i(p_i - K)
\]

This is equivalent to evaluating if \( K \leq \max_{S \in S} \frac{1}{v_0} \sum_{i \in S} v_i(p_i - K) \), allowing us to focus on the optimization problem:

\[
\arg \max_{S \in S} \sum_{i \in S} v_i(p_i - K).
\]  

(1)

Define the set \( U \) as \( U = \{u^S = (u_1, u_2, \cdots u_n) \mid u_i = 1\{i \in S\}, S \in S\} \), where \( 1\{\cdot\} \) represents the indicator function \( (u^S) \) is the characteristic vector of the set \( S \).

Let \( (p - K) \) be the vector in \( \mathbb{R}^n \) with its \( i \)th component as \( p_i - K, \ i = 1, 2, \cdots n \). Define the set \( Z(K) = \{z^S | z^S = u^S \circ (p - K), u^S \in U\} \) where \( a \circ b \) represents the Hadamard product between the vectors \( a \) and \( b \). Thus, the optimization problem (1) can be written as \( \arg \max_{z^S \in Z(K)} v \cdot z^S \). This problem is finding the vector \( z^S \) which has the highest inner product with \( v \), which is a MIPS problem. The number of points in the search space is \( N = |S| \). The reduction of the compare step to a MIPS problem is summarized in Algorithm 2.

Once we find a solution to the MIPS problem, the value of the objective is compared with \( K \). This comparison narrows the search space for the optimal revenue, and the entire process is repeated till the size of the search space is below the required tolerance level (\( \epsilon \)).
Algorithm 2 Compare step in Assort-MNL

\[ p - K = (p_1 - K, \ldots, p_n - K) \]
\[ Z(K) = \{ z^S | z^S = u^S \circ (p - K), u^S \in U \} \]
\[ z^S = \arg \max_{x^S \in Z(K)} v \cdot z^S \]
\[ K \leq \max_{S \in S} f(S, v) \equiv K \leq \frac{v^T z^S}{v_0} \]

Also, note that MIPS problem instances can be solved quite efficiently in practice [13], either exactly or approximately.

3.2 Second Algorithm: Assort-MNL(Approx)

In Assort-MNL any sub-routine which solves the MIPS problem can be used. In particular, approaches which solve MIPS approximately can also be used. Such approaches achieve runtime performance gains as they are typically faster while trading off accuracy (see Section 4).

We now give details of an LSH based implementation for solving the MIPS problem approximately, primarily due to its superior performance in high dimensions and worst case runtime guarantees.

**Approximation in Assort-MNL(Approx):** To solve the MIPS problem introduced in the Assort-MNL algorithm, we need the nearest point to \( v \) according to the inner product metric. For this, we create multiple hash structures as described in Section 2 using different threshold values but with same success probability \( \delta \) (by choosing appropriate values of \( L_1 \) and \( L_2 \) for each of them). When a query vector is received, we calculate the near-neighbors using the hash structure with the largest threshold. We continue checking with decreasing value of thresholds till we find a near neighbor. Let \( \tilde{D} \) be the first threshold for which there is at least one near neighbor. This means that for the nearest neighbor, the inner product with the query vector is greater than \( \tilde{D} \). This further implies that the probability that we don’t find the true nearest neighbor is at most \( \delta \). We assume that the different radii in the hash structures are such that the number of points returned for the threshold \( \tilde{D} \) is sublinear in \( N \) i.e. \( O(N^\eta) \) where \( \eta < 1 \). This assumption is reasonable as long as the different radii are not very far apart.

We denote the above operation of finding approximate nearest neighbor as \( \text{approx arg max} \). More precisely, \( \text{approx arg max} (q, B) \) returns a vector from the set \( B \). This is the true nearest neighbor (i.e. has the highest inner product among all points in set \( B \)) to the query vector \( q \) with probability \( 1 - \beta \), where \( \beta \) is the probability of error. For the LSH structure described above, \( \beta \leq \delta \). The Assort-MNL(Approx) algorithm is then, the same as Assort-MNL except the operation \( z^S = \arg \max_{x^S \in Z(K)} v \cdot z^S \) is replaced by \( z^S = \text{approx arg max}_{x^S \in Z(K)} v \cdot z^S \).

Because we are doing binary search comparisons, the LSH data structures described above need to be created for all possible values of threshold \( K \) we can encounter, which are \( O \left( \frac{p_1}{\epsilon} \right) \) in number. Although these structures can be created as part of pre-processing as they don’t require the knowledge of the query vector \( v \), they have a significantly high space requirement. We can reduce the space complexity using the following transformation to end up using a
single set of LSH data structures instead of one set for every threshold $K$.

Define vector $\hat{v}_K$ in $\mathbb{R}^{2n}$ as $\hat{v}_K = (v_1, \ldots, v_n, -v_1 K, -v_2 K, \cdots - v_n K)$. For a given set $S$, let $\hat{z}^S = (\hat{p} \circ \hat{u}^S, \hat{u}^S)$ where $\hat{u}^S$ is the characteristic vector of set $S$ and $\hat{p} = (p_1, p_2 \cdots p_n)$. We then define $\hat{Z} = \{\hat{z}^S : S \in S\}$. It is easy to see that for every $z \in Z(K)$, there exists $\hat{z} \in \hat{Z}$ such that $v \cdot z = \hat{v}_K \cdot \hat{z}$. The new LSH structure now depends on the vectors in the search space $\hat{Z}$, which is now independent of $K$. Thus, by doubling the dimension of the search space, we can reduce the space complexity of Assort-MNL(Approx) by a factor of $2^{\lceil \log_2 \frac{\epsilon}{\epsilon'} \rceil}$ which can be significant when the ratio $\frac{p_1}{\epsilon}$ is large. The space complexity of this efficient implementation is $O(N^2 + \rho)$ and is summarized in Algorithm 3.

Algorithm 3 ASSORT-MNL(Approx) (efficient)

Require: Prices $\{p_i\}_{i=1}^n$, tolerance parameter $\epsilon$

$L_1 = 0$, $U_1 = p_1$, $t = 1$, $p = (p_1, \cdots, p_n)$

$S^* = \{1\}$, $\hat{Z} = \{\hat{z}^S : \hat{z}^S = (\hat{p} \circ \hat{u}^S, \hat{u}^S), S \in S\}$

while $U_t - L_t > \epsilon$ do

$K = \frac{L_t + U_t}{2}$

$\hat{v}_K = (v_1, \cdots, v_n, -v_1 K, -v_2 K, \cdots - v_n K)$

$\hat{z}^S = \text{approx } \text{arg } \text{max} (\hat{v}_K, \hat{Z})$

if $K \leq \frac{\hat{v}^S}{v_0}$ then

$L_{t+1} = \frac{L_t + U_t}{2}, U_{t+1} = U_t$, $S^* = \hat{S}$

else

$L_{t+1} = L_t, U_{t+1} = \frac{L_t + U_t}{2}$

return $S^*$

3.3 Third Algorithm: Assort-MNL(BZ)

In the binary search routine of ASSORT-MNL, we narrow our search space based on the result of the MIPS query. But when an approximate method is used for solving the MIPS query, there is a chance of narrowing down the search space to an incorrect range. To address this, we modify the binary search process to the BZ process [5] to accommodate the possibility of receiving incorrect answers to the comparison query $K \leq \max_{S \in S} f(S, v)$. Note that the probability of receiving incorrect answer ($P_e$) to the comparison query $K \leq \max_{S \in S} f(S, v)$ is upper bounded by the probability of not being returned the true nearest neighbor, which is $\delta$.

The key difference in the BZ algorithm from a standard binary search is the choice of the decision threshold point at which the function is tested at each stage. In binary search, we choose the mid-point of the current search interval. As we cannot rule out any part of the original search space when we receive noisy answers, in the BZ algorithm, we maintain a distribution on the value of the optimal revenue. In every iteration, we test if $\hat{K} \leq f(S, v)$

---

1 A factor of 2 appears in the denominator as the space complexity depends on the product $dn$ where $d$ is the dimension of the search space.
where \( \hat{K} \) is the median of the distribution and then update the distribution based on the result of the comparison.

In summary, the ASSORT-MNL(BZ) algorithm given in Algorithm 4 is constructed based on the BZ algorithm [6] and the LSH data structures for approximating the MIPS queries.

**Error Analysis of Algorithm 4:** We claim that the probability of error for ASSORT-MNL(BZ) decays exponentially with the number of iterations of the algorithm. This is based on a similar result for the BZ algorithm [5]. Let \( Q_\epsilon = 1 - P_\epsilon \) and \( \beta = 1 - \alpha \), where \( \alpha < 0.5 \) is an upperbound on the true error probability \( P_\epsilon \). After \( T \) iterations, we have

\[
P(|\hat{\theta}_T^* - \theta^*| > \epsilon) \leq \frac{p_1 - \epsilon}{\epsilon} \left( \frac{P_\epsilon}{2\alpha} + \frac{Q_\epsilon}{2\beta} \right)^{T^*},
\]

where \( \theta^* \) is the true optimal revenue and \( \hat{\theta}_T^* = f(S^*, \mathbf{v}) \). This bound\(^2\) depends on the unknown quantity \( P_\epsilon \), so we do the following modification when only an upperbound on its value (say \( P_{\text{max}} \)) is known. Let \( W(P_\epsilon, \alpha) = \frac{P_\epsilon}{2\alpha} + \frac{Q_\epsilon}{2\beta} \). This is a linear and increasing function of \( P_\epsilon \). Thus, \( W(P_\epsilon, \alpha) \leq W(P_{\text{max}}, \alpha) \). By appropriately tuning the approximate query oracle (always possible), let \( P_{\text{max}} < 0.25 \). Choosing \( \alpha = \sqrt{P_{\text{max}}} \) (a valid upperbound on \( P_\epsilon \)), we get \( W(P_{\text{max}}, \sqrt{P_{\text{max}}}) = 0.5 + \sqrt{P_{\text{max}}} \). Thus, we have

\[
P(|\hat{\theta}_T^* - \theta^*| > \epsilon) \leq \frac{p_1 - \epsilon}{\epsilon} \left( \sqrt{P_{\text{max}}} + 0.5 \right)^{T^*}.
\]

With the above manipulation, the number of iterations required to get to a desired reliability level can now be estimated. If the desired confidence level is \( \gamma \) i.e., \( P(|\hat{\theta}_T^* - \theta^*| > \epsilon) \leq \gamma \), then \( T^* \geq -\log_{0.5+\sqrt{P_{\text{max}}}} \frac{p_1 - \epsilon}{\epsilon} \). Thus, the number of iterations grows logarithmically in the desired accuracy level. Finally, note that for the LSH based approx arg max calculation \( P_{\text{max}} = \delta \).

### 3.4 Time Complexities of Proposed Algorithms

In ASSORT-MNL, the MIPS query in every iteration can be solved in time \( O(nN) \) where \( n \) is the dimension of the vectors (equals the number of items) and \( N \) is the number of points in the search space (equals the number of feasible sets). In every iteration, we cut down the search space for the optimal revenue by half. We start with a search space of range \([0, p_1]\). Thus, the number of iterations to get to the desired tolerance is \( \lceil \log \frac{p_1}{\epsilon} \rceil \). Thus, the time taken by the algorithm is \( O(nN \log \frac{p_1}{\epsilon}) \).

In ASSORT-MNL(Approx), it takes \( O(nN^\rho) \) time to run the approximate near neighbor query. After retrieving the near neighbors, the nearest neighbor is calculated by finding the highest inner product value among all the near neighbors which takes \( O(nN^\rho) \) time. Thus, the time complexity is \( O(nN^{\max\{\rho, \eta\}} \log \frac{p_1}{\epsilon}) \). The space complexity of the efficient version is \( O(nN^{(1+\rho)}) \). Note that the runtime is sub-linear in \( N \) as \( \rho, \eta < 1 \).

Similarly, the time complexity of ASSORT-MNL(BZ) is \( O(nN^{\max\{\rho, \eta\}} \log \frac{p_1 - \epsilon}{\gamma \epsilon}) \) and the space complexity is \( O(nN^{(1+\rho)}) \). As a benchmark, an exhaustive search over all feasible sets

\(^2\) Note that when ASSORT-MNL(BZ) is run for \( T \) iterations, the convergence result is not expressed in terms of \( T \) but in terms of \( T^* \), where \( T^* \) is the last iteration where an assortment with a revenue greater than the specified threshold \( K_j \) is found.
Algorithm 4 ASSORT-MNL(BZ)

Require: Prices \( \{p_i\}_{i=1}^{n} \), tolerance parameter \( \epsilon \) such that \( p_1 \epsilon^{-1} \in \mathbb{N} \), number of steps \( T \), upper bound \( \alpha < 0.5 \) on the error probability in the approx arg max operation, \( \beta = 1 - \alpha \)

Posterior \( \pi_j : [0, p_1] \rightarrow \mathbb{R} \) after \( j \) stages:

\[
\pi_j(x) = \sum_{i=1}^{p_1 \epsilon^{-1}} a_i(j) 1_{I_i}(x),
\]

where \( I_i = [0, \epsilon] \) and \( I_i = (\epsilon(i-1), \epsilon i] \) for \( i \in \{2, \cdots, p_1 \epsilon^{-1}\} \).

Initialize \( a_i(0) = p_1^{-1} \epsilon \forall i, S^* = \{1\}, T^* = 1, j = 0 \).

while \( j < T \) do

Sample Selection: Define \( u(j) \) such that \( \sum_{i=1}^{u(j)-1} a_i(j) \leq \frac{1}{2} \), \( \sum_{i=1}^{u(j)} a_i(j) > \frac{1}{2} \). Let

\[
K_{j+1} = \begin{cases} 
 p_1^{-1} \epsilon(u(j) - 1) & \text{with probability } Q(j) \\
 p_1^{-1} \epsilon u(j) & \text{with probability } 1 - Q(j)
\end{cases}
\]

where \( Q(j) = \frac{\tau_2(j)}{\tau_1(j) + \tau_2(j)} \) and

\[
\tau_1(j) = \sum_{i=u(j)}^{p_1 \epsilon^{-1}} a_i(j) - \sum_{i=1}^{u(j)-1} a_i(j),
\]

\[
\tau_2(j) = \sum_{i=1}^{u(j)} a_i(j) - \sum_{i=u(j)+1}^{p_1 \epsilon^{-1}} a_i(j).
\]

Noisy Observation:

\( z^S = \text{approx arg max } (v, Z^i(K_{j+1})) \)

\( h(K_{j+1}) = 1 \{ K_{j+1} \leq \frac{z^S}{\epsilon_0} \} \)

If \( h(K_{j+1}) = 1 \), \( S^* = \hat{S} \) and \( T^* = j \).

Update posterior: Note that \( K_{j+1} = p_1^{-1} \epsilon u, u \in \mathbb{N} \). For \( i \leq u \), we have

\[
a_i(j+1) = \begin{cases} 
 a_i(j) & \text{if } h(K_{j+1}) = 0 \\
 \frac{\alpha a_i(j)}{\sum_{o=1}^{u} a_o(j) + \alpha \sum_{o=u+1}^{p_1 \epsilon^{-1}} a_o(j)} & \text{if } h(K_{j+1}) = 1
\end{cases}
\]

For \( i > u \), we have

\[
a_i(j+1) = \begin{cases} 
 0 & \text{if } h(K_{j+1}) = 0 \\
 \frac{\alpha a_i(j)}{\sum_{o=1}^{u} a_o(j) + \alpha \sum_{o=u+1}^{p_1 \epsilon^{-1}} a_o(j)} & \text{if } h(K_{j+1}) = 1
\end{cases}
\]

return \( S^*, \hat{\theta}_{T^*} = f(S^*, v), T^* \)
$N$ would require $O(nN)$ time. Both ASSORT-MNL and the exhaustive search have the same
time complexity but we observe significant improvement in practical runtimes when using
ASSORT-MNL. Also, both these algorithms don’t require any additional storage except that
used for storing all the feasible sets. Thus, their space complexity is $O(nN)$.

3.5 Assortment Planning with Capacity Constraints

The above algorithms can be optimized further (both in terms of time and space complexity)
if we assume more structure on the constraints. Significant improvement can be obtained
in the capacity constrained setting i.e., when $S = \{S : |S| \leq C\}$, where constant $C$
specifies the maximum size of feasible assortments. The key insight is that in this case,
the operation $\arg \max_{z^i \in Z(K)} v \cdot z^i = \arg \max_{S \subseteq S} \sum_{i \in S} v_i (p_i - K)$ can be decoupled into
problems of smaller size. Define $\bar{U}$ be the set of canonical basis vectors i.e. $\bar{U} = \{\bar{u} = (u_1, u_2, \cdots, u_n) | u_i \in \{0, 1\} \forall i, \sum_{i=1}^n u_i = 1\}$. Let $(p - K)$ be the vector in $\mathbb{R}^n$ with its $i$th component as $p_i - K$, $i = 1, 2, \cdots, n$. Define the set $\bar{Z}(K) = \{\bar{z} | \bar{z} = \bar{u} \circ (p - K), \bar{u} \in \bar{U}\}$ where $\circ$ represents the Hadamard/element-wise product between the vectors $a$ and $b$.

Then the optimization problem 1 can be equivalently stated as

$$\arg \max_{\{z^1, z^2, \cdots, z^l : z^i \in \bar{Z}(K), i = 1, \cdots, l, \ l \leq C, z^1 \neq z^2 \neq \cdots \neq z^l\}} \sum_{i=1}^l v \cdot z^i.$$ 

Notice that this problem is finding a set of at most $C$ vectors that have the highest inner
product with $v$ and the value of the inner product is positive. Thus, this is a top-$C$ variant of
the MIPS problem i.e., the problem of finding the top-$C$ points according to the inner product
metric with the query vector. The main advantage of this reformulation is that the number
of points in the search space is now $n$ instead of $N = O(n^C)$. Similar to the approx arg max
operation, we define approx $C$ arg max $(q, B)$ which returns a set of $C$ vectors from the set $B$.
This set is the set of top $C$ nearest neighbors to the query vector $q$ with probability $1 - \beta_C$,
where $\beta_C$ is the probability of error. For the LSH based implementation, $\beta_C \leq 1 - (1 - \delta)^C$.
Once we find a solution to this top-$C$ MIPS problem, the value of the objective is compared
with $K$ and the algorithm proceeds as before. A memory efficient implementation for this
algorithm is summarized in Algorithm 5.

Using the LSH structures described before, the time taken to find top $C$ nearest neighbors
in $\mathbb{R}^n$ is also $O(n^{1+\rho})$. With this implementation, the time complexity of ASSORT-MNL,
ASSORT-MNL(APPROX) and ASSORT-MNL(BZ) become $O(n^2 \log \frac{p_N}{\epsilon}), O(n^{1+\max(\rho, \eta)} \log \frac{p_N}{\epsilon})$ and $O(n^{1+\max(\rho, \eta)} \log \frac{p_N}{\gamma \epsilon})$. Note that other algorithms for solving the capacitated assortment
planning problem like ADXOpt and STATIC-MNL have time complexity quadratic in $n$
whereas ASSORT-MNL(APPROX) and ASSORT-MNL(BZ) have sub-quadratic dependence
on $n$.

The above modification can be extended to some other capacity-like constraints as
described below: (1) Lower bound on assortment size - This constraint requires that all
feasible assortments have at least $c$ items in addition to having at most $C$ items. In every
iteration of Algorithm 5, we find the top $C$ points (by the inner product metric) which give
a positive value of the inner product. This can be modified to finding the top $C$ neighbors
and including at least the top $c$ of them irrespective of the sign of the inner product i.e.
We empirically validate the runtime performance of two of the proposed algorithms, namely Algorithm 5 and Assort-MNL with 3.6GHz) with python 2.7 (we also use LSH Forest [14] and NearestNeighbors from Scikit-performance comparisons.  

4 Experiments

We empirically validate the runtime performance of two of the proposed algorithms, namely ASSORT-MNL and ASSORT-MNL(APPROX) using real and synthetic datasets\textsuperscript{3}. For the case of general assortments, we compare these against an algorithm which performs exhaustive search. And for the case of cardinality constrained assortments, we compare these with other algorithms such as Static-MNL [15], ADXOpt [10] and a linear programming (LP) formulation [8]. All experiments are run on a 6 core 64GB 64-bit intel machine (i7-6850K 3.6GHz) with python 2.7 (we also use LSH Forest [14] and NearestNeighbors from Scikit-learn as well as CPLEX 12.7 for solving the LP). Note that LSH Forest performs similarity

\textsuperscript{3}Because we expect ASSORT-MNL(BZ) to perform similar to ASSORT-MNL(APPROX), we omit its performance comparisons.

\begin{algorithm}[H]
\caption{ASSORT-MNL(APPROX) (efficient) for the capacity-constrained setting}
\begin{algorithmic}
\Require\hspace{1cm}Prices \(\{p_i\}_{i=1}^n\), tolerance parameter \(\epsilon\)
\State \(L_1 = 0, U_1 = p_1, t = 1, S^* = \{1\}, \hat{U} = \{u = (u_1, u_2, \cdots \cdot u_n)|u_{n+i} = u_i, i \in 1, \cdots n, u_i \in \{0,1\}\forall i, \sum_{i=1}^n u_i = 1\}\), \(\hat{p} = (p_1, \cdots \cdot p_n, 1, 1, \cdots 1)\), \(\hat{Z} = \{\hat{z}|\hat{z} = u \odot \hat{p}, \hat{u} \in \hat{U}\}\)
\While \(U_t - L_t > \epsilon\) do
\State \(K = \frac{L_t + U_t}{2}\)
\State \(\hat{v}_K = (v_1, \cdots \cdot v_n, -K, -K, \cdots \cdot -K), \hat{v}_K \in \mathbb{R}^{2n}\)
\State \(A = \text{approx}\ C \arg\max \ (\hat{v}_K, \hat{Z})\)
\State \(A = \{\hat{z}: \hat{z} \in \hat{A}, \hat{v}_K \cdot \hat{z} > 0\}\)
\If \(K \leq \frac{1}{v_0} \sum_{z \in A} v \cdot z\) then
\State \(L_{t+1} = \frac{L_t + U_t}{2}, U_{t+1} = U_t, S^* = \{i|z_i \neq 0, z \in A\}\)
\Else
\State \(L_{t+1} = L_t, U_{t+1} = \frac{L_t + U_t}{2}\)
\EndIf
\EndWhile
\State \Return \(S^*\)
\end{algorithmic}
\end{algorithm}
search based on the cosine distance. Thus, in order to use it for MIPS that is needed in \textsc{Assort-MNL(Approx)}, we transform our query and points before indexing using the lifting trick [13] as described in Section 2. Performance is measured in terms of the mean computational time (sec), mean relative error in the revenue obtained as well as the mean overlap between the assortment output by our algorithms and the optimal assortment output by either an exhaustive search or some other exact method.

4.1 Datasets

We use two different types of real data sets, one as source for real prices and the second as a source for general assortments. For prices, we use the publicly shared online micro price dataset from the Billion Prices Project [7] to generate item prices. This dataset contains daily prices for all goods sold by 7 large retailers in Latin America (3 retailers) and the USA (4 retailers) between 2007 to 2010. Among the US retailers, we use pricing data from a supermarket and an electronics retailer to generate our assortment planning instances of varying sizes. The former contains 10 million daily observations for 94 thousand items and the latter contains 5 million daily observations for 30 thousand items. We use prices from 50 different days when generating 50 instances of Monte Carlo runs under different settings described below. The collection of assortments in these instances are either general or capacitated. The MNL parameters (each coordinate of \(v\)) are chosen from the uniform distribution \([0, 1]\).

For generating feasible assortments, we use publicly available transaction datasets used in frequent itemset mining [4]. In particular, we use the retail, foodmart, chainstore and e-commerce transaction logs [9] to create collections of general assortments. We use the FPgrowth algorithm from the spmf [9] program to first generate frequent itemsets with appropriate minimum supports and then prune out frequent itemsets with low cardinalities to obtain our collection of assortments. Table 1 describes some statistics of the assortments generated. We again create 50 instances from each dataset by generating the price and the MNL parameter vectors using uniform distributions \([0, 1000]\) and \([0, 1]\) respectively.

| Dataset      | retail | foodmart | chainstore | e-commerce |
|--------------|--------|----------|------------|------------|
| Number of transactions | 88162  | 4141     | 1112949    | 540455     |
| Number of items       | 3160   | 1559     | 321        | 2208       |
| Number of general assortments | 80524  | 81274    | 75853      | 23276      |
| Size of largest assortment | 12     | 14       | 16         | 8          |
| Size of smallest assortment | 3      | 4        | 5          | 3          |

4.2 General Assortments

In the first setting, we use the assortments that were obtained using frequent itemset mining and post-processing (to remove assortments of low cardinality) in order to compare
the performance of ASSORT-MNL, ASSORT-MNL(Approx) and exhaustive search. The tolerance parameter $\epsilon$ is set to 0.1 for ASSORT-MNL as well as ASSORT-MNL(Approx) in this and all subsequent experiments. There are two parameters for the underlying LSH Forest [14] subroutine for ASSORT-MNL(Approx): (a) number of candidates (set to 80), and (b) number of estimators (set to 20). These parameters govern the accuracy and speed of the approximate similarity search operation. The results are given in Figure 1 (mean values across 50 runs are reported). As can be inferred from the plots, ASSORT-MNL(Approx) is 2-3× faster than exhaustive search and also better than ASSORT-MNL without sacrificing much of the revenue. As mentioned before, these instances cannot be efficiently handled by either integer programming formulations (unless there is an efficient representation of the feasible assortments) or by other specialized approaches.

Figure 1: Performance of ASSORT-MNL and ASSORT-MNL(Approx) over instances derived from four different frequent itemset datasets.

In the second setting, we use the pricing data from the Billion prices project and generate a fixed number of assortments from the set of all assortments uniformly at random. In particular, we vary the number of assortments to be from the set \{100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200\}. Again, we report results averaged over 50 Monte Carlo runs, as shown in Figure 2. We run multiple versions of ASSORT-MNL(Approx), with the number of candidates parameter and the number of estimates parameter ranging over sets \{80, 160, 200\} and \{20, 40, 100\} respectively for the underlying LSH Forest [14] subroutine. As can be observed, our proposed algorithms are much better than exhaustive search under all three performance metrics. In particular, we can trade off speed (timing performance) of ASSORT-MNL(Approx) with accuracy (lower percentage of assortment set overlap). We obtained qualitatively similar performances for synthetic data (prices uniformly generated from the interval $[0, 1000]$) and omit these results here due to space constraints.

4.3 Cardinality-constrained Assortments

In this experiment, we explore how our general purpose algorithms fare as compared to the specialized algorithms viz., ADXOpt, LP and Static-MNL (exhaustive search is not considered because its performance is an order of magnitude worse than all these methods). We run experiments with both synthetically generated instances as well as instances generated with prices from the Billion prices dataset (to capture real price distributions), and observe similar qualitative results. The cardinality parameter is chosen to be 50 irrespective of the number of items. Figure 3 shows the performance of our algorithms (ASSORT-MNL(Approx) was run
Figure 2: Performance of Assort-MNL and Assort-MNL(Approx) over instances derived from the Billion prices dataset. The x-axis corresponds to the number of feasible assortments (uniformly sampled).

with number of estimators and candidates for LSH Forest to be equal to 20 and 80 respectively). The number of items was varied from the set \{100, 250, 500, 1000, 3000, 5000, 7000, 10000, 20000\}. We can observe that Assort-MNL(Approx) runs much faster than both Assort-MNL and LP. In terms of the accuracy of the results, Assort-MNL(Approx) produces assortments that look different from the optimal assortment. Nonetheless, the percentage loss in relative revenue is very close to 0. Note that the accuracy can be improved at the cost of timing performance, just as in the previous setting. We have plotted the timing performance of ADXOpt separately in Figure 4, as its running time (plotted as intensity) varies quite a bit as a function of the size of the optimal assortment. If the instance happens to have a small optimal assortment, ADXOpt can get to this solution very quickly. On the other hand, it spends a lot of time when the optimal assortment size is large. This is illustrated through an intensity plot as opposed to a one-dimensional mean timing plot. Similarly, we plot the timing performance of Static-MNL separately in Figure 4 because its performance is much worse than all the other methods even for moderate sized instances.

Figure 3: Performance of Assort-MNL and Assort-MNL(Approx) over instances derived from the Billion prices dataset. The x-axis corresponds to the number of items.

In summary, our algorithms are competitive with the state of the art algorithms, viz., ADXOpt, Static-MNL and LP even though these algorithms are specialized for the case of cardinality constraints, whereas our algorithms can handle any type of constraints over feasible assortments even when there is no efficient representation of the corresponding polytope. Further, the proposed algorithms vastly increase the assortment planning problem instances that can be solved efficiently under the MNL choice model. For instance, we show computational results when instances have the number of items of the order of $\sim 10^5$ easily,
Figure 4: Performance of ADXOpt (top) and Static-MNL (bottom) over instances derived from the Billion prices dataset. The intensity (time in seconds) shows that the performance of ADXOpt is highly dependent on the size of the optimal assortment. Static-MNL’s timing performances are an order of magnitude worse even for moderate sized instances.

whereas the regime in which experiments of the current state of the art methods [3] were carried out is with $n \sim 10^3$, thus representing two orders of magnitude improvement.

5 Concluding Remarks

We proposed multiple efficient algorithms that solve the assortment optimization problem under the Multinomial Logit (MNL) purchase model even when the feasible assortments cannot be compactly represented. In particular, we motivate how frequent itemsets can be used as candidate assortments, making the planning problem data-driven. Our algorithms are iterative and build on binary search and fast methods for maximum inner product search to find optimal solutions for large scale instances. Though the $\sim 10x$ improvement is a significant gain, there is scope for extending this work in many ways. For instance, studies by psychologists have revealed that buyers are affected by the assortment size as well as how frequently they change over the course of their interactions. In summary, such scalable methods for assortment planning, and revenue management in general, can have material impact on facilitating personalized interactions between sellers and buyers.

Acknowledgements

The authors would like to thank Dr. Vivek Farias (MIT) for initial discussions on this research.

References

[1] S. Agrawal, V. Avadhanula, V. Goyal, and A. Zeevi. A near-optimal exploration-exploitation approach for assortment selection. In ACM Economics and Computation, 2016.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM, 51(1):117, 2008.
[3] D. Bertsimas and V. V. Mišic. Data-driven assortment optimization. Technical report, MIT Sloan School, 2015.

[4] C. Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):437–456, 2012.

[5] M. V. Burnashev and K. Zigangirov. An interval estimation problem for controlled observations. *Problemy Peredachi Informatsii*, 10(3):51–61, 1974.

[6] R. Castro and R. Nowak. Active learning and sampling. In *Foundations and Applications of Sensor Management*, pages 177–200. Springer, 2008.

[7] A. Cavallo. Scraped Data and Sticky Prices. Review of Economics and Statistics, Forthcoming, 2016.

[8] J. Davis, G. Gallego, and H. Topaloglu. Assortment planning under the multinomial logit model with totally unimodular constraint structures. *Tech Report. Department of IEOR, Columbia University*, 2013.

[9] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, V. S. Tseng, et al. Spmf: a java open-source pattern mining library. *Journal of Machine Learning Research*, 15(1):3389–3393, 2014.

[10] S. Jagabathula. Assortment optimization under general choice. *Available at SSRN 2512831*, 2014.

[11] A. G. Kök, M. L. Fisher, and R. Vaidyanathan. Assortment planning: Review of literature and industry practice. In *Retail supply chain management*, pages 99–153. 2008.

[12] R. D. Luce. Individual choice behavior, a theoretical analysis. *Bulletin of the American Mathematical Society*, 66:259–260, 1960.

[13] B. Neyshabur and N. Srebro. On symmetric and asymmetric LSHs for inner product search. In *32nd International Conference on Machine Learning*, pages 1926–1934, 2015.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

[16] K. E. Train. *Discrete choice methods with simulation*. Cambridge University Press, 2009.