MANUFACTURING & SERVICE OPERATIONS MANAGEMENT
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 1523-4614 | EISSN 1526-5498 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

Assortment Optimization and Pricing under a Nonparametric Tree Choice Model

Alice Paul
Operations Research and Information Engineering, Cornell University, Ithaca, NY 14850, ajp336@cornell.edu

Jacob Feldman*
Olin Business School, Washington University, St. Louis, MS 63108, jbfeldman@wustl.edu

James Mario Davis
Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Champaign, IL 61801, jamesmariodavis@gmail.com

We consider assortment and pricing problems when customers purchase products according to a nonparametric choice model. Each customer arrives with a preference list and will purchase the highest-ranking offered product in her preference list. We assume the set of customer classes is derived from paths in a tree, in which the order of nodes visited along each path gives the corresponding preference list. First, we study assortment problems, in which the goal is to find which products to offer to maximize expected revenue. We give a dynamic programming solution, which can be extended to versions of the assortment problem in which there are fixed costs for offering a product, shelf constraints, or substitution costs. Second, we study the joint assortment and pricing problem, in which the goal is to simultaneously select the set of offered products as well as their prices. We solve the pricing problem optimally when customers have some universal ranking of the products, and hence the tree takes the form of a single path. We also solve the problem optimally on the general tree when the prices are restricted to be quality consistent; higher quality products must be priced above lower quality products. Lastly, we present computational experiments on both synthetic data and real hotel purchase data. Our estimation procedure shows both how to build the tree of products and how to estimate the underlying arrival probabilities of each customer type from historical sales data. These experiments show that the tree choice model captures customer purchasing behavior more accurately than the multinomial logit choice model in the majority of test cases.

Key words: customer choice models, dynamic programming, assortment optimization, price optimization

1. Introduction

The ability to accurately model customer demand is critical for any retailer since this model will guide everything from inventory decisions to pricing to promotion strategies. Customer choice models provide a way to model this demand; a customer choice model maps any

* Corresponding author.

assortment of available products to the probabilities the products in the assortment are purchased. Through these models, we can also capture how a product's features affect its attractiveness and hence its probability of being purchased. A common feature that is considered is price, since it influences both demand and profit margins. Further, since the prices of each product can be varied after the products have been produced, it is a straightforward lever to optimize profits. A variety of customer choice models exist, each capturing the effects of substitution and price sensitivities differently. An ideal choice model is one which is simple to describe, easy to estimate, and whose corresponding revenue management problems admit tractable solutions.

In this paper, we consider assortment and pricing problems when customers choose among the offered products according to a special case of the full nonparametric ranking-based choice model dating back to Mahajan and van Ryzin (2001a) and Mahajan and van Ryzin (2001b). In the full nonparametric ranking-based choice model, each customer class is distinguished by an arrival probability and a unique ranking on a subset of products. For the remainder of this paper, we use the term preference list to refer to the unique ranking of the products associated with a customer class. Further, we use the term customer class and customer type synonymously. In this full model, there are no restrictions on the set of potential preference lists and hence the number of customer types grows exponentially in the number of products. When presented with an assortment of products, a customer will purchase the highest ranking offered product in her preference list, and if there is no offered product in her preference list, then she leaves without making a purchase.

In the pricing setting, customer classes will also have budgets, and will purchase the highest ranking product in their preference list that is priced within their respective budget. We note that in modeling price sensitivity in this manner, we assume that prices only play a role in determining the consideration set of each customer. We use the term consideration set to refer to the subset of products that a customer of a particular type would ever be interested in purchasing. From a random utility maximization perspective, this amounts to the assumption that prices only influence the utility that a customer associates with each product in a binary manner: if a product is priced above the budget of a given customer, then this customer will associate a utility of zero with this product, otherwise, the associated utility can be viewed as a function of the other features of the product and is not influenced by price.

This modeling assumption is supported by the work of Gilbride and Allenby (2004), who study a two-stage choice model in which consumers first form consideration sets based on screening rules for the attributes of products, and then proceed to purchase the product with the highest utility within their consideration set. The authors find that choice models built on the groundwork of conjunctive screening rules, under which a product makes it into a consumer's consideration set only if it is found acceptable with regards to all relevant attributes, fit the data the best. The idea of a budget threshold is singled-out as one such attribute that could form the basis of a conjunctive screening rule. Further, using survey data on camera purchases, they find that price and body style play an important role in determining the consideration set, but not in the final choice from among the offered products. It is not hard to imagine that customers would exhibit similar behavior when purchasing other leisure pieces of technology such as televisions, which have distinguishing features similar to those of a camera. In general, this pricing assumption seems to capture settings in which customers purchasing a mid-range item have a threshold discount at which they will substitute to a higher-end product.

The full nonparametric choice model has the ability to generalize any random utility choice model; however this modeling flexibility comes at a cost. Due to the potentially large number of customer types, it can be difficult to estimate the underlying arrival probabilities of each customer type and to provide tractable algorithms for the accompanying revenue management problems. For example, under the general nonparametric choice model, there is no efficient algorithm to determine the assortment that maximizes a retailer's expected revenue, a fundamental problem. The model we present and study places structure on the set of allowable preferences lists in a manner that always guarantees a manageable number of customer types. To be more precise, given an undirected tree in which each node corresponds to a unique product, the set of all possible customer types is characterized by the set of all paths in the tree. We restrict these paths to be linear in the sense that they must either move progressively towards or away from the root node. We fully formalize the notion of a linear path as well as the model as a whole in Section 2.

The general tree model that we present should be viewed as a generalization of the intree and outtree models introduced in Honhon et al. (2012). These models are similar to ours in that customer classes correspond to paths in an underlying tree, but they also have the critical restriction that each path must include the root node. This restriction

ultimately leads to significant shortcomings in the practicality of these two models. By allowing preference lists to be associated with arbitrary linear paths, we alleviate many of the shortcomings of the intree and outtree models, and as a result, we are able to capture a broader range of retail settings.

The intree model is appropriate when customers substitute from specific, specialized products to more general products. General products that are designed to appeal to a wide range of customers would be located towards the root of the tree, with the root being the most general. Products targeted to specific customer segments would be located towards the leaves of the tree, with the leaves being the most targeted products. The critical limitation of the intree model is that all preference lists include the root product, implying in our example, that all customers are willing to substitute to the most general product type. First, this means that if the product corresponding to the root node is made available for purchase, then every arriving customer will make a purchase. In settings in which there are high numbers of no-purchase events, such as e-commerce, it is likely that the intree model will fail to explain large portions of the sales data. Further, in assuming that all customer types substitute down to the root, the intree model is unable to capture any differentiation in pickiness within the customer population. In Honhon et al. (2012), the motivating example for the intree shows how it can be used to model customers purchasing various shampoos. The all-purpose shampoo is placed at the root, while shampoos targeted at very specific hair types are located at the leaves. The intree model assumes that all customers are willing to substitute to the all-purpose shampoo if their preferred, more targeted product is unavailable. In contrast, since the general tree model allows for preference lists that can end anywhere in the tree, it allows us to capture the purchasing behavior of customers who will leave the store without making a purchase if they cannot find their desired targeted product.

In the outtree model, all customer classes are associated with paths that begin at the root node and terminate at an interior or leaf node. The outtree model is appropriate when customers substitute from products with many features to products with less robust feature sets. This is the case, for example, in a product line that is targeted to a wide range of consumer budgets, with more expensive products having richer feature sets. Expensive products with many features would be located towards the root of the tree, with the most feature rich product at the root node. Less expensive products with less rich feature sets

are located toward the leaves of the tree, with the least feature rich located at the leaves. The outtree model has similar limitations to the intree model: all preference lists include the root product as the highest ranked product, implying that all customers will purchase this product if it is offered. For customers who are budget conscious, this is clearly an unrealistic assumption. Again, since paths associated with preference lists can start at any node in the general tree model, our model captures the various budgets associated with different segments of customers.

Building on these motivating examples for the intree and outtree, we note that it is best to use the general tree model in settings in which customers have monotonic preferences for features in a product line, and the retailer would like to understand how customers trade off between various combinations of these features when making a purchase. One example of a setting in which the general tree model is likely suitable is in describing customers purchasing iPhones; newer editions are preferred to older ones and more memory is more desirable than less memory. Another setting in which the general tree model is applicable is hotel bookings. Here, customers consider the trade-off between features such as price, bed size, square footage, and the presence or absence of beautiful views among others. In these cases, the structure of the underlying tree provides insights into how customers value these features. However, in most scenarios, including the hotel example that we study, it is unclear exactly how the products should be ordered in the tree. Hence for the tree model to be practically useful, it is essential that the tree structure can be teased out from sales data and that we can solve the common revenue management problems that arise in a tractable fashion. In the remainder of this paper, we show how to accomplish both of these tasks. The combination of efficient estimation procedures for the model and tractable algorithms for the optimization problems allows the tree model to form a practical basis of revenue management systems.

Contributions.

First, we consider assortment optimization problems under the general tree model. In the assortment optimization problem, the retailer is presented with a collection of products from which she must choose an assortment of products to offer to customers so as to maximize expected revenue. In this case, we show that the assortment problem can be solved with a dynamic program. This dynamic program has a small state space, which leads to efficient optimization. The key insight that we make in the dynamic program boils down

to the idea that the purchase probability of any item within an arbitrary assortment can be computed recursively with only knowledge of each product's closest offered predecessor in the tree. The dynamic program for the pure assortment problem can be extended to settings in which the retailer has additional cost considerations. We consider scenarios in which there are fixed costs to include products in the offered assortment and penalties when a customer is forced to substitute to a less preferred product. Substitution penalties model a loss of customer good will, a common consideration for retailers. Finally, we extend the dynamic program to the cardinality constrained assortment optimization problem. In this problem, the available products are grouped into categories and the retailer can offer a limited number of products from each category. In the simplest case, all products are in a single category and the retailer is constrained to have an assortment of limited size.

The second problem that we consider has come to be known as the joint assortment and pricing problem. In this problem, the retailer must choose an assortment of products to offer to customers as well as the prices for these offered products with the goal of maximizing the expected revenue from each arriving customer. In order to capture each consumer's sensitivity to price, we assume that each customer class is distinguished by a budget in addition to an arrival probability and preference list. Arriving customers will purchase the highest ranking product in their respective preference list that is priced below their budget. It is not difficult to see that this problem generalizes the pure assortment problem and hence it is no surprise that the additional pricing element renders this problem more difficult. As a result, we place additional restrictions on the set of potential customer classes. We first assume that all preference lists are derived from a line graph; that is a tree consisting of a single path from the root to a leaf node. We call this model the interval model since all preference lists will be of the form $[i, i+1, \ldots, j]$. When prices are exogenous, this model reduces to the one-way substitution model of Honhon et al. (2012). It is important to note that endogenizing prices in this manner renders the techniques presented for the one-way substitution model in Honhon et al. (2012) irrelevant to our setting. As such, we provide the first polynomial-time algorithm for this problem when there are no restrictions on the set of prices that the retailer can charge. In Section 2 of the Online Appendix, we consider the joint assortment and pricing problem on general trees when prices are restricted to be quality consistent. We defer this analysis to the appendix since the techniques we use mirror those used for the pure assortment problem.

In addition to considering the above assortment and pricing problems, we also provide evidence for the practical importance of the general tree model. First, we show how to generate the tree structure from historical sales data rather than having the tree specified in advance, as in Honhon et al. (2012). This is an important differentiation since the structure may not always be clear or well-defined. We run two sets of experiments; the first uses synthetic sales data generated from a known ground choice model and the second uses the real hotel booking data provided in Bodea et al. (2009). We show that the fitted general tree models derived from the estimated tree structures capture customer behavior better than the well-known multinomial logit (MNL) choice model for both sets of experiments. For the experiments based on synthetic sales data, we find the optimal assortments under the fitted general tree model and the fitted MNL model, respectively. Then, we check the performance of these recommended assortments under the ground truth choice model, which we used to generate the data. We find that the assortments recommended by the general tree model often outperform those recommended by the MNL model by over 10%. These results give evidence against the use of revenue ordered assortments, which are well known to be optimal under the MNL choice model and often employed because of their intuitive nature. For the hotel data set, we do not know the true ground choice model so we test the fitted models based on the metric of likelihood. We find that the general tree model outperforms the MNL in the majority of the test cases.

Related Literature. There are a several papers that have considered the assortment optimization problem under the nonparametric choice model. The work that is most closely related to ours is Honhon et al. (2012), which considers the assortment problem restricted to intrees and outtrees. Both of these models have restrictions on which preference lists can be associated with customer classes. We extend the results of this paper by lifting many of these restrictions and working in a more general setting. Two other papers that are closely related to our work are Aouad et al. (2015b) and Aouad et al. (2015a). The former proves various hardness results related to the assortment problem under the nonparametric choice model. The latter considers the assortment optimization problem under the nonparametric choice model when customer preference lists are associated with structured set systems defined over a single overarching ordering of the products (e.g. a laminar family). The general algorithm provided in this paper can be used to solve the outtree case described in Honhon et al. (2012), but it does not generalize to the more complex intree case.

Our work on the joint assortment and pricing problem under the interval model most closely resembles the work of Rusmevichietong and Jagabathula (2015), who consider the same joint assortment and pricing problem under the most general form of the nonparametric choice model. Under this more general form, the joint assortment and pricing problem is NP-Hard. Motivated by this result, the authors present a polynomial-time approximation scheme (PTAS) whose runtime scales exponentially in a parameter they call d, which essentially represents how much any feasible pricing scheme is allowed to break a quality consistent structure. Their approach relies on a fairly intricate dynamic program which places a carefully chosen grid on the set of prices that the retailer can charge. To contrast, while we consider a less general choice model, the algorithms that we provide are optimal and their runtime is polynomial in all input parameters. There are a few earlier works that consider the joint assortment and pricing problem under the nonparametric choice model. Aggarwal et al. (2004) are the first to develop algorithms with provable performance guarantees for variations of the joint assortment and pricing problem. Most notably, when the prices are constrained by a price ladder, the authors are able to develop a PTAS. Rusmevichientong et al. (2006) also restrict the set of feasible prices to a price ladder. With this simplification of the pricing structure, the authors develop various heuristics, which they show work well in practice.

There is vast literature on assortment optimization problems under various other choice models. Talluri and van Ryzin (2004) solve the assortment optimization problem under the multinomial logit model. Extending this result, Rusmevichientong et al. (2010), Wang (2012), Davis et al. (2013), and Wang (2013) study various versions of the constrained assortment problems when customers choose according to the MNL model. Mendez-Diaz et al. (2010), Desir and Goyal (2013), and Rusmevichientong et al. (2014) focus on assortment problems when customer choices are governed by a mixture of multinomial logit models. Li et al. (2015), Davis et al. (2014), and Li and Rusmevichientong (2014) develop efficient methods for the unconstrained assortment problem when customers choose under the nested logit model. Gallego and Topaloglu (2014) and Feldman and Topaloglu (2015) consider the space and cardinality constrained versions of the assortment problem when customers choose according to the nested logit model.

More recently, Blanchet et al. (2016) introduce the Markov chain (MC) choice model and show that it subsumes the MNL model in addition to approximating other well-known

choice models quite accurately. Further, Hosseinalifam et al. (2015) show that the MC choice model subsumes any nonparametric choice model in which the preference lists are nested, meaning they take the form $[1, 2, \dots j]$. Since this nested nonparametric choice model is a special case of the outtree model, it is natural to wonder whether the general tree model can be captured with a MC choice model. It turns out that any deviation from this nested structure on the preference lists breaks the validity of the reduction presented in Hosseinalifam et al. (2015). In Section 4 of the Online Appendix, we show that we can construct a tree model on just three products that the MC choice model cannot capture. In this same appendix, we also comment on the difficulties in estimating the parameters of the MC choice model in relation to the general tree model.

The remainder of this paper is organized as follows. In Section 2, we describe the general tree model and introduce the assortment optimization problems that we study. In Section 2.1, we give our dynamic programming approach to solve the unconstrained assortment problem and show how this approach can be extended to consider cost or capacity considerations (Section 2.2). Next, we introduce and solve the joint assortment and pricing problem under the interval model in Section 3. We present computational experiments in Section 4 to validate the efficiency of our dynamic programming approach. In Section 5, we show that the general tree model can be effectively estimated. In Section 6, we conclude and provide directions for future work.

2. Assortment Optimization

First, we describe the general nonparametric choice model and the difficulty of its corresponding assortment problems in an effort to motivate our research for the general tree choice model. A retailer has access to a collection of n substitutable products indexed by $N = \{1, \dots, n\}$. There is a collection of customer classes $\mathcal{G}$, where each customer class $g \in \mathcal{G}$ is defined by an arrival probability λ_g and a product preference list σ_g defined over a subset of N. The list σ_g represents a customer's product preferences. We let $\sigma_g(i)$ be the rank of product i in customer class g's preference list and let $\sigma_g^{-1}(k)$ be customer class g's k^{th} most preferred product. Note that σ_g may not include all products in N. If the retailer offers assortment $S \subseteq N$ and the list σ_g contains an element of S, then a customer of type g will purchase product $\pi_g(S) := \arg\min_{i \in S \cap \sigma_g} \sigma_g(i)$. If σ_g does not contain an element of S, this customer does not make a purchase. In this case, we abuse notation and let $\pi_g(S) = 0$ and

say that the customer has "purchased" the so-called no-purchase option, which we give index 0.

Given the collection of customer classes $\mathcal{G}$ and offer set S, the probability that item i is purchased is

$$\Pr_i(S) = \sum_{g \in \mathcal{G}: \pi_g(S) = i} \lambda_g.$$

For every $j \in N$, we let $r_j > 0$ denote the revenue of product j. When assortment S is offered, the expected revenue is then

$$R(S) = \sum_{i \in S} r_i \cdot \Pr_i(S).$$

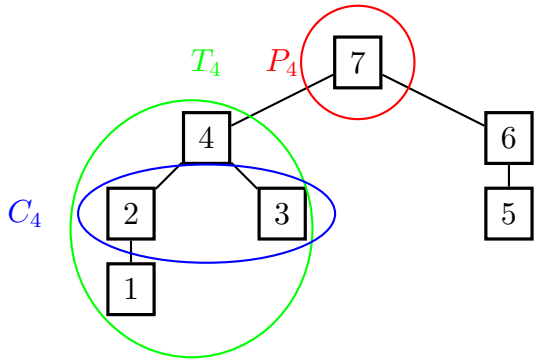
Our objective is to find a set $S^* \subseteq N$ that maximizes expected revenue. The assortment optimization problem is expressed as follows:

$$R^* = \max_S R(S). \tag{1}$$

Aouad et al. (2015b) show that problem (1) is NP-Hard to approximate within a factor of $O(n^{1-\epsilon})$ for any $\epsilon > 0$. Further, the hardness result of Aouad et al. (2015b) holds even when the preference lists for each customer type are constructed from a single overarching ordering, i.e. there exists an ordering $\prec$ on the products where $\sigma_g(i) < \sigma_g(j)$ implies $i \prec j$ for all $g \in \mathcal{G}$. As a result, a natural next step is to simplify the space of potential customer types in order to render the assortment problem tractable while not making too large a sacrifice in terms of modeling flexibility.

We will be interested in customer classes based on a rooted undirected tree structure $T = (N, E)$. The nodes in the tree represent all products that the retailer can potentially offer. Any customer class $g \in \mathcal{G}$ has a preference list σ_g associated with a path in T; the ordering of the products in σ_g corresponds to the order products are visited in a path through T. We restrict our attention to *linear* paths, which we define as paths that visit at most one child of every node. See Figure 1. In an effort to solve the assortment problems for the most general form of the tree model, we allow the paths that we associate with preference lists to move towards or away from the root node. However, we emphasize that the main benefit of the general tree model is that paths associated with customer classes can start anywhere in the tree. When no confusion arises, we identify σ_g with a path in the tree and refer to the preference list as moving towards or away from the root. In what

Figure 1 **Example Tree.**



Note. An example of a set of customer classes represented as a rooted binary tree. Possible customer preference lists are all linear paths including $(7,4,2,1)$, $(3,4)$, and (5). The path $(1,2,4,3)$ is not linear and would not correspond to a possible customer class.

follows, we will assume the tree T is a binary tree. This assumption is without loss of generality; we can meet this requirement by adding at most n nodes that represent null products that provide no cost or benefit to the retailer.

2.1. Unconstrained Problem

In this section, we provide a dynamic program for the assortment problem given in (1). The structure of the underlying tree T will guide the steps of computation in our dynamic program. Before stating the dynamic program, we first introduce additional notation and develop specific insights into solving (1) in a tree T. Table 1 summarizes the various pieces of notation that we use.

Table 1 **Tree Notation**

T	$\triangleq$	Rooted tree structure (N,E)
T_i	$\triangleq$	Subtree rooted at node $i \in N$ containing all successors of i
C_i	$\triangleq$	Children of node $i \in T$
P_i	$\triangleq$	Parent of node $i \in T$
$\phi_i(S)$	$\triangleq$	For $S \subseteq N$ and $i \in S$, i's closest predecessor in S
$\delta_i(S)$	$\triangleq$	For $S \subseteq N$ and $i \in S$, the set of closest successors to i in S.
$\Phi(i)$	$\triangleq$	All of i's predecessors in T in addition to the no-purchase option

Given a vertex i, we let C_i be the children of i in T. Further, we say that i is the parent of all $j \in C_i$ and define $P_j = i$. Note that for leaves of T, $C_i = \emptyset$. We will also be interested in complete subtrees of T. We let T_i be the subtree rooted at i containing all successors of i. When there is no confusion we will also use T_i to refer to the products represented by the nodes of the complete subtree. Without loss of generality, we can index the nodes such that the root node has index n and if $T_i \subset T_j$ then $j > i$. See Figure 1.

The tree T will be used to define *blocking* relationships among products. For a customer class g, we say i blocks product j when S is offered if $\pi_g(S) = i$ and $\pi_g(S \setminus \{i\}) = j$. More generally, we say i blocks j when S is offered whenever there exists at least one class g for which this blocking relationship holds. Intuitively, i blocks j when the removal of i from the offer set induces a customer class to purchase product j. Since T defines the ordered lists for customer classes, these blocking relationships are tied to T. We define, for any pair of nodes, the degree to which these nodes block each other. Specifically, we let

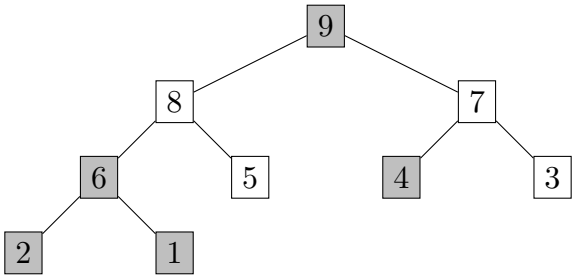
$$B_{i,j} = \sum_{g \in G : \pi_g(\{i,j\}) = i, \pi_g(\{j\}) = j} \lambda_g.$$

Note that $B_{i,j}$ is not identical to $B_{j,i}$ since these two terms involve customer classes moving in opposing directions, which may have different associated probabilities. In addition to describing blocking in terms of probability, we will also describe blocking in terms of revenue. We let $r_j B_{i,j}$ be the revenue i blocks from j when $\{i, j\}$ is offered.

Given a subset S and $i \in S$, we define $\phi_i(S)$ to be i's closest predecessor in S and $\delta_i(S) = \{j \in S | \phi_j(S) = i\}$ to be the set of closest successors to i in S. If no predecessor of i is offered in S, we let $\phi_i(S) = 0$. If no successors are offered, we let $\delta_i(S) = \emptyset$. Further, we use $\Phi(i)$ to represent all of i's predecessors in T in addition to product 0, the no-purchase option. See Figure 2.

If we offer subset S and $i \in S$, any customer class g traveling away from the root that has i in its preference list does not end up purchasing i if and only if she purchases a predecessor j of i. Since all customer classes are linear, this customer class must also contain $\phi_i(S)$ in σ_g before i. Therefore, we know σ_g contains both $\phi_i(S)$ and i but ranks $\phi_i(S)$ above i. Similarly, a customer class traveling up the tree towards the root that has i in its preference list does not purchase i if and only if it purchases a successor of i. Since all customer classes are linear, σ_g must contain both a node $j \in \delta_i(S)$ and i but ranks j

Figure 2 Understanding Predecessors and Successors.



Note. If we offer products $S = \{9, 4, 6, 2, 1\}$, then the closest offered predecessor of product 2 is $\phi_2(S) = 6$ and the closest offered successors of product 9 are $\delta_9(S) = \{6, 4\}$.

above i. These considerations allow us to rewrite the probability $i \in S$ is purchased using our blocking notation:

$$\Pr_i(S) = \Pr_i(\{i\}) - B_{\phi_i(S),i} - \sum_{j \in \delta_i(S)} B_{j,i}. \tag{2}$$

This alternative expression is critical for the development of our dynamic program.

Our dynamic program is based on maximizing *adjusted revenues* in complete subtrees of T. Intuitively, given T_i and a node $p \in \Phi(i)$, the adjusted revenue of an offer set $S_i \subseteq T_i$ is the revenue received from products in S_i when we offer $S_i \cup \{p\}$ minus the revenue S_i blocks from the product p. More precisely, given a subset $S_i \subseteq T_i$ and a closest offered predecessor p of i, we define the adjusted revenue of S_i to be

$$A(S_i, p) = \sum_{j \in S_i} r_j \Pr_j(S_i \cup \{p\}) - r_p \sum_{k \in \delta_p(S_i \cup \{p\})} B_{k,p}.$$

Note that this expression also holds when $p = 0$. The first term is the revenue received from products in S_i when the offer set is $S_i \cup p$. The second term accounts for the revenue S_i blocks from p. The proposition below shows that the revenue of any assortment can be computed by summing adjusted revenues.

PROPOSITION 1. *Consider any subset $S \subseteq N$ and node i with children l and r. Let $S_i = T_i \cap S$, $S_l = T_l \cap S$, and $S_r = T_r \cap S$. Lastly, let $p = \phi_i(S)$. Then,*

$$A(S_i, p) = \begin{cases} A(\{i\}, p) + A(S_l, i) + A(S_r, i) & i \in S, \\ A(S_l, p) + A(S_r, p) & \textit{otherwise.} \end{cases}$$

In particular, this shows that $A(S, 0) = \sum_{i \in S} A(\{i\}, \phi_i(S)) = R(S)$.

Proof. We delay the proof to Appendix A. □

By Proposition 1, we can rewrite the unconstrained assortment problem given in (1) as $R^* = \max_{S \subseteq N} A(S, 0)$. We now present our dynamic programming formulation. Each stage is a product i under consideration for inclusion in S and the one dimensional state space is a product p, possibly equal to 0, that is the closest offered predecessor of i in T. Our value function $V_i(p)$ is the maximum adjusted revenue that can be achieved from subsets of T_i when p is the closest offered predecessor of i.

$$V_i(p) = \max\{r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \sum_{k \in C_i} V_k(i), \sum_{k \in C_i} V_k(p)\}. \tag{3}$$

For leaves of T, our base case, this simplifies to $V_i(p) = \max\{r_i \Pr_i(\{i\}) - r_p B_{i,p} - r_i B_{p,i}, 0\}$.

THEOREM 1. $V_i(p) = \max_{S_i \subseteq T_i}\{A(S_i, p)\}$.

Proof. We delay the proof to Appendix A. □

The special case of Theorem 1 when $i = n$ and $p = 0$ shows that our dynamic program computes the optimal solution to (1). We can now analyze the computational complexity of computing the necessary $V_i(j)$. Let $\mathcal{D}$ be the depth of T. The depth of a tree T is the length of the longest path from the root to one of the leaves of the tree. We pre-compute each $\Pr_i(\{i\})$ and $B_{i,j}$. The number of customer classes is $|\mathcal{G}| = O(n\mathcal{D})$ since linear paths in the tree are uniquely determined by a starting and ending point in the tree. Each $g \in \mathcal{G}$ contributes to at most $\mathcal{D}^2$ blocking values $B_{i,j}$, since $|\sigma_g| \leq \mathcal{D}$ and we are interested in pairs of nodes in σ_g. Each customer class also contributes to at most $\mathcal{D}$ singleton purchase probabilities $\Pr_i(\{i\})$. Therefore, calculating the $\Pr_i(\{i\})$ and $B_{i,j}$ values has running time $O(n\mathcal{D}^3)$. After this pre-computation, each of the $O(n\mathcal{D})$ values $V_i(j)$ can be computed in constant time. This leads to an overall running time of $O(n\mathcal{D}^3)$. For a full binary tree, $\mathcal{D} = \log n$ leading to a running time of $O(n \log^3 n)$.

2.2. Extensions of the Dynamic Program

This dynamic program can be easily extended to incorporate additional considerations. First, we focus on two cost considerations proposed in Honhon et al. (2012): a setup or stocking cost incurred when offering a product and a substitution penalty incurred when a customer is forced to substitute to less desirable products. To model setup costs we introduce a constant fixed cost of k_i for offering product i. To model the substitution penalty we introduce a function $f(l)$ that represents the penalty incurred when a customer

purchases their l^{th} most preferred product. Specifically, if a customer of type g purchases product i and $l = \sigma_g(i)$ then the retailer incurs a penalty of $f(l)$. In Honhon et al. (2012) they assume that $f(\cdot)$ is linear and increasing; we consider arbitrary functions. We modify our dynamic program to include these costs.

Similar to our previous blocking terms, we let

$$\text{Pen}_i = \sum_{g \in G: i \in \sigma_g} \lambda_g f(\sigma_g(i))$$

be the sum of penalties the retailer incurs if assortment $S = \{i\}$ is offered. When offering i prevents a customer from substituting further down in their list, it can potentially lower the total penalty. This inspires a notion of "blocking" similar to that which we introduced in the previous section. Given any pair of nodes, we let

$$Q_{i,j} = \sum_{g \in G: \pi_g(\{i,j\}) = i, \pi_g(\{j\}) = j} \lambda_g f(\sigma_g(j))$$

be the penalty i blocks from j. We can now write the modified dynamic program:

$$V_i(p) = \max\{r_i \text{Pr}_i(i) - r_i B_{p,i} - r_p B_{i,p} - k_i - \text{Pen}_i + Q_{i,p} + Q_{p,i} + \sum_{k \in C_i} V_k(i), \sum_{k \in C_i} V_k(p)\}. \quad (4)$$

For leaves of T, our base case, this simplifies to $V_i(p) = \max\{r_i \text{Pr}_i(i) - r_i B_{p,i} - r_p B_{i,p} - k_i - \text{Pen}_i + Q_{i,p} + Q_{p,i}, 0\}$. The value functions in (4) capture the adjusted revenue for product i and all of its successors given that product p is the closest offered successor of i. Here, $r_i \text{Pr}_i(i) - r_i B_{p,i} - k_i - \text{Pen}_i + Q_{p,i}$ is the revenue received from i, modified to include costs and penalties, when p is the closest offered predecessor and other products in T_i are not offered.

Second, we consider the cardinality constrained assortment problem, in which the retailer is restricted by a cardinality constraint and can offer at most $C \leq n$ products. For example, C might be how many products are displayed on a website or the physical space allotted to these products. In this problem setting, we will need an additional state space that represents the remaining number of products in T_i that we have left to offer. Our value functions $V_i(p, c)$ will be the maximum adjusted revenue that can be achieved from subsets of T_i by offering at most c products when p is the closest offered predecessor of i. The modified dynamic programming recursion is as follows:

$$V_i(p, c) = \max\{r_i \text{Pr}_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \max_{c_l, c_r : c_l + c_r \leq c-1} V_l(i, c_l) + V_r(i, c_r), \quad (5)$$

$$\max_{c_l, c_r : c_l + c_r \leq c} V_l(p, c_l) + V_r(p, c_r)\}.$$

For leaves of T, our base case, this simplifies to

$$V_i(p,c) = \begin{cases} \max\{r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p}, 0\} & c > 0 \\ 0 & c = 0 \end{cases}.$$

This inner maximization represents an optimal allocation of the remaining products to i's left and right children, which we represent as nodes l and r respectively.

3. Joint Assortment and Pricing

In the joint assortment and pricing problem, the retailer must simultaneously decide which products to offer and the prices to charge for each of these offered products with the goal of maximizing the expected revenue from each arriving customer. As defined in Section 2, the general tree choice model does not explicitly capture consumer sensitivities to prices. As a result, we add a budget, or willingness to pay, to the identifying characteristics of each customer class. In this setting, each customer class $g \in \mathcal{G}$ is distinguished by an arrival probability λ_g, a preference list σ_g, and a budget b_g. We let $m = |\mathcal{G}|$, which will be used in analyzing our algorithm's runtime. Let $\{b_1, b_2, \ldots, b_d\}$ be the set of budgets for all customers in $\mathcal{G}$. We assume the budgets are indexed such that $b_i \le b_{i+1}$ for all $i = 1, \ldots, d-1$. An arriving customer will purchase the highest ranking product that is priced within her respective budget, if any.

To formally define our joint assortment and pricing problem, suppose again we have products $N = \{1, 2, \ldots, n\}$. The retailer must choose a price for each product with the goal of maximizing the expected revenue from each arriving customer. If a customer purchases product i priced at $p \in \mathcal{R}_+$, then the retailer makes a revenue of $p - c_i$, where c_i is the fixed unit cost of acquiring one unit of product i. We represent our pricing decisions as the vector $\mathcal{P} = (p_1, p_2, \ldots, p_n) \in \mathcal{R}_+^n$, where p_i is the price that we charge for product i. We use the convention that setting $p_i = \infty$ is equivalent to not offering the product. If the retailer sets prices $\mathcal{P}$, then a customer of type g will purchase product $\pi_g(\mathcal{P}) := \arg\min_{l \in \sigma_g : p_l \le b_g} l$. Therefore, the probability product i is purchased under prices $\mathcal{P}$ is

$$\Pr_i(\mathcal{P}) = \sum_{g \in \mathcal{G} : \pi_g(\mathcal{P}) = i} \lambda_g.$$

We can find the optimal assortment and prices to offer by solving the problem

$$\mathrm{OPT}_p = \max_{\mathcal{P} \in \mathcal{R}_+^n} \sum_{i \in N} (p_i - c_i) \Pr_i(\mathcal{P}). \tag{6}$$

In Section 1 of the Online Appendix, we show that we can simplify the problem by noting that there exists an optimal pricing scheme $\mathcal{P}^*$ to problem (6) where $\mathcal{P} \in \{b_1, \ldots, b_d, b_{d+1}\}^n$, where $b_{d+1} = \infty$. This shows that we can optimally solve problem (6) by only considering a finite menu of prices. Note that the number of budgets cannot exceed the number of customer classes so $d = O(m)$. For the remainder of this paper, we say that product i is priced at level j to mean that $p_i = b_j$. Further, we use $r_{i,j} = b_j - c_i$ to represent the revenue when product i is priced at price level j.

It is easy to see that the joint assortment and pricing problem generalizes the standard assortment optimization problem from Section 2.1, and thus problem (6) presents new difficulties. As a result, we consider the problem for two special cases. We first consider the case when all preference lists are derived from a tree that is a single path on the products. We rename this model the interval model, since the preference lists derived from the aforementioned tree structure can be viewed as intervals of consecutive integers when the products are indexed appropriately. For the interval model, we develop a novel dynamic programming approach that is completely distinct from the approach given in Section 2.1. In Section 2 of the Online Appendix, we consider the case when we have a general tree model, but are restricted in that the set of feasible pricing policies must be quality consistent within the tree structure, i.e., the price of any product offered is at most the price of any of its predecessors. In this case, we show that the optimal prices can be derived from the dynamic programming idea that we developed for the pure assortment problem.

3.1. Pricing with the Interval Model

In the interval model, we assume that products are indexed by decreasing quality, so that product 1 has the highest quality and product n the lowest, and that customers come in considering a *quality interval*. This quality interval represents the range of qualities of a particular product that a customer is potentially willing to purchase. For example, we might have a customer that just considers the top quality product. In contrast, we might have another customer that will consider only mid-quality to low-quality products. This may be because the products of higher quality can sometimes come with a trade-off such as ease of use or weight. For example, an average user may not be able to easily use a high-end camera and a backpacker may not want the added bulk of high-end luggage or the possibility of it being stolen. Such a scenario could be derived from a conjunctive screening rule with lower thresholds for quality or ease. To model such a scenario in our setting,

each customer class g is characterized by a budget b_g and preference list σ_g of the form $[i, i+1, \ldots, j]$ with $1 \leq i \leq j \leq n$. It is through these preference lists that we capture the quality interval of each customer class. Our algorithm extends if we also allow preference lists in opposite order, that is of the form $[i, i-1, i-2, \ldots]$. However, for ease of exposition, we ignore these potential customer classes.

The dynamic program that we develop focuses on the optimal way to price subintervals of products. As we price products, we continuously partition our intervals into smaller and smaller non-intersecting subintervals, which admit independent pricing problems. At the heart of our dynamic program formulation is the recursive manner with which we are able to stitch together the optimal pricing schemes for each interval while correctly accounting for the accrued revenue as we do so. To further build intuition, suppose that we start by pricing product l at price level 1, the lowest price level. If we decide to price every product $k < l$ at a price level above 1, then we know that all customers $g \in \mathcal{G}$ with $l \in \sigma_g$ and $b_g = b_1$ will purchase product l irrespective of the other pricing decisions. This fully accounts for all customers purchasing product l with budget b_1. We can now consider pricing products with index smaller than l at price levels b_2 and higher. On the other hand, products indexed higher than l can still potentially be purchased at b_1, but these purchases will be made by customers whose interval starts after product l. Therefore, l acts as a breakpoint in the interval, and we can show that the problem decomposes by the products indexed higher and lower than this point.

In particular, let $V_{i,j}(k_1, k_2)$, for all $1 \leq i, j \leq n$ and $1 \leq k_1 \leq k_2 \leq d$, be the value functions of our dynamic program. The value function represents the maximum expected revenue that can be accrued from customers whose highest ranked product is in the interval $[i, i+1, \ldots, j]$, when:

- the price level charged for product $j+1$ is k_1. If $j = n$, we write $k_1 = \emptyset$,
- we only account for customers with a budget that is at least b_{k_2}, and
- we price each product in the interval $[i, i+1, \ldots, j]$ at a price level of k_2 or higher.

Note that $V_{1,n}(\emptyset, 1)$ gives the optimal expected revenue for problem (6). We can calculate $V_{i,j}(k_1, k_2)$ using the following dynamic program

$$V_{i,j}(k_1, k_2) = \max\{ \sum_{g \in \mathcal{G}: \sigma_g^{-1}(1) \geq i, j+1 \in \sigma_g, b_g = b_{k_2}} \lambda_g \cdot r_{j+1,k_1} + V_{i,j}(k_1, k_2+1), \tag{7}$$
$$\max_{l: i \leq l \leq j} [\sum_{g \in \mathcal{G}: \sigma_g(l) = 1, b_g = b_{k_2}} \lambda_g \cdot r_{l,k_2} + V_{i,l-1}(k_2, k_2) + V_{l+1,j}(k_1, k_2)] \}$$

with the base cases

$$V_{i,i-1}(\cdot,\cdot)=0 \quad \text{and} \quad V_{i,j}(\cdot,d+1)=0.$$

The two cases in the maximum of the dynamic program given in (7) correspond to our decision of whether or not to price a product in the interval $[i,i+1,\ldots,j]$ at level k_2. The first case corresponds to not pricing any of these products at level k_2. In this case, each customer class $g\in\mathcal{G}$ such that $\sigma_g^{-1}(1)\geq i$ and $j+1\in\sigma_g$ whose budget is b_{k_2} will purchase product $j+1$ at price b_{k_1}. For the interval $[i,i+1,\ldots,j]$, we move on to considering prices and budgets at the k_2+1 price level. In the second term, we make the decision to price one of the products in the interval $[i,i+1,\ldots,j]$ at price level k_2. The inner maximization finds the best product l to price at such a level. When we make this decision, we know that all customers $g\in\mathcal{G}$ such that $\sigma_g(l)=1$ with budget $b_g=b_{k_2}$ will purchase product l at b_{k_2}. We then decompose the problem into disjoint intervals $[i,\ldots,l-1]$ and $[l+1,\ldots,j]$ whose optimal expected revenues can be computed separately. Since product l is priced at level k_2, we update the first entry of the state space to k_2 for the left interval.

THEOREM 2. $V_{1,n}(\emptyset,1)=\mathrm{OPT}_p$.

Proof. We delay the proof to Appendix A. □

4. Computational Experiments

In this section, we provide computational experiments which demonstrate the efficiency of the dynamic program presented in (4) to solve the costed assortment problem. We benchmark ourselves against the algorithm provided for intrees in Honhon et al. (2012). This algorithm has a theoretical runtime that is exponential is the number of products, but has been been shown to work far better in practice. Since this algorithm is only valid when applied to problems in which the least preferred product of all customer types is the root node, we restrict our computational experiments to cases of this nature.

4.1. Experimental Setup

In our computational experiments we generate a number of intree instances to test the efficacy of our dynamic program. For each instance, we solve the costed assortment problem using two different strategies. The first strategy utilizes the dynamic program given in (4), which we refer to as DP. The second approach uses the algorithm given in Honhon et al. (2012), which we refer to as ALG3 since it is labeled Algorithm 3 in this paper. Our goal is

to compare the performance of DP and ALG3 by measuring the respective CPU seconds required to solve each instance of the assortment problem.

We generate each of the intree instances in the following manner. Each of the instances that we consider consists of customer classes derived from a complete binary tree. In other words, the total number of nodes or products in each intree is $n = 2^{\mathcal{D}} - 1$, and we vary the number of levels in the tree $\mathcal{D} \in \{3, 4, \ldots, 9, 10\}$. Since ALG3 is only valid when the least preferred product of each customer is the root node, we restrict the set of customer classes derived from each intree to be of this variety. For each instance, we consider all n customer types and assume that each type arrives with equal probability. The revenues of each products are generated uniformly from the interval $[0, n]$. Once the revenues have been generated for a given problem instance, we then generate a fixed cost k_i for each product $i \in N$ uniformly over the interval $[0, r_{min}]$, where r_{min} is the smallest randomly generated revenue for the given instance. In this way, we ensure that the cost of offering a product never exceeds the revenue gained from a sale of the product. We leave out substitution costs since they do not necessarily complicate the assortment problems we study.

4.2. Computational Results

Table 2 summarizes our computational results. In all cases we used Python 2.7 on a Dell with an Intel Core i7-2600 Processor with 2.4 GHz and 8GB of RAM. The first column gives the number of levels in the intrees that we consider. For each value of $\mathcal{D}$, we generate 100 unique intrees using the method described in the previous section. The second column gives the average CPU seconds required for DP to solve the 100 instances, and the third column gives the maximum CPU seconds for DP over these 100 instances. Columns 4 and 5 give the same two statistics for ALG3.

The results in Table 2 indicate that DP significantly outperforms ALG3 in both average performance and worst case performance. Most notably, we confirm that DP does in fact scale polynomially with the number of nodes, while ALG3 appears to be on more of an exponential trajectory. Further, for DP, we observe that the maximum runtime is at most 25% larger than the average runtime over all values of $\mathcal{D}$. On the contrary, when $\mathcal{D} = 7$, the maximum runtime for ALG3 exceeded the average runtime by over 2500%. Since the maximum runtime appears to be growing exponentially with $\mathcal{D}$, it was not possible to get a sense of how ALG3 performs on the bigger instances with $\mathcal{D} > 7$. On the other hand, DP solves instances of the costed assortment problem with over 1000 products in fractions of a second.

Table 2 Comparing runtimes of DP and ALG3.

	DP		ALG3	
$\mathcal{D}$	Avg Secs.	Max Secs.	Avg Secs.	Max Secs.
3	2.6×10^{-4}	3.2×10^{-4}	4.4×10^{-4}	1.0×10^{-3}
4	7.7×10^{-4}	8.9×10^{-4}	1.9×10^{-3}	0.017
5	2.1×10^{-3}	2.3×10^{-3}	0.011	0.089
6	5.4×10^{-3}	5.7×10^{-3}	1.00	30.78
7	0.014	0.015	9.92	262.5
8	0.033	0.035	NA	NA
9	0.081	0.084	NA	NA
10	0.19	0.20	NA	NA

Comparing DP and ALG3 in terms of CPU seconds required to solve the costed assortment problem.

5. Estimating the General Tree Model

In this section, we provide computational experiments that demonstrate that the general tree model is more effective at capturing customer behavior than the well-known MNL model in two distinct settings. In the first setting, we compare the two models on synthetic data generated from general nonparametric choice models whose underlying preference lists start as a tree in our initial test cases but become progressively more noisy and random in later test cases. In this way, we are not only able to show that we are able to accurately recapture an underlying general tree model, but we also show that the general tree model performs quite well even when the retailer's understanding of the choice process is only vaguely accurate. Our main finding is that when assortment decisions are made based on fitted general tree models rather than fitted MNL models, the increase in profits can be as high as 20%. In the second setting, we compare the fitted general tree model versus the fitted MNL model on real sales data from two different hotels. We show that for this data set, the fitted general tree model has log-likelihoods that are on average slightly higher than that of the fitted MNL model.

Our estimation procedure for fitting the general tree model has two stages. Given a set of sales data, we first use a greedy heuristic described in the next section to construct the tree T that determines the set of feasible preference lists for our general tree model. Next, we derive the fitted general tree model and the fitted MNL model through maximum likelihood estimation (MLE). We use NP and ML to denote the fitted general tree model and MNL model respectively. In an effort to fit sparser choice models, we build the tree by only considering customer types associated with linear paths that move away from the root. In other words, we build generalized outtrees in which the most preferred product of any customer type can be any product in the tree.

5.1. Building and Fitting the General Tree Model

We now describe how we build the tree T of products that is used to construct the preference lists of the general tree model. We assume that we have access to the past purchase history of τ customers. We represent this purchase history as the set $\text{PH} = \{(S_t, z_t) : t = 1, \ldots, \tau\}$, where S_t is the assortment of products offered to customer t and z_t is the product purchased by this customer. We set $z_t = 0$ if customer t selects the no-purchase option. We use a greedy heuristic that incrementally adds nodes to the existing tree with the goal of maximizing the number of customer classes that could have arrived in each period. Specifically, let T^i be the tree in iteration i of the greedy procedure and let $\mathcal{G}(T^i)$ be the set of customer types that can be derived from the tree T^i. For a customer that was offered assortment S and purchased product z, we say that customer class g could have arrived if $\pi_g(S) = z$. For a given tree T^i, we let $I(T^i) = \sum_{t=1}^{\tau} \sum_{g \in \mathcal{G}(T^i)} 1_{\pi_g(S_t) = z_t}$ be the total number of customer classes that could have arrived over the τ customer arrivals when the set of preference lists is derived from the tree T_i. We use the function $I(T^i)$ as a proxy for how well the given tree explains the historical sales data.

Let the set $\hat{\cup}$ represent all insertions of product $j \notin T^i$ into the current tree T_i that maintain the desired tree structure. In iteration $i+1$ of the greedy heuristic, we find the best way to add the next product into the tree by setting $(j^*, \cup^*) = \arg\max_{j \in N \setminus T^i} \arg\max_{\cup \in \hat{\cup}} I(T^i \cup \{j\})/|\mathcal{G}(T^i \cup \{j\})|^p$ through complete enumeration. We then set $T^{i+1} = T^i \cup^* \{j^*\}$. We continue in this manner for $n-1$ iterations, at which point we will have placed all of the products in the tree. Notice that we normalize $I(\cdot)$ by the number of customer classes that can be derived from the tree raised to a power p. By varying p, we show that we can control the depth of the tree that we discover. Choosing to build the tree with a smaller value of p will lead to trees with larger depths. These trees have will have a larger number of customer types whose preferences closely follow some universal ranking of the products, since there will be little branching in these trees. On the other hand, trees built with a larger value of p will have smaller depths and hence quite a bit of branching. These trees will have fewer customer types, but a greater heterogeneity in the preference lists. In our computational experiments, we vary $p \in \{0, 0.5, 1\}$ and for each fitted model we report the average depth of the trees that our heuristic finds. In Section 3 of the Online Appendix, we provide an example of running this greedy heuristic and we also give one of the trees produced from our heuristic on the hotel data set.

Now that we have built the tree from the past purchase history, we need to estimate the arrival probabilities for all possible paths. For a given tree T, the log-likelihood can be expressed as a function of the arrival probabilities $(\lambda_1, \ldots, \lambda_m)$. We write the log-likelihood of the training set as $\mathcal{L}(\lambda) = \sum_{t=1}^{\tau} \log \sum_{g \in \mathcal{G}(T)} \lambda_g 1_{\pi_g(S_t)=z_t}$. Note that it is immediately obvious that the log-likelihood is concave and thus maximum likelihood estimation will be tractable. In our computational experiments, we use MATLAB's built in non-linear constrained solver $fmincon$ to get our maximum likelihood estimates. Since we estimate models with at most $O(n^2)$ customer classes this approach is highly efficient.

5.2. Known Ground Choice Model

In this set of computational experiments, we generate the historical sales data from a nonparametric choice model that differs significantly from the general tree model in most of the test cases. We refer to the model that generates the sales data as the ground truth choice model. In the ground truth choice model, the set of customer types is given by $\mathcal{G} = \{1, \ldots, m\}$. The preference list and arrival probability of each customer class $g \in \mathcal{G}$ are respectively given by σ_g and λ_g. To generate the arrival probabilities $(\lambda_1, \ldots, \lambda_m)$, we set $\lambda_g = 1/m$ so that each customer class arrives with equal probability. Our approach for generating the preference lists is motivated by a setting in which a retailer sells a set of vertically differentiated products, meaning there is an overarching ordering on the qualities of the products. To start, we associate a quality interval, as described in Section 3, with each customer class. In order to better capture a true heterogeneous customer population, we introduce noise into the ordering of the products. Specifically, we assume that some customers drop items from their respective quality interval. In addition, we also allow for customers to have slight deviations from the overarching quality rankings. In particular, we assume that the ordering of products in a quality interval can be flipped. By including such idiosyncrasies, the preference lists of the underlying customer population differ significantly from any set that could be generated from a general tree model.

To be more specific, the following procedure was used to generate the preference list for each customer class. We assume that product 1 has the highest quality and product n has the lowest quality. For each customer class $g \in \mathcal{G}$, we first generated an initial quality interval $q_g = [i_g, \ldots, j_g]$. The most preferred product i_g is generated uniformly from the set $\{1, \ldots, n\}$, and the least preferred product j_g is then generated uniformly from the set $\{i_g, \ldots, n\}$. We then drop each product $k \in q_g$ with probability p_d. We update q_g to

be the resulting preference list. Finally, we consider $\mathcal{F}$ flip events on q_g, which are each executed with probability 0.5. If a flip event is executed, we uniformly sample a product $k \in q_g$ and flip its ordering with the product ranked immediately ahead of it. We repeat the above procedure until we have generated m unique preference lists. To ensure that we consider a diverse array of underlying ground truth choice models, we vary $(p_d, \mathcal{F}) \in \{0, 0.25, 0.5\} \times \{0, 2, 4\}$. Note that when $p_d = 0$ and $\mathcal{F} = 0$, we recover the preference lists of the interval model described in Section 3. We include this combination of parameters in our computational experiments to show that when the ground truth choice model is a general tree model, then we significantly outperform the MNL model.

In all the computational experiments, we set the number of products to $n = 10$ and the number of customer classes to $m = 20$. Once the ground truth choice model has been generated, we then generate the historical sales data under the assumption that the purchasing behavior of each arriving customer is governed by the ground truth choice model. We assume that we have access to the past purchasing history of τ customers. Recall that we represent this purchasing history as the set $\text{PH} = \{(S_t, z_t) : t = 1, \ldots, \tau\}$, where S_t is the assortment of products offered to customer t and z_t is the product purchased by this customer. We sample the subsets S_t such that each product is included in the assortment with probability 0.75. The class g_t that customer t belongs to is sampled from the distribution $(\lambda_1, \ldots, \lambda_m)$. Given that the ground truth choice model is a nonparametric choice model, we set $z_t = \arg\min_{i \in S_t} \sigma_{g_t}(i)$.

For each choice of p_d and $\mathcal{F}$, we generate 10 ground truth choice models. Then, for each of these choice models we generate 10 past purchase histories with $\tau = 2500$. Lastly, for each of these past purchase histories we generate 100 different possible revenues for the products where the revenue of each product is generated uniformly at random from the interval $[0, 100]$. This gives us $9 \times 10 \times 10 \times 100 =$ 90,000 data sets where data set D_k is associated with a purchase history PH_k and a set of revenues $(r_1^k, \ldots, r_n^k)$.

We test the efficacy of the fitted models by computing the optimal assortment recommended by each of the fitted models under the assumption that choice is governed by the fitted model. We then test the performance of these recommended assortments under the ground truth choice model. We also compare how well the two fitted models predict future buying behavior by computing the log-likelihoods of each fitted model on a testing set of sales data.

We first find the optimal assortments under the assumption that customer choice is governed by each of the fitted models. Suppose on a data set D_k generated from ground truth choice model GC, we have fitted models NP and ML. For CM $\in \{\text{NP}, \text{ML}, \text{GC}\}$, let $P_i^{\text{CM}}(S)$ be the probability that product i is purchased under choice model CM. We compute the optimal recommended assortment under the fitted general tree model as $S^k(\text{NP}) = \arg\max_{S\subseteq N} \sum_{i\in N} r_i^k P_i^{\text{NP}}(S)$ and the optimal assortment under the fitted MNL model as $S^k(\text{ML}) = \arg\max_{S\subseteq N} \sum_{i\in N} r_i^k P_i^{\text{ML}}(S)$. We then check the performance of these assortments by computing how well these assortments perform under the ground truth choice model, which is assumed to be reality. In particular, we compute expected revenues $R^k(\text{NP}) = \sum_{i\in N} r_i^k P_i^{\text{GC}}(S^k(\text{NP}))$ and $R^k(\text{ML}) = \sum_{i\in N} r_i^k P_i^{\text{GC}}(S^k(\text{ML}))$. For each ground truth choice model, we store the average expected revenue of the recommended assortments over the data sets for both of the fitted models.

	Param. Comb.								
p	(0, 0)	(0,0.25)	(0, 0.5)	(2, 0)	(2,0.25)	(2, 0.5)	(4, 0)	(4,0.25)	(4, 0.5)
0	17.00	11.50	12.73	16.17	12.11	11.75	13.42	10.79	12.07
0.5	16.50	10.73	12.38	15.09	10.76	10.99	12.07	9.62	11.31

Table 3 The average percentage improvement in expected revenue of the general tree choice model over the MNL model for $p = 0$ and $p = 0.5$.

Table 3 compares the predictive powers of the general tree model and the MNL model over the various parameter combinations given in Columns 2-10. The numbers reported are the percentage gains in the expected revenue of the recommended assortment of the fitted general tree model compared with the fitted MNL model averaged over the data sets. So for the ground truth choice models generated with $\mathcal{F} = p_d = 0$, the assortments recommended by the general tree model fit with $p = 0$ have expected revenues that are on average 17% higher than the expected revenues of the assortments recommended by the fitted MNL models. Overall, it is clear from Table 3 that the assortments recommended by the fitted general tree model are far more profitable than the assortments recommended by the fitted MNL models. For the trees fit with $p = 0$, the average percentage gain across all parameter combinations never drops below 10%. For the trees fit with $p = 0.5$, the smallest average is 9.62%. Further, there are instances for which the improvements of the general tree model exceed 20% and there is only a single instance for which the average percentage

improvement is below 5%. Last, we expect that as p increases, the depth of the fitted tree decreases. This turns out to be exactly what we observe: when $p = 0$ the average depth is 10 (always builds an interval model) and when $p = 0.5$ the average depth is 8.47 with a standard deviation of 1.21 and we get quite a diverse array of trees. Generally, the trees built with $p = 0$ only perform slightly better than the trees built with $p = 0.5$, which is somewhat surprising considering that the trees built with $p = 0$ have significantly more customer types. The ability to fit trees of varying depth could be especially useful when the number of products is too large to estimate the $O(n^2)$ parameters of the interval model. In Section 3 of the Online Appendix, we show that the predictive power of both fitted models are comparable as measured by an out-of-sample log-likelihood.

5.3. Hotel Data Set

In this section, we compare the general tree model and the MNL model on the hotel bookings data set provided in Bodea et al. (2009). This data set consists of bookings at 5 different hotels made from March 12, 2007 to April 15, 2007 made primarily by business customers through online channels or customer relationship employees. We decide to focus only on Hotels 1 and 3 since the number of purchases in the sales data at these two hotels exceeds the purchases at the other three hotels by a factor of five. Each hotel offers a variety of rooms (suite, king, queen, etc . . .) at differing rates based on additional accommodations that the room might come with. For example, Hotel 1 offers a King Room both at Rate 1 and at Rate 5. The former is a discounted advanced purchase rate and the latter is a rate that includes city activities such as dining and shopping. Since the price for a specific room varies by its accompanying rate, we treat each room type and rate tuple as a different product. We discard products that have fewer than 5 purchases throughout the selling horizon. This notion of a product differs from the works of Vulcano et al. (2016) and Vulcano and van Ryzin (2015)), who also use this data to fit nonparametric choice models. In these works, a product is simply a room type, and then, in order to capture the effect of the various rates, the authors assume that customers always buy up within the same room type. The upside of our approach is that we capture a more granular view of choice. However, in modeling a product as a room-rate tuple we have more products and hence more parameters to estimate. To control for overfitting, we perform a rigorous 10-fold cross validation procedure, which we describe later in this section.

The data set provides detailed information about the set of products that were offered and the product that was purchased at the time of each booking booking. Consistent with the other works that use this data set, we restrict our study to bookings for which there is at least one transaction per product and for which the observed purchase comes from the available options. Hotel 1 ends up with 33 products and 1,267 bookings and Hotel 3 ends up with 30 product and 1,109 bookings. Since the data set only gives booking information, there are no data points for which the no-purchase option is selected. To make up for this deficiency in the data set, for each booking record we generate $np \in \{0, 1, \ldots, 10\}$ no-purchase records. In this way, we can study the performance of the various choice models for varying levels of no-purchase tendencies.

We index each of the data sets we fit using the the tuple $(h, np) \in \{1, 3\} \times \{0, 1, \ldots, 10\}$, in which the first entry h corresponds to the hotel that we consider and the second entry np gives the number of additional no-purchase bookings we add for each booking record. For each data set, we perform 10-fold cross validation to compare the out-of-sample performance of the MNL model to the general tree model. To do so, we randomly partition each data set into ten equal segments. Nine of the ten segments make up the training data set, while the remaining segment is used for testing. For each test case (h, np), we build three trees by varying the normalizing constant $p \in \{0, 0.5, 1\}$. We fit the MNL model and the three general trees using MLE on the training data set and then measure the accuracy of these fitted models using the log-likelihood of the testing sets. Of the three general tree models that we fit, we only test the tree model that has the largest training log-likelihood. We repeat this procedure ten times so that each segment is the testing set at one point. Then, for each data set, we repeat this 10-fold cross validation ten times to ensure that our results are robust to the randomization that occurs within the cross validation. We average the test log-likelihoods over the ten trials and the ten folds.

The average percentage improvements in log-likelihood of the fitted general tree model over the MNL choice model are given in Table 4. For lower values of np, the MNL provides slight improvements over the general tree model. However, as np increases the general tree model appears to dominate. Even though we improve over the MNL model by only a few fractions of a percent, these results are nonetheless surprising in light of the performance of the most general nonparametric model on these same data sets presented in past studies. Specifically, the work of Vulcano and van Ryzin (2015) shows that the MNL model

	# of addition no-purchase events										
Hotel #	0	1	2	3	4	5	6	7	8	9	10
Hotel 1	-1.93	-1.08	-0.32	-0.03	-0.04	**0.27**	**0.36**	**0.33**	**0.41**	**0.49**	**0.49**
Hotel 3	**0.49**	-1.21	**0.22**	**0.24**	-0.56	-0.03	-0.02	**0.08**	**0.20**	**0.14**	**0.25**

Table 4 Percentage improvement in log-likelihood of the general tree choice model over the MNL model on the hotel datasets.

outperforms the general nonparametric model for all 5 hotels on a likelihood-based metric which is normalized by the number of parameters fitted. Our results show that there can be benefits to imposing specific sparse structures on the set of preference lists in settings in which there is clearly some notion of feature differentiation among the products. The tree model seems to be a valid candidate for imposing this structure.

Last we note that trees with $p = 1$ are the best fitting trees over 75% of the time, which is fairly surprising since the trees fit with $p = 0$ have significantly more customer types. These results are in stark contrast to our results for the synthetic data in which the trees built with $p = 0$ generally performed the best. We turn to the underlying structure of the ground choice model to explain this trend. In the first set of experiments, the ground choice is assumed to be some slight perturbation of the interval model, which has depth n. As a result, it is no surprise that setting $p = 0$ performs best. In this second set of experiments, the fact that trees with smaller depths perform better in this setting indicates that there are a diverse array of preferences for hotel rooms among travelers.

6. Conclusion

In this paper, we introduce the general tree customer choice model. We begin by studying the assortment problem. We give the first polynomial-time algorithm to solve the general tree case for linear customer classes. We formulate this problem as a dynamic program in which the offer decision for each product can be made by simply storing each node's closest offered predecessor. Further, our dynamic programming formulation extends naturally for the cardinality constrained assortment problem and costed assortment problem. We then study the joint assortment and pricing problem under special cases of the general tree model. We develop novel dynamic programming approaches that additionally reveal nice structural results of the optimal pricing scheme. We conclude our analysis by providing a series of computational experiments that validate the use of the general tree model in practice. As shown, the sparsity of this model admits tractable estimation and optimization problems while capturing more complex customer behavior than the MNL model.

References

Aggarwal G, Feder T, Motwani R, Zhu A (2004) Algorithms for multi-product pricing. *International Colloquium on Automata, Languages, and Programming*, 72–83.

Aouad A, Farias V, Levi R (2015a) Assortment optimization under consider-then-rank choice models, unpublished manuscript.

Aouad A, Farias V, Levi R, Segev D (2015b) The approximability of assortment optimization under ranking preferences, unpublished manuscript.

Blanchet J, Gallego G, Goyal V (2016) A markov chain approximation to choice modeling. *Operations Research* 64(4):886–905.

Bodea T, Ferguson M, Garrow L (2009) Data set: Choice-based revenue management: Data from a major hotel chain. *Manufacturing & Service Operations Management* 11(2):356–361.

Davis J, Gallego G, Topaloglu H (2013) Assortment planning under the multinomial logit model with totally unimodular constraint structures, unpublished manuscript: available at `http://legacy.orie.cornell.edu/~huseyin/publications/publications.html`.

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

Desir A, Goyal V (2013) An FPTAS for capacity constrained assortment optimization. Technical report, Columbia University, School of Industrial Engineering and Operations Research.

Feldman J, Topaloglu H (2015) Capacity constraints across nests in assortment optimization under the nested logit model. *Operations Research* 63(4):812–822.

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

Gilbride T, Allenby G (2004) A choice model with conjunctive, disjunctive, and compensatory screening rules. *Marketing Science* 23(3):391–406.

Honhon D, Pan X, Sreelata J (2012) Optimal algorithms for assortment selection under ranking-based consumer choice models. *Manufacturing and Service Operations Management* 14(2):279–289.

Hosseinalifam M, Marcotte P, Savard G (2015) Network capacity control under a nonparametric demand choice model. *Operarions Research Letters* 43(5):461–266.

Li G, Rusmevichientong P (2014) A greedy algorithm for the two-level nested logit model. *Operations Research Letters* 42(5):319–324.

Li G, Rusmevichientong P, Topaloglu H (2015) The d-level nested logit model: Assortment and price optimization problems. *Operations Research* 63(2):325–342.

Mahajan S, van Ryzin G (2001a) Inventory competition under dynamic consumer choice. *Operations Research* 49(5):646–657.

Mahajan S, van Ryzin G (2001b) Stocking retail assortment under dynamic consumer substitution. *Operations Research* 49(3):334–351.

Mendez-Diaz I, Bront J, Vulcano G, Zabala P (2010) A branch-and-cut algorithm for the latent-class logit assortment problem. *Electronic Notes in Discrete Mathematics* 36:383–390.

Rusmevichientong P, Roy BV, W P (2006) A nonparametric approach to multiproduct pricing. *Operations Research* 54(1):82–98.

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

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

Rusmevichietong P, Jagabathula S (2015) A nonparametric joint assortment and price choice model, unpublished manuscript.

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

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

Vulcano G, van Ryzin G, Ratliff R (2016) An expectation-maximization method to estimate a rank-based choice model of demand. *Operations Research* .

Wang R (2012) Capacitated assortment and price optimization under the multinomial logit model. *Operations Research Letters* 40:492–497.

Wang R (2013) Assortment management under the generalized attraction model with a capacity constraint. *Journal of Revenue and Pricing Management* 12(3):254–270.

Appendix A: Proofs

A.1. Proof of Proposition 1

Proof. First, suppose $i \in S$. Then, $\delta_p(S_i \cup \{p\}) = \{i\}$ since it is the only node in S_i for which p is the closest offered predecessor, i.e. $\phi_i(S_i \cup \{p\}) = p$. We have

$$\begin{aligned}
A(S_i,p) &= \sum_{j\in S_i} r_j \Pr_j(S_i\cup\{p\}) - r_p \sum_{k\in\delta_p(S_i\cup\{p\})} B_{k,p} \\
&= \sum_{j\in S_i} r_j \left(\Pr_j(\{j\}) - B_{\phi_j(S_i\cup\{p\}),j} - \sum_{k\in\delta_j(S_i\cup\{p\})} B_{k,j} \right) - r_p B_{i,p} \\
&= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \sum_{j\in S_i\setminus\{i\}} r_j \Pr_j(S_i\cup\{p\}) - r_i \sum_{k\in\delta_i(S_i\cup\{p\})} B_{k,i} \\
&= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \sum_{j\in S_i\setminus\{i\}} r_j \Pr_j(S_i) - r_i \sum_{k\in\delta_i(S_i)} B_{k,i}
\end{aligned}$$

where the second line follows from (2) and the last line comes from the fact that for each $j \in S_i \setminus \{i\}$, the closest predecessor of j is not p (since i is offered) and so $\Pr_j(S_i \cup \{p\}) = \Pr_j(S_i)$.

We can simplify this expression further. The set S_i can be decomposed into $\{i\}$ and two additional sets: $S_l = S_i \cap T_l$ and $S_r = S_i \cap T_r$. Let $j \in S_i \setminus \{i\}$. Without loss of generality, let $j \in T_l$. Then, $\phi_j(S_i) = \phi_j(S_l \cup \{i\})$ since its closest predecessor must be i or in the same subtree as j. This also shows $\delta_j(S_i) = \delta_j(S_l \cup \{i\})$ and $\Pr_j(S_i) = \Pr_j(S_l \cup \{i\})$. Lastly, we can easily see that $\delta_i(S_i) = \delta_i(S_l \cup \{i\}) \cup \delta_i(S_r \cup \{i\})$. Continuing from the above expression, these observations allow us to write

$$\begin{aligned}
A(S_i, p) &= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} \\
&\quad + \sum_{j \in S_l} r_j \Pr_j(S_l \cup \{i\}) - r_i \sum_{k \in \delta_i(S_l \cup \{i\})} B_{k,i} \\
&\quad + \sum_{j \in S_r} r_j \Pr_j(S_l \cup \{i\}) - r_i \sum_{k \in \delta_i(S_r \cup \{i\})} B_{k,i} \\
&= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + A(S_l, i) + A(S_r, i) \\
&= r_i \Pr_i(\{i, p\}) - r_p B_{i,p} + A(S_l, i) + A(S_r, i) \\
&= A(\{i\}, p) + A(S_l, i) + A(S_r, i).
\end{aligned}$$

The last inequality follows by the definition of the adjusted revenue and noting that $\delta_p(\{i, p\}) = p$. By very similar analysis, when $i \notin S$ we get $A(S_i, p) = A(S_l, p) + A(S_r, p)$.

Therefore, by unraveling the recursion given in the statement of the proposition we get that:

$$A(S, 0) = \sum_{i \in S} A(\{i\}, \phi_i(S)) = \sum_{i \in S} r_i \Pr_i(\{i\}) - r_i B_{\phi_i(S),i} - r_{\phi_i(S)} B_{i,\phi_i(S)} = \sum_{i \in S} r_i \Pr_i(S) = R(S),$$

where the second to last equality follows by equation (2). □

A.2. Proof of Theorem 1

Proof. First, consider the base case. For the leaves of T, $V_i(p) = \max\{r_i \Pr_i(\{i\}) - r_p B_{i,p} - r_i B_{p,i}, 0\}$. This first term is equivalent to $A(\{i\}, p)$ since $r_i \Pr_i(\{i, p\}) = r_i \Pr_i(\{i\}) - r_p B_{i,p}$ and the second term is $A(\emptyset, p)$ so the claim holds.

Now consider a node i that is not a leaf and suppose that the claim holds for all successors of i. Let l and r be the left and right children of i, respectively. Let $S_i^* \subseteq T_i$ be a subset that maximizes $A(S_i, p)$. In the first case, suppose $i \in S_i^*$. Then, $\delta_p(S_i^* \cup \{p\}) = \{i\}$ since it is the only node in S_i^* for which p is the closest offered predecessor, i.e. $\phi_i(S_i^* \cup \{p\}) = p$. From Proposition 1, we have that

$$A(S_i^*, p) = r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + A(S_l^*, i) + A(S_r^*, i).$$

Noting that S_i^* is the maximizer of $V_i(p)$ and that $A(S_l^*, i)$ and $A(S_l^*, i)$ are completely independent since they do not share any successors in the tree, we see that we can express our optimization problem recursively:

$$\begin{aligned}
\max_{S_i \subseteq T_i : i \in S_i} A(S_i, p) &= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \max_{S_l \subseteq T_l} A(S_l, i) + \max_{S_r \subseteq T_r} A(S_r, i) \\
&= r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + V_l(i) + V_r(i)
\end{aligned}$$

where we have used the inductive hypothesis.

By very similar analysis, when $i \notin S_i^*$ we get

$$\max_{S_i \subseteq T_i : i \notin S_i} A(S_i, p) = \max_{S_l \subseteq T_l} A(S_l, p) + \max_{S_r \subseteq T_r} A(S_r, p) = V_l(p) + V_r(p).$$

By combining these expressions we reach the desired claim

$$\begin{aligned}\max_{S_i \subseteq T_i} A(S_i, p) &= \max\{\max_{S_i \subseteq T_i : i \notin S_i} A(S_i, p), \max_{S_i \subseteq T_i : i \in S_i} A(S_i, p)\} \\ &= \max\{r_i \Pr_i(\{i\}) - r_i B_{p,i} - r_p B_{i,p} + \sum_{k \in C_i} V_k(i), \sum_{k \in C_i} V_k(p)\}.\end{aligned}$$

□

A.3. Proof of Theorem 2

Proof. We prove the result by proving the correctness of the dynamic program given in (7) through an inductive argument. The base cases hold trivially. We will now prove the correctness of the dynamic program for arbitrary value function $V_{i,j}(k_1, k_2)$. The base cases give rise to the following induction hypothesis. We can assume that the value function $V_{i',j'}(k_1, k_2')$ are correctly computed for the following combinations of i', j', and k_2': $i' = i, j' = j, k_2' = k_2 + 1$ and $i \le i', j' \le j, k_2' = k_2$.

Suppose that we decide not to price any product in the interval $[i, i+1, \ldots, j]$ at price level k_2. Then, customers $g \in \mathcal{G}$ such that $\sigma_g^{-1}(1) \ge i$ and $j+1 \in \sigma_g$ with budget $b_g = b_{k_2}$ will purchase product $j+1$ at price b_{k_1}. From these customers we gain an expected revenue of

$$\sum_{g \in \mathcal{G} : \sigma_g^{-1}(1) \ge i, j+1 \in \sigma_g, b_g = b_{k_2}} \lambda_g \cdot r_{j+1,k_1}.$$

It remains to maximize the expected revenue from customers $g \in \mathcal{G}$ such that $i \le \sigma_g^{-1}(1) \le j$ and whose budget is at least b_{k_2+1}. Since we have decided not to price any of these products at price level k_2, we can now consider price levels $k_2 + 1$ and higher for such customers. By our induction hypothesis, this expected revenue is given by $V_{i,j}(k_1, k_2 + 1)$ since product $j+1$ is still priced at level k_1. Combining both terms gives exactly the first term in the maximization of our dynamic program given in (7):

$$\sum_{g \in \mathcal{G} : \sigma_g^{-1}(1) \ge i, j+1 \in \sigma_g, b_g = b_{k_2}} \lambda_g \cdot r_{j+1,k_1} + V_{i,j}(k_1, k_2 + 1).$$

Otherwise, let product l be priced at level k_2. Note that the inner maximization over l ensures that we choose the optimal product to price at level k_2. Then, any customer class $g \in \mathcal{G}$ such that $\sigma_g(l) = 1$ and budget $b_g = b_{k_2}$ will purchase product l. This generates expected revenue

$$\sum_{g \in \mathcal{G} : \sigma_g(l) = 1, b_g = b_{k_2}} \lambda_g \cdot r_{l,k_2}.$$

We are still left to account for the revenue accrued from customers $g \in \mathcal{G}$ such that $i \le \sigma^{-1}(1) < l$ with budget $b_g \ge b_{k_2}$. Since product l is now priced at level k_2, we compute this revenue inductively from $V_{i,l-1}(k_2, k_2)$. On the other hand, the maximum expected revenue from customers with $l < \sigma_g^{-1}(1) \le j$ can be found inductively from $V_{l+1,j}(k_1, k_2)$. Therefore, the overall maximum expected revenue is

$$\sum_{g \in \mathcal{G} : \sigma_g(l) = 1, b_g = b_{k_2}} \lambda_g \cdot r_{l,k_2} + V_{i,l-1}(k_2, k_2) + V_{l+1,j}(k_1, k_2).$$

Taking the maximum over these two possibilities proves the claim. □