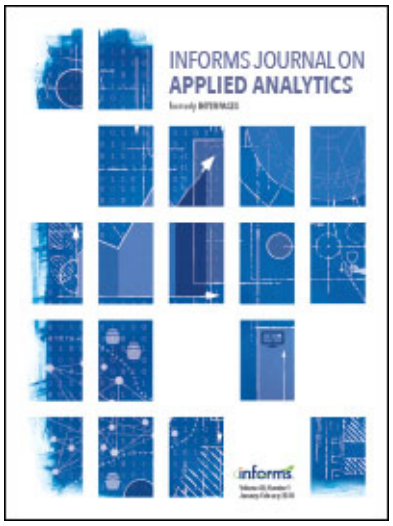

INFORMS Journal on Applied Analytics

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Formulation Matters: Reciprocating Integer Programming for Birchbox Product Assortment

Irvin Lustig, Patricia Randall, Robert Randall

To cite this article:
Irvin Lustig, Patricia Randall, Robert Randall (2021) Formulation Matters: Reciprocating Integer Programming for Birchbox Product Assortment. INFORMS Journal on Applied Analytics 51(5):347-360. https://doi.org/10.1287/inte.2021.1081

http://pubsonline.informs.org/journal/inte

INFORMS JOURNAL ON APPLIED ANALYTICS
Vol. 51, No. 5, September–October 2021, pp. 347–360
ISSN 2644-0865 (print), ISSN 2644-0873 (online)

Formulation Matters: Reciprocating Integer Programming for Birchbox Product Assortment

Irvin Lustig,[a] Patricia Randall,[a] Robert Randall[a]

[a] Princeton Consultants, Princeton, New Jersey 08540

Contact: irv@princeton.com, https://orcid.org/0000-0003-4571-1802 (IL); prandall@princeton.com (PR); rrandall@princeton.com (RR)

https://doi.org/10.1287/inte.2021.1081

Abstract. Birchbox created a mixed-integer programming formulation to determine the products that it will send to its subscribers in individual boxes on a monthly basis. The goal of this formulation is to produce a set of different box configurations that are then assigned to customers to meet the diverse needs of its varied customer base. As Birchbox's business grew, the mixed-integer program was taking days to solve, and experimenting with different business requirements to determine the best set of configurations became impossible. Therefore, Princeton Consultants created the Reciprocating Integer Programming technique to reduce these solution times, thus decreasing them to typically under 20 minutes. This has dramatically changed the way that Birchbox can run its subscription business.

History: This paper has been accepted for the *INFORMS Journal on Applied Analytics* Special Issue—2020 Daniel H. Wagner Prize for Excellence in the Practice of Advanced Analytics and Operations Research.

Keywords: mixed-integer programming • column generation • e-commerce

Overview

The online subscription-box service market was estimated to be more than $7.5 billion in 2019 (Fenyo and Mitchell 2019), with thousands of different services available (Panko 2019). Birchbox is the pioneer in these services, with millions of customers who receive a box of samples of beauty products each month. The customers expect that the products they receive each month will differ and will be tailored toward their beauty profiles and preferences based on their reviews of previous products they have received. In a typical month, Birchbox distributes 100 different samples in boxes, each of which contains five to seven unique samples. From a production point of view, uniquely tailoring each box to each customer is impossible because assembling individual boxes for each customer would be too expensive. As a result, Birchbox uses machine-learning algorithms to cluster its subscribers into groups. Following this clustering step, it then creates a set of box configurations to assign to each subscriber group to maximize the customer experience. To provide a better experience for its customers, increasing the number of clusters (currently around 4,000) would be desirable. However, more box configurations may be necessary to meet the product requirements of each individual cluster, because each group of customers has preferences that include both products that the customers would like to receive and products that they must not receive each month. In addition, Birchbox has rules that determine a valid box configuration. For example, one rule specifies that specific product pairs should not be in the same box; another limits the number of products from the same brand that are in a box.

In 2014, Birchbox created a mixed-integer programming (MIP) formulation to solve this assortment problem. It used the Gurobi optimizer as the integer programming solver; see Gurobi (2020) for information on the optimizer. In addition, the company acquired a powerful 32-core computer to solve this problem each month. However, when Birchbox decided to increase the number of samples in each box, the problem became nearly impossible to solve in a reasonable amount of time, and coming up with an acceptable solution often required multiple days. This limited Birchbox's flexibility to meet the needs of its ever-increasing and diverse customer base. Therefore, it retained Princeton Consultants to determine whether improving the solution times for this assortment problem was possible. The Princeton team reformulated the problem using concepts from column generation and then developed a new algorithm, the Reciprocating Integer Programming (RIP) technique, to solve this formulation. The RIP technique takes advantage of the power of the Gurobi MIP solver and leverages linear and integer programming concepts to reduce solution times from days to under 20 minutes for a typical data set.

In the Assortment Problem Description section, we provide more details of the requirements of the

assortment problem and the proposed solution approach. We show the notation used for the problem in the Problem Notation section, describe the original MIP formulation in the Original MIP Formulation section, and discuss a new formulation that sets the foundation for application of the RIP technique in the New MIP Formulation section. The Generating Box Configurations section formulates a pattern-generation problem that is solved to generate columns for the RIP technique, which we describe in the section The RIP Technique. The final section, Computational Results and Impact, provides a summary of the computational results using the RIP technique and summarizes the impact on the Birchbox business.

Assortment Problem Description

As we describe in the Overview section, Birchbox uses machine-learning algorithms to create clusters of its subscribers so that similar subscribers receive boxes with the same content each month. For ease of exposition, we will refer to the groups of subscribers as simply "subscribers." A typical data set has between 1,000 and 5,000 such subscribers and 20–100 products (the exact number depends on the product line), which can potentially be assigned to these subscribers. The goals of the assortment problem are to:

- Build a set of box configurations that contains subsets of the products (typically five or six products per box configuration) where the products in each box satisfy a set of rules that define a valid box.
- Assign each subscriber exactly one box configuration.
- Ensure that each configuration is sent to at least a specified minimum number of subscribers so that operations can efficiently build the planned boxes.
- Ensure that each configuration is sent to at most a specified maximum number of subscribers.
- Ensure that the number of planned box configurations is between a specified minimum and maximum.
- Ensure that if a product is used, it must be used for a desired minimum number of subscribers and for at most a desired maximum number of subscribers.

The rules for valid boxes are specified as follows:

- Box quantity: Each box must contain at least a specified minimum number of products and at most a specified maximum number of products. Each product may be used at most once within each box.
- Attributes: A set of attributes (e.g., value, weight, and volume) exist for each product. A valid box should have a total value between a specified minimum and maximum for each attribute.
- Pairing rules: Specific product pairs must be shipped together (i.e., if one of the pair is present in a box, the other product of the pair must also be present), and other product pairs must not be in the same box.

Finally, for each product, there is a (possibly empty) set of subscribers that must receive this product and another (possibly empty) set of subscribers that must not receive this product.

Birchbox's original MIP model to solve this product-to-subscriber assignment problem worked adequately for several years; however, as the customer base grew and offerings expanded, solutions to this model were difficult to obtain in a reasonable amount of time, limiting business flexibility. Operations executives were forced to painstakingly manage and monitor the model because it frequently could not find adequate solutions, and, when it could, the average run time of 30–50 hours jeopardized production deadlines.

When the company introduced improvements to its box experience in early 2019, its managers sought to increase the flexibility of assigning products to a box—for example, adding a sixth sample—but not compromising the integrity of the assortment. However, testing determined that the model could not adequately handle the additional mathematical complexity. Infeasibility was also an issue. Some sets of products, quantities, eligibility requirements, and "legal" box definitions (i.e., ones that meet the various rules) are not conducive to finding a valid assortment; however, the existing model did not allow operations to quickly ascertain if the inputs could produce a feasible solution or test a new configuration of rules.

Working with Birchbox personnel, we identified areas where the current formulation was leading to poor performance, such as the use of a blended objective function that attempted to measure feasibility (e.g., whether each subscriber receives a box), while minimizing the number of box configurations and meeting product minimums. The formulation also struggled with symmetry (Liberti 2012) because the same subset of products could be assigned to multiple box configurations, which made it difficult for the MIP algorithm to choose different subsets. After attempting several "quick fixes," such as randomly perturbing the objective function, implementing symmetry constraints, and adding various cuts, without significant performance improvement, we reformulated the problem using concepts based on column-generation techniques. This led to the creation of the RIP technique described below.

The first step in the new formulation was to generate a set of initial patterns (i.e., box configurations), assuring that each subscriber has feasible patterns and favoring products with a high volume of inventory. This set of patterns forms a restricted master problem. The restricted master problem is solved twice—first, as a linear program with multiple rounds of pattern generation to establish that every subscriber can receive a box (i.e., is the problem feasible?) and, second, as a MIP to do the actual box-to-subscriber assignments. Once all subscribers can be assigned a box, the total number of box configurations is minimized. The reformulation not only remediates the performance issues of the prior formulation, but also addresses the

need to quickly determine whether a specific set of inputs results in an infeasible problem.

Performance testing on a suite of test problems showed that the reformulation provided significant performance improvements, with problems that originally required hours to solve now being solved within minutes. The average run time is 10 minutes—an improvement of more than 99% over the previous model. The improved performance allows the operations team to tweak inputs and quickly rerun the model, thus enabling the team to evaluate different parameters, levels of subscriber aggregation, definitions of a "good" box, and even optimize on box value. Such experimentation was impossible previously because the team struggled to produce a single solution each cycle. David Bendes, Vice President of Global Business Technology at Birchbox, described this improvement as "life altering" (Lustig and Randall 2020).

The remainder of this paper describes notation for the data representing the problem, the original MIP formulation, and, finally, the details of the revised formulation, with a description of the RIP technique.

Problem Notation

To help the reader understand the new formulation and algorithm, we first describe some notation to represent the problem's data.

Sets

- $\mathcal{P}$ is the set of products.
- $\hat{\mathcal{R}} = (p_1, p_2) \ni p_1, p_2 \in \mathcal{P}$ are the product pairs that must be shipped together (if either is present in a box).
- $\widetilde{\mathcal{R}} = (p_1, p_2) \ni p_1, p_2 \in \mathcal{P}$ are the product pairs that must not be in the same box.
- $\hat{\mathcal{P}}_s \subset \mathcal{P}$ is the set of products that a subscriber must receive.
- $\widetilde{\mathcal{P}}_s \subset \mathcal{P}$ is the set of products that a subscriber must not receive.
- $\mathcal{S}$ is the set of subscribers.
- $\hat{\mathcal{S}}_p \subseteq \mathcal{S}$ is the set of subscribers (possibly empty) that must receive product p.
- $\widetilde{\mathcal{S}}_p \subset \mathcal{S}$ is the set of subscribers (possibly empty) that must not receive product p.
- $\mathcal{B}$ is the set of boxes.
- $\mathcal{A}$ is the set of attributes (e.g., value, weight, and volume).

Box- and Product-Related Data

- τ^l is the minimum number of box configurations to build.
- τ^u is the maximum number of box configurations to build.
- σ^l is the minimum number of subscribers to which a box configuration must be assigned.
- σ^u is the maximum number of subscribers to which a box configuration can be assigned.
- ϕ_p^l is the minimum number of subscribers to which product $p \in \mathcal{P}$ should be sent.
- ϕ_p^u is the maximum number of subscribers that should receive product $p \in \mathcal{P}$.
- γ^l is the minimum number of products each box must contain.
- γ^u is the maximum number of products each box can contain.
- μ_{ap} is the value of attribute $a \in \mathcal{A}$ for product $p \in \mathcal{P}$.
- α_a^l is the minimum total amount of attribute $a \in \mathcal{A}$ a box must have.
- α_a^u is the maximum total amount of attribute $a \in \mathcal{A}$ a box can contain.

Original MIP Formulation

The original formulation assumes that, at most, τ_u box configurations will be built, and thus the set $\mathcal{B} = 1, 2, \ldots, \tau_u$ is predetermined.

Decision Variables

- $y_b \in 0, 1$ is a binary variable representing whether box $b \in \mathcal{B}$ is built.
- $n_b \geq 0$ is the integer number of boxes to build (i.e., box size) for box $b \in \mathcal{B}$.
- $x_{pb} \in 0, 1$ is a binary variable representing whether to assign product $p \in \mathcal{P}$ to box $b \in \mathcal{B}$.
- $q_{pb} \geq 0$ is the integer quantity of product $p \in \mathcal{P}$ used by box $b \in \mathcal{B}$.
- $w_{sb} \in 0, 1$ is a binary variable representing whether subscriber $s \in \mathcal{S}$ is assigned to box $b \in \mathcal{B}$.
- $z_p \geq 0$ is the integer variable measuring the gap between the minimum quantity of product $p \in \mathcal{P}$ and the amount of that product that has been assigned across all boxes.

Constraints

The total quantity of product $p \in \mathcal{P}$ assigned to boxes should not exceed the maximum quantity for product p.

$$\sum_{b \in \mathcal{B}} q_{pb} \leq \phi_p^u \quad \forall p \in \mathcal{P}. \tag{1}$$

The total quantity of product $p \in \mathcal{P}$ assigned to boxes plus a slack variable should be at least the minimum quantity for product p.

$$\sum_{b \in \mathcal{B}} q_{pb} + z_p \geq \phi_p^\ell \quad \forall p \in \mathcal{P}. \tag{2}$$

The quantity assigned slack variable for product $p \in \mathcal{P}$ should not exceed the maximum quantity for product p.

$$z_p \leq \phi_p^u \quad \forall p \in \mathcal{P}. \tag{3}$$

At least the minimum number of boxes must be built.

$$\sum_{b \in \mathcal{B}} y_b \geq \tau^\ell. \tag{4}$$

The number of boxes built of box $b \in \mathcal{B}$ should be zero if box b is not being built.

$$n_b \le \sigma^u y_b \quad \forall b \in \mathcal{B}. \tag{5}$$

The number of boxes built of box $b \in \mathcal{B}$ should be at least the minimum if the box is built.

$$n_b \ge \sigma^\ell y_b \quad \forall b \in \mathcal{B}. \tag{6}$$

The total quantity of products assigned to box $b \in \mathcal{B}$ should be at least the quantity needed based on the box's size and the minimum number of products required in a box if the box is built.

$$\sum_{p \in \mathcal{P}} q_{pb} \ge \gamma^\ell n_b \quad \forall b \in \mathcal{B}. \tag{7}$$

The total quantity of products assigned to box $b \in \mathcal{B}$ should not exceed the quantity needed based on the box's size and the maximum number of products allowed in a box if the box is built.

$$\sum_{p \in \mathcal{P}} q_{pb} \le \gamma^u n_b \quad \forall b \in \mathcal{B}. \tag{8}$$

The size of box $b \in \mathcal{B}$ must be greater than or equal to the quantity of product $p \in \mathcal{P}$ assigned to the box.

$$q_{pb} \le n_b \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{9}$$

The quantity of product $p \in \mathcal{P}$ assigned to box $b \in \mathcal{B}$ cannot exceed the product's maximum available amount if the box is built and is zero if the box is not built.

$$q_{pb} \le \phi_p^u y_b \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{10}$$

Subscriber $s \in \mathcal{S}$ can be assigned to at most one box.

$$\sum_{b \in \mathcal{B}} w_{sb} \le 1 \quad \forall s \in \mathcal{S}. \tag{11}$$

The size of box $b \in \mathcal{B}$ should be equal to the number of subscribers assigned to the box.

$$\sum_{s \in \mathcal{S}} w_{sb} = n_b \quad \forall b \in \mathcal{B}. \tag{12}$$

The quantity of product $p \in \mathcal{P}$ assigned to box $b \in \mathcal{B}$ is greater than or equal to the size of box $b \in \mathcal{B}$ if product $p \in \mathcal{P}$ is assigned to the box.

$$q_{pb} \ge n_b - M(1 - x_{pb}) \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{13}$$

The quantity of product $p \in \mathcal{P}$ assigned to box $b \in \mathcal{B}$ cannot exceed the product's maximum available amount if the product is assigned to the box and is zero if the product is not assigned to the box.

$$q_{pb} \le \phi_p^u x_{pb} \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{14}$$

Product $p \in \mathcal{P}$ cannot be assigned to box $b \in \mathcal{B}$ if the box is not built.

$$x_{pb} \le y_b \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{15}$$

Subscribers cannot be assigned to box $b \in \mathcal{B}$ if they must receive product $p \in \mathcal{P}$ and product p is not assigned to box b.

$$\sum_{s \in \hat{\mathcal{S}}_p} w_{sb} \le |\hat{\mathcal{S}}_p| x_{pb} \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{16}$$

Subscribers cannot be assigned to box $b \in \mathcal{B}$ if they are ineligible for product $p \in \mathcal{P}$ and product p is assigned to box b.

$$\sum_{s \in \tilde{\mathcal{S}}_p} w_{sb} \le |\tilde{\mathcal{S}}_p|(1 - x_{pb}) \quad \forall b \in \mathcal{B},\ p \in \mathcal{P}. \tag{17}$$

Each box configuration $b \in \mathcal{B}$ should satisfy the attribute constraints for each attribute $a \in \mathcal{A}$ if box b is built.

$$\alpha_a^\ell y_b \le \sum_{p \in \mathcal{P}} \mu_{ap} x_{pb} \le \alpha_a^u y_b \quad \forall b \in \mathcal{B},\ a \in \mathcal{A}. \tag{18}$$

The number of products assigned to box $b \in \mathcal{B}$ should not exceed the maximum number of products allowed in a box if the box is built.

$$\sum_{p \in \mathcal{P}} x_{pb} \le \gamma^u y_b \quad \forall b \in \mathcal{B}. \tag{19}$$

The number of products assigned to box $b \in \mathcal{B}$ should be at least the minimum number of products required for a box if the box is built.

$$\sum_{p \in \mathcal{P}} x_{pb} \ge \gamma^\ell \cdot y_b \quad \forall b \in \mathcal{B}. \tag{20}$$

For each $(p_1, p_2) \in \hat{\mathcal{R}}$, if product p_1 is assigned to box $b \in \mathcal{B}$, product p_2 must also be assigned to the box, and if p_1 is not assigned to the box, p_2 cannot be assigned to the box.

$$x_{p_1 b} = x_{p_2 b} \quad \forall (p_1, p_2) \in \hat{\mathcal{R}}, b \in \mathcal{B}. \tag{21}$$

For each $(p_1, p_2) \in \widetilde{\mathcal{R}}$, products p_1 and p_2 cannot be in the same box.

$$x_{p_1 b} + x_{p_2 b} \le 1 \quad \forall (p_1, p_2) \in \widetilde{\mathcal{R}}, b \in \mathcal{B}. \tag{22}$$

Objective Function

The original formulation used a composite objective function that included four parts:

1. Count the number of boxes built.
2. Count how many subscribers received boxes.
3. Introduce a perturbation to count how many product/box combinations were used.
4. Penalize not meeting the minimum requirements for use of the products.

$$\text{maximize } \frac{1}{2}\sum_{b \in \mathcal{B}} n_b + \frac{1}{2}\sum_{s \in \mathcal{S}, b \in \mathcal{B}} w_{sb} + \sum_{b \in \mathcal{B}}\sum_{p \in \mathcal{P}} \frac{1}{3{,}600} \cdot x_{pb} - \sum_{p \in \mathcal{P}} \frac{2}{\left(\phi_p^\ell + 1\right)} \cdot z_p. \tag{23}$$

New MIP Formulation

In the original formulation, the binary decision variables x_{pb} exist for each product $p \in \mathcal{P}$ and for a fixed number of possible box configurations $1 \le b \le \tau^u$. As we mention in the Original MIP Formulation section, this predefines the set $\mathcal{B} = 1, 2, \ldots, \tau_u$. It also creates $2|\mathcal{P}|\tau^u$ decision variables x_{pb} and q_{pb} corresponding to the possibility that any product can appear in any of the τ^u boxes. The new formulation, which we will refer to as the *full master problem*, assumes that all possible configurations of boxes exist and then uses a strategy to dynamically generate these configurations using mixed-integer programming. The concept of this formulation is motivated by the concepts pioneered by Gilmore and Gomory (1961) for solving the cutting-stock problem. One main difference between the concept of this formulation and that of the cutting-stock problem and other similar applications is that the additional box configurations increase both the number of decision variables *and* the number of constraints of the full master problem.

We define $\overline{\mathcal{B}}$ to be all possible box configurations that satisfy the box quantity, attribute, and pairing rules, as we describe above. For each $b \in \overline{\mathcal{B}}$, there is a set of products $\overline{\mathcal{P}}_b \subset \mathcal{P}$ that are *fixed* in that particular box configuration. We also denote by $\overline{\mathcal{B}}_p$ the set of box configurations containing product $p \in \mathcal{P}$, with the property that $b \in \overline{\mathcal{B}}_p$ if and only if $p \in \overline{\mathcal{P}}_b$. In addition, there is a *fixed* set of subscribers $\overline{\mathcal{S}}_b$ that represents the set of subscribers eligible to be assigned to box b, based on the sets $\hat{\mathcal{S}}_p$ and $\tilde{\mathcal{S}}_p$. We also denote by $\overline{\mathcal{B}}^s$ the set of box configurations to which subscriber $s \in \mathcal{S}$ can be assigned, with the property that $b \in \overline{\mathcal{B}}^s$ if and only if $s \in \overline{\mathcal{S}}_b$. Note that the set $\overline{\mathcal{B}}$ is quite large. For example, with 100 products and six products per box, there could be $\binom{100}{6}$ (more than 1.1 billion) possible box configurations.

Decision Variables

- $y_b \in 0, 1$ is a binary variable representing whether box $b \in \overline{\mathcal{B}}$ is built.
- $n_b \ge 0$ is the integer number of boxes to build (i.e., box size) for box $b \in \overline{\mathcal{B}}$.
- $q_{pb} \ge 0$ is the integer quantity of product $p \in \overline{\mathcal{P}}_b$ used by box $b \in \overline{\mathcal{B}}$.
- $w_{sb} \in 0, 1$ is a binary variable representing whether subscriber $s \in \overline{\mathcal{S}}_b$ is assigned to box $b \in \overline{\mathcal{B}}$.
- $z_p \ge 0$ is the integer variable measuring the gap between the minimum quantity of product $p \in \mathcal{P}$ and the amount that has been assigned to boxes.

In Table 1, note the reduction in dimensionality of the q_{pb} and w_{sb} variables. In the new formulation, these variables are created only if product $p \in \mathcal{P}$ is in box $b \in \overline{\mathcal{B}}$ and subscriber $s \in \mathcal{S}$ is eligible to receive box b. In the original formulation, each box configuration $b \in \mathcal{B}$ could contain any product; in the new formulation, a box configuration $b \in \overline{\mathcal{B}}$ contains a fixed set of products. In the old formulation, a subscriber could be assigned to any box. In the new formulation, a subscriber can only be assigned to a box containing products that the subscriber must receive (if any) and not containing products that the subscriber must not receive (if any). We emphasize again that in the original formulation, the model determined which products went in each box configuration. In the new formulation, the products in each box configuration are known.

Table 1. This Table Contrasts the Dimensions of Each Decision Variable Between the Two Formulations

Variable	Old formulation	New formulation
n_b	τ^u	$\lvert\bar{\mathcal{B}}\rvert$
x_{pb}	$\lvert\mathcal{P}\rvert \times \tau^u$	Removed
q_{pb}	$\lvert\mathcal{P}\rvert \times \lvert\mathcal{B}\rvert$	$\sum_{b\in\bar{\mathcal{B}}} \lvert\bar{\mathcal{P}}_b\rvert$
w_{sb}	$\lvert\mathcal{S}\rvert \times \tau^u$	$\sum_{b\in\bar{\mathcal{B}}} \lvert\bar{\mathcal{S}}_b\rvert$
z_p	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{P}\rvert$

Constraints

The constraints for the reformulation are replacements for Constraints (1)–(12). To make the correspondence easier to follow, we use the suffix "r" in the equation numbers to correspond to revised equations and explain the revision; for constraints that are unchanged, we repeat them with their original equation numbers.

The total quantity of product $p \in \mathcal{P}$ assigned to boxes should not exceed the maximum quantity for product p.

$$\sum_{b\in\overline{\mathcal{B}}_p} q_{pb} \le \phi_p^u \quad \forall p \in \mathcal{P}. \tag{1r}$$

The total quantity of product $p \in \mathcal{P}$ assigned to boxes plus a slack variable should be at least the minimum quantity for product p.

$$\sum_{b\in\overline{\mathcal{B}}_p} q_{pb} + z_p \ge \phi_p^\ell \quad \forall p \in \mathcal{P}. \tag{2r}$$

The replacements for Constraints (1) and (2) take advantage of the knowledge that the boxes containing product p are known because all possible box configurations have been enumerated. In the original formulation, the box configurations are unknown.

The quantity assigned for the slack variable for product $p \in \mathcal{P}$ should not exceed the maximum quantity for product p.

$$z_p \le \phi_p^u \quad \forall p \in \mathcal{P}. \tag{3}$$

Constraint (3) remains unchanged.

At least the minimum number of boxes must be built.

$$\sum_{b\in\overline{\mathcal{B}}} y_b \ge \tau^\ell. \tag{4r}$$

Constraint (4) now sums over the known box configurations $\overline{\mathcal{B}}$ rather than the fixed set $\mathcal{B}$.

The number of boxes of configuration $b \in \overline{\mathcal{B}}$ that are built must be zero if no box of configuration b is built.

$$n_b \le \sigma^u y_b \quad \forall b \in \overline{\mathcal{B}}. \tag{5r}$$

The number of boxes of configuration $b \in \overline{\mathcal{B}}$ that are built must be at least the minimum if the configuration is built.

$$n_b \ge \sigma^\ell y_b \quad \forall b \in \overline{\mathcal{B}}. \tag{6r}$$

Constraints (5) and (6) are the same, except that they are stated for every box configuration, rather than for the limited number τ^u of box configurations.

The total quantity of products assigned to box configuration $b \in \overline{\mathcal{B}}$ should be at least the quantity needed based on the box configuration's size and the minimum number of products required in a box if the box is built.

$$\sum_{p \in \overline{\mathcal{P}}_b} q_{pb} \ge \gamma^\ell n_b \quad \forall b \in \overline{\mathcal{B}}. \tag{7r}$$

The total quantity of products assigned to box configuration $b \in \overline{\mathcal{B}}$ should not exceed the quantity needed based on the box configuration's size and the maximum number of products allowed in a box if the box is built.

$$\sum_{p \in \overline{\mathcal{P}}_b} q_{pb} \le \gamma^u n_b \quad \forall b \in \overline{\mathcal{B}}. \tag{8r}$$

Constraints (7) and (8) are the same, except that they are stated for every known box configuration, as opposed to the limited number τ^u of box configurations. In addition, the sums are over the sets $\overline{\mathcal{P}}_b$ rather than the entire product set $\mathcal{P}$.

The size of box configuration $b \in \overline{\mathcal{B}}$ is equal to the quantity of product $p \in \overline{\mathcal{P}}_b$ assigned to the box configuration.

$$q_{pb} = n_b \quad \forall b \in \overline{\mathcal{B}},\ p \in \overline{\mathcal{P}}_b. \tag{9r}$$

Constraint (9) becomes an equality rather than an inequality because Constraints (5r)–(8r) link the variable y_b, representing whether a box is built with the decision variables q_{pb} and n_b. In addition, Constraint (9r) is stated for every known box configuration, as opposed to the limited number τ^u of box configurations. Moreover, it is only stated for the products $p \in \overline{\mathcal{P}}_b$ that are known to be in box configuration b, rather than for all products.

The quantity of product $p \in \overline{\mathcal{P}}_b$ assigned to box configuration $b \in \overline{\mathcal{B}}$ cannot exceed the product's maximum available amount if the box is built and is zero if the box is not built.

$$q_{pb} \le \phi^u_p y_b \quad \forall b \in \overline{\mathcal{B}},\ p \in \overline{\mathcal{P}}_b. \tag{10r}$$

Constraint (10) is the same, except that it is stated for every known box configuration, as opposed to the limited number τ^u of box configurations. In addition, it is only stated for the products $p \in \overline{\mathcal{P}}_b$ that are known to be in box configuration b, rather than for all products.

Subscriber $s \in \mathcal{S}$ can be assigned to at most one box configuration.

$$\sum_{b \in \overline{\mathcal{B}}^s} w_{sb} \le 1 \quad \forall s \in \mathcal{S}. \tag{11r}$$

Constraint (11) is the same, except that the sum is over only the box configurations that subscriber s is eligible to receive.

The size of box configuration $b \in \overline{\mathcal{B}}$ must be equal to the number of subscribers assigned to the box.

$$\sum_{s \in \overline{\mathcal{S}}_b} w_{sb} = n_b \quad \forall b \in \overline{\mathcal{B}}. \tag{12r}$$

Constraint (12) is the same, except that it is stated for every known box configuration, as opposed to the limited number τ^u of box configurations. In addition, the sum is over only the subscribers $s \in \overline{\mathcal{S}}_b$ that are eligible to receive box configuration b.

With the removal of the decision variables x_{pb}, Constraints (13)–(15) are no longer necessary. Those constraints were stated to represent the *decision* of placing a product p in box b to the decision variable y_b, indicating whether the box was built and to the number of products in box b. This linking is no longer needed, because the decisions of placing product p in box configuration b are represented by the set $\overline{\mathcal{P}}_b$. The method for handling Constraints (18)–(22) is described in the Generating Box Configurations section.

We also introduced a constraint that would possibly tighten Constraint (1r): The total number of box configurations containing product $p \in \mathcal{P}$ is limited by the maximum quantity of product p and the minimum number of subscribers per box σ^ℓ:

$$\sum_{b \in \overline{\mathcal{B}}_p} y_b \le \left\lfloor \frac{\phi^u_p}{\sigma^\ell} \right\rfloor \quad \forall p \in \mathcal{P}. \tag{24}$$

Because the set $\overline{\mathcal{B}}$ is quite large, this formulation has many more constraints and variables than the original formulation. The remainder of this paper describes a solution strategy that is similar to column-generation techniques and is based on mixed-integer programming, which avoids generating the entire set $\overline{\mathcal{B}}$ of all possible box configurations.

Objective Function

Instead of using a compositive objective function, we will use hierarchical optimization. The objectives, in priority order, are

Maximize the number of subscribers receiving a box

$$\text{maximize} \sum_{b \in \overline{\mathcal{B}},\, s \in \overline{\mathcal{S}}_b} w_{sb}. \tag{25}$$

Minimize the required number of box configurations

$$\text{minimize} \sum_{b \in \overline{\mathcal{B}}} y_b. \tag{26}$$

Minimize the violation of product minimums

$$\text{minimize} \sum_{p \in \mathcal{P}} \frac{1}{\left(\phi_p^{\ell} + 1\right)} z_p. \tag{27}$$

The second objective listed above (i.e., minimizing the required number of box configurations) is needed because the set $\overline{\mathcal{B}}$ is quite large, and the goal is to choose a subset of those possible configurations of, at most, size τ^u. We handle this requirement in one of two ways via an option chosen by the user:

1. **Hard**: Explicitly include the constraint

$$\sum_{b \in \overline{\mathcal{B}}} y_b \le \tau^u. \tag{28}$$

Do not include the Objective (26).

2. **Soft**: Do not include Constraint (28), but use Objective (26) to reduce the number of required boxes.

The computational results indicated that there was a preference for the **Hard** strategy in terms of overall performance. However, the **Soft** option provides an additional benefit, because it allows Birchbox to determine whether its source data will support assigning every subscriber a box, independent of the target for the number of boxes needed.

Generating Box Configurations

Rather than generate all possible box configurations, we create a MIP, termed the *pattern- generation problem* or *subproblem*, which is used to generate valid patterns of products that can be placed together in a box configuration. This pattern-generation problem uses binary decision variables $\overline{x}_p \in 0,1,\ p \in \mathcal{P}$. There is a subproblem that corresponds to each subscriber $s \in \mathcal{S}$, and solving a subproblem yields a single box configuration that will be dynamically added to the set $\overline{\mathcal{B}}$.

Subproblem Constraints

We replace Equations (16)–(22) with the constraints described in this section.

Subscriber $s \in \mathcal{S}$ must receive product $p \in \mathcal{P}$ if required.

$$\overline{x}_p = 1 \quad \forall p \in \hat{\mathcal{P}}_s. \tag{16r}$$

Subscriber $s \in \mathcal{S}$ should not receive product $p \in \mathcal{P}$ if that subscriber is ineligible for product p.

$$\overline{x}_p = 0 \quad \forall p \in \widetilde{\mathcal{P}}_s. \tag{17r}$$

In the original formulation, Constraints (16) and (17) linked together the decision variables x_{pb} representing whether product p was placed in box b and the decision variables w_{sb} representing whether subscriber s was assigned to box b. Now, this linking is implicit due to the construction of the sets $\overline{\mathcal{S}}_b$ and $\overline{\mathcal{B}}_s$ that represent whether subscriber s can be assigned to box b. For the subproblems that generate valid box configurations for subscriber s, we simply have to add bound constraints on $\overline{x}_p$ to force product p to be in or out of the box if subscriber s must receive or not receive product p, respectively.

Each box-configuration pattern should satisfy the attribute constraints for each attribute $a \in \mathcal{A}$.

$$\alpha_a^{\ell} \le \sum_{p \in \mathcal{P}} \mu_{ap} \overline{x}_p \le \alpha_a^{u} \quad \forall a \in \mathcal{A}. \tag{18r}$$

The number of products assigned to a configuration pattern should not exceed the maximum number of products allowed in a box.

$$\sum_{p \in \mathcal{P}} \overline{x}_p \le \gamma^{u}. \tag{19r}$$

The number of products assigned to a box should be at least the minimum number of products required for a box if the box is built.

$$\sum_{p \in \mathcal{P}} \overline{x}_p \ge \gamma^{\ell}. \tag{20r}$$

In the original formulation, Constraints (18), (19), and (20) included the variable y_b, which represents whether the box b would be built. In the subproblem, we are creating proposed box configurations, and the master problem will determine whether that box configuration is used.

For each $(p_1, p_2) \in \hat{\mathcal{R}}$, if product p_1 is assigned to a box, product p_2 must also be assigned to the box, and if p_1 is not assigned to the box, p_2 cannot be assigned to the box.

$$\overline{x}_{p_1} = \overline{x}_{p_2} \quad \forall (p_1, p_2) \in \hat{\mathcal{R}}. \tag{21r}$$

For each $(p_1, p_2) \in \widetilde{\mathcal{R}}$, products p_1 and p_2 cannot be in the same box.

$$\overline{x}_{p_1} + \overline{x}_{p_2} \le 1 \quad \forall (p_1, p_2) \in \widetilde{\mathcal{R}}. \tag{22r}$$

Constraints (18)–(22) are the same as in the original formulation; however, they represent the constraints of a *single* box configuration, whereas in the original formulation, the constraints were stated for the maximum number τ^u of box configurations.

With these constraints, suppose we generate a feasible solution $\overline{x}_p^b$, where b corresponds to the new box that will correspond to this pattern. We can then compute $\overline{\mathcal{P}}_b = \{p \in \mathcal{P} : \overline{x}_p^b = 1\}$ to represent the set of products $p \in \mathcal{P}$ in box configuration b. For future generation of patterns, we then add the constraint:

Do not regenerate a pattern

$$\sum_{p \in \overline{\mathcal{P}}_b} \overline{x}_p - \sum_{p \notin \overline{\mathcal{P}}_b} \overline{x}_p \le \left|\overline{\mathcal{P}}_b\right| - 1 \quad \forall b \in \mathcal{B}. \tag{29}$$

As new patterns are generated, these constraints are added to the pattern-generation problem for every

subscriber s so that a pattern is not regenerated and guarantee that a pattern will not be regenerated by the subproblem MIP.

Subproblem Objective Function

The subproblem objective function is modified during the course of the RIP technique described in the section The RIP Technique. For notation purposes, we will write that function as

$$\text{maximize} \quad \sum_{p\in\mathcal{P}} \overline{\pi}_p \overline{x}_p. \tag{30}$$

As the RIP technique generates new subproblems, the coefficients $\overline{\pi}_p$, $p \in \mathcal{P}$ are adjusted as described in that section.

Subproblem Dimensions

The subproblems have exactly $|\mathcal{P}|$ binary decision variables. The number of constraints depends on the sizes of the sets $\mathcal{A}$, $\hat{\mathcal{R}}$, and $\overline{\mathcal{R}}$, plus the number of patterns generated, which will be the size of the set $\overline{\mathcal{B}}$ that dynamically grows. In the experiments listed in the Computational Results and Impact section, the number of generated patterns never exceeded 700, which kept the size of the subproblem mixed-integer program small for the Gurobi solver.

Subproblem Solution Procedure

After a subproblem is solved for one subscriber $s_1 \in \mathcal{S}$ to determine a new box b and the set $\overline{\mathcal{P}}_b$, the next subproblem will be solved for a different subscriber $s_2 \in \mathcal{S}$ such that $s_2 \neq s_1$. The difference between these successive subproblems is due to Constraints (16r) and (17r) related to the individual subscribers' product requirements, the addition of Constraint (29) to eliminate the regeneration of the newly generated pattern, and a possible change in Objective Function (30), as we describe in the section The RIP Technique. The most recently generated feasible solution is no longer feasible; however, the Gurobi optimizer can use that solution (and other solutions previously generated) as a MIP start to influence the solution process for the next subscriber s_2.

The RIP Technique

The RIP technique is motivated by significant improvements over the past 20 years in the ability to practically solve mixed-integer programs, as Bixby (2012) and Achterberg and Wunderling (2013) describe. The RIP technique repetitively solves the pattern-generation subproblem we describe in the Generating Box Configurations section to generate a subset $\hat{\mathcal{B}} \subset \overline{\mathcal{B}}$ of all possible box configurations. This subset is then used to generate the MIP we describe in the New MIP Formulation section, where the subset $\hat{\mathcal{B}}$ is used to define the set $\overline{\mathcal{B}}$ and the related sets $\overline{\mathcal{B}}_p$ and $\overline{\mathcal{B}}^s$ and, by definition, the sets $\overline{\mathcal{P}}_b$ and $\overline{\mathcal{S}}_b$. This version of that MIP is termed the *restricted master problem*. For notational purposes, we will use $\overline{\mathcal{B}}$ to refer to both the complete set of box-configuration patterns in the full master problem and the growing set of box-configuration patterns in the restricted master problem.

Since Dantzig and Wolfe (1960) introduced the Dantzig-Wolfe decomposition procedure, the concepts of using a restricted master program with an associated subproblem for generating columns of the restricted master program have been used in a variety of applications. Desaulniers et al. (2005) provide many articles that describe various examples of column generation. However, the concept of generating patterns that add both columns *and* rows to the restricted master program seems not to have been addressed in the literature. In general, column-generation techniques are based on keeping the number of constraints in the restricted master problem fixed. For the master problem we describe in the New MIP Formulation section, as the set $\overline{\mathcal{B}}$ grows, there are four different variable classes that grow in dimension. In addition, Constraints (5r)–(10r) must be added to the restricted master problem to correspond to new patterns added to $\overline{\mathcal{B}}$. Because new patterns cause the restricted master problem to grow in both the number of variables *and* the number of constraints, typical branch-and-price techniques, as well as other column-generation techniques, do not apply. We address this by solving each restricted master problem as a mixed-integer program. Vanderbeck (2005, p. 353) writes: "A standard heuristic is to initialize the master with a set of heuristically generated columns and to solve the MIP master problem restricted to that initial set… However, there is typically no guarantee that such a static restricted set of columns holds a feasible integer solution." The RIP technique addresses this latter issue by specifying coefficients for the objective function of the subproblem to generate new patterns that accelerate the convergence to finding an integer-feasible solution.

One reason that the RIP technique works well is because it can take advantage of many of the advanced features of the Gurobi solver. Branch-and-price techniques, as Barnhart et al. (1998) describe, are based on generating columns (new decision variables) at each node in the branch-and-bound tree. This means that many of the techniques used to improve the performance of MIP solvers, such as presolve (Achterberg et al. 2019), primal heuristics (Fischetti and Lodi 2011), and cutting planes (Marchand et al. 2002), are not available. Although recent work (Sadykov et al. 2019) has extended primal heuristics in the context of branch and price, these implementations most likely do not have all of the features of the heuristics built into Gurobi. Furthermore, as we mention above, the growth in the restricted master problem in both

dimensions prohibits the use of methods that assume the number of constraints is fixed. By using the full power of the MIP solver, the RIP technique accelerates the solution of the Birchbox product-assortment problem.

Although the RIP technique could be used to find a provably optimal solution to the Birchbox problem, this is not required by the business. Discussions with Birchbox personnel revealed that the company had three main goals for the optimization process:

1. Ensure that each subscriber is assigned to a box configuration.

2. Use at most τ^u different box configurations, because additional configurations increase the production costs of assembling the boxes.

3. Attempt to meet the minimum product-shipment requirements represented by ϕ_p^ℓ.

With respect to the first goal, the nature of the process used to determine the sets $\hat{\mathcal{P}}_s$ and $\widetilde{\mathcal{P}}_s$, which represent the products that subscriber $s \in \mathcal{S}$ must and must not receive, respectively, along with the maximum inventory amounts ϕ_p^u, may make assigning a configuration for each subscriber impossible. For example, if 1,000 subscribers must receive product number 69, but only 500 units of that product are available, assigning each subscriber a box configuration will be impossible. In addition, if the **Hard** option was chosen, the value of τ^u may be too small to warrant a solution. (Note that implementing a method to help diagnose these types of infeasibilities was beyond the scope of our Birchbox project.) We observed that if the RIP technique was slowly converging in the **Relax** step described below, then it was likely that inventory amounts were too low; there were too many restrictions on the products that subscribers must receive or not receive; or that the **Hard** option was used, and τ^u was too small. Birchbox would then use this information about slow convergence to modify its input values to be able to make assigning each subscriber a box more likely.

With these goals in mind, we designed the Birchbox RIP technique with the following steps, which we explain in the subsections of this section, except for the **Boxes** and **Product** steps, which need no further detail.

1. **Bootstrap**: Generate an initial set of patterns by repetitively solving the pattern-generation problem, which we describe in the Generating Box Configurations section, and form the restricted master problem.

2. **Relax**: Solve the linear programming (LP) relaxation of the restricted master problem with the first objective function (25) using multiple rounds of pattern generation and concepts from sifting, as Bixby et al. (1992) describe. Because of Constraint (11r), it is only possible to reach the first goal if the optimal objective function of this linear program is $|\mathcal{S}|$ (i.e., the number of subscribers). If that target value cannot be reached, and all patterns have been generated, terminate and indicate infeasibility.

3. **Populate**: Using the solution from the **Relax** step combined with patterns that have already been generated, choose patterns that are more likely to be used in the MIP to assign each subscriber a box.

4. **Subscribe**: Solve the relaxed master problem as a MIP using the first Objective Function (25). As soon as the objective function value reaches $|\mathcal{S}|$, move to the **Boxes** step. If the MIP solver proves that the upper bound for the MIP is less than $|\mathcal{S}|$, or if the MIP solver starts a search and hits a small node limit (we chose 100), generate additional patterns based on the current MIP solution, and solve the new restricted master problem with the additional patterns.

5. **Boxes**: At this point, we know that assigning each subscriber a box s is possible; therefore, we add the constraint

$$\sum_{b \in \overline{\mathcal{B}}, s \in \mathcal{S}} w_{sb} \geq |\mathcal{S}|. \tag{31}$$

6. If the **Hard** option was used, we skip to the **Product** step. Otherwise, with the **Soft** option, we then solve the restricted master problem using the same set of patterns, but with Objective (26). We terminate the solving process as soon as the objective value is less than or equal to τ^u.

7. **Product**: At this point, we now know that assigning each subscriber a box s is possible, using at most τ^u different configurations. If using the **Soft** option, we add Constraint (28) to the restricted master problem. (In the case of the **Hard** option, that constraint was already present.) Either way, we then solve the restricted master problem using Objective (27) to optimality. We let the user choose to abort the optimization procedure by providing details of the values of z_p, $p \in \mathcal{P}$ for each newly found feasible solution. We also set a branching node limit of 100, because we observed that the solutions were not improving much once branching began, and Birchbox personnel informed us that having an optimal solution for reducing the gap on minimum product amounts was less important to the business.

Bootstrap

The first step in the RIP technique is to generate an initial set of patterns by repetitively solving the pattern-generation problem described in the Generating Box Configurations section. In the bootstrap phase, we do not solve the pattern-generation problem to optimality. Instead, we stop the MIP solver as soon as a feasible solution is found. This increases the speed of the pattern-generation process. We start by picking the first subscriber $s \in \mathcal{S}$ and using an objective function of $\overline{\pi}_p = \phi_p^u$ for all $p \in \mathcal{P}$. This incents the solution to include products with the highest available inventory amounts. After

a pattern $\overline{b}$ is generated for a subscriber s, that pattern is checked to see if it is eligible for any other subscribers $\hat{s} \in \mathcal{S}$, where $\hat{s} \neq s$ by considering the sets $\hat{\mathcal{P}}_{\hat{s}}$ and $\overline{\mathcal{P}}_{\hat{s}}$. Note that Constraints (16r) and (17r) are subscriber specific, whereas Constraints (18r)–(22r) and (28) apply to all subscribers, so checking whether pattern $\overline{b}$ is eligible for subscriber $\hat{s}$ is straightforward. If the pattern is eligible for less than σ^{ℓ} subscribers, the pattern is discarded, because it would then be impossible to satisfy Constraint (6r) in the master problem, which states that each box configuration must be sent to at least σ^{ℓ} subscribers.

As patterns are generated, a count PatCnt[s] of how many patterns are feasible for each subscriber $s \in \mathcal{S}$ is maintained. In addition, a count ProdCnt[p] of how often each product $p \in \mathcal{P}$ is used across the generated patterns is also maintained. As the pattern-generation process proceeds, the subscriber to use is based on choosing a subscriber $s \in \mathcal{S}$ with the smallest value of PatCnt[s]. The pattern-generation process is terminated if all patterns have been generated, which occurs if the pattern-generation problem is infeasible without Constraints (16r) and (17r). It is also terminated when the value of PatCnt[s] is five for each subscriber $s \in \mathcal{S}$, and the total number of generated patterns is at least $2\tau^{u}$.

During the pattern-generation process, the objective function values $\overline{\pi}_p$ are dynamically modified after each pattern is generated, based on the values of ProdCnt[p]:

$$\overline{\pi}_p = \frac{\phi_p^u \max_{\hat{p} \in \mathcal{P}} \texttt{ProdCnt}[\hat{p}]}{1 + \texttt{ProdCnt[p]}} \quad \forall p \in \mathcal{P}. \tag{32}$$

This dynamic change to the objective function incents products that are rarely used to be used more often during the pattern-generation process.

If after this process has completed, there exists $p \in \mathcal{P}$ such that $\phi_p^{\ell} > 0$ and the value of ProdCnt[p] is zero, additional patterns are generated to ensure that at least 10 patterns are available for any product that has a desired minimum product count. The subscriber to use in the pattern-generation problem is chosen based on which subscribers are eligible to receive those products.

Relax

Given an initial set of patterns from the **Bootstrap** process, the corresponding LP relaxation is solved. We use a feature of the Gurobi optimizer called "deterministic concurrent optimization," which simultaneously uses the dual simplex, interior point, and primal simplex methods to solve the linear program, terminating as soon as one of the methods finds an optimal solution. If the optimal objective value is $|\mathcal{S}|$, we move to the **Populate** phase, because we have an LP solution that assigns each subscriber a box, but some subscribers may have fractional assignments.

If the optimal objective value is smaller than $|\mathcal{S}|$, we then use a variant of the sifting procedure described in Bixby et al. (1992) to add and purge patterns. We compute the reduced costs and basis status of the variables w_{sb} for each $b \in \overline{\mathcal{B}}$, $s \in \overline{\mathcal{S}}_b$. For any nonbasic variables with a negative reduced cost, we remove them from the relaxed restricted master problem. After this removal, we consider for each box configuration $b \in \overline{\mathcal{B}}$ the number of w_{sb} variables that remain. If that count is smaller than σ^{ℓ}, then the entire pattern b is removed from the restricted master problem. Note that these patterns may be reintroduced later when solving the restricted master problem as a MIP.

We then retrieve the dual values π_p^1 and π_p^{24} for all $p \in \mathcal{P}$ associated with Constraints (1r) and (24), respectively, which correspond to the maximum quantity of products that can be shipped to all subscribers and the maximum number of boxes that can be used per product. Note that in the current solution to the relaxed restricted problem, if a product p has excess inventory, then the corresponding constraint will have $\pi_p^1 = 0$. Also, $\pi_p^{24} = 0$ in this case if ϕ_p^u is not a multiple of σ^{ℓ}. We set $\hat{\pi}_p = \left(\pi_p^1 + \pi_p^{24}\right)$ to represent the totals of these two dual values.

For any patterns $b \in \overline{\mathcal{B}}$ that were previously removed, if

$$1 - \sum_{p \in \overline{\mathcal{P}}_b} \hat{\pi}_p > 0, \tag{33}$$

then we place b back in the set $\overline{\mathcal{B}}$ for use in the restricted master problem. This is equivalent to computing the reduced cost for w_{sb} in the restricted master problem and adding back variables with a positive reduced cost.

To generate additional patterns, we do not use the direct dual values; instead, we use the values $\hat{\pi}_p$ and the values v_p for the optimal slack values of Constraint (1r) to determine the objective values $\overline{\pi}_p$ for the Objective Function (30) for the subproblems:

$$\overline{\pi}_p = \begin{cases} \frac{1}{2} \min_{p \in \mathcal{P}: \hat{\pi}_p > 0} \hat{\pi}_p \left(\frac{v_p}{1 + \max_{p \in \mathcal{P}} v_p} - 1 \right) & \text{if } \hat{\pi}_p = 0 \\ -\hat{\pi}_p & \text{if } \hat{\pi}_p > 0. \end{cases} \tag{34}$$

The purpose of this perturbation is to account for the primal degeneracy of the restricted master problem that yields multiple dual solutions. Here, we perturb the dual values that are zero, which correspond to products that have remaining inventory. Because the subproblem is a maximization problem, this will prioritize generating patterns that have used up inventory to provide more combinatorial flexibility in how those products can be used in a final solution. Note that the perturbation guarantees that if $\hat{\pi}_{p_1} = 0$ and $\hat{\pi}_{p_2} > 0$, then $\overline{\pi}_{p_1} > \overline{\pi}_{p_2}$.

When solving each additional LP relaxation, we use the optimal basis from the previous LP relaxation to

warm-start the LP optimizers. We continue to use the concurrent optimizers so that if the warm start helped the primal or dual simplex method, it would provide the optimal solution, but, if not, the interior point method would provide the best solution. We observed that 75% of the time, the interior point method was the best algorithm; 20% of the time, the primal simplex was best; and 5% of the time, dual simplex was best. This makes sense, because toward the end of the **Relax** phase, the LP relaxations are closer to optimality, and we have fundamentally added only columns to the problem. The slack and artificial variables for the additional rows are considered basic in the initial solution, and the solver can use the previous basis plus these slack and artificial variables as an effective warm start.

Populate

After the **Relax** phase has completed, we have solved the linear-programming relaxation of the restricted master problem for an existing set of patterns that we will represent as $\hat{\mathcal{B}}$. We first put back into that set any previously deleted patterns that we removed due to negative reduced costs, as we explain above. For each $b \in \hat{\mathcal{B}}$, there is a set of eligible subscribers $\hat{\mathcal{S}}_b$. If we include all the corresponding variables w_{sb} in the MIP version of the relaxed master problem, this creates too many columns for the MIP optimizer; therefore, the **Populate** phase judiciously chooses the subscribers that are eligible for each pattern. This phase is choosing a *new* subset $\tilde{\mathcal{B}} \subset \hat{\mathcal{B}}$ to initiate the MIP optimization on the MIP version of the restricted master problem. The goal of the procedure is to have $\overline{m} = 10\sigma^{\ell}$ subscribers per pattern and $\hat{m} = 50$ patterns per subscriber (if the subscriber is eligible for that many different box configurations). The choices of these limits seemed to work well over the test set that we used and provided a proper balance between the diversity of the patterns per subscriber and subscribers per pattern and, in addition, made the size of the resulting MIP manageable for the Gurobi MIP solver.

A key aspect of this procedure is that we build up a set of box/subscriber pairs (b,s), where we consider all the patterns $b \in \hat{\mathcal{B}}$ generated so far and the associated subscribers $s \in \hat{\mathcal{S}}_b$ eligible to receive box configuration b. For each box b, we do not necessarily include all eligible subscribers for that box. As the set $\tilde{\mathcal{B}}$ is populated, the sets $\tilde{\mathcal{S}}_b$ for each $b \in \tilde{\mathcal{B}}$ and $\tilde{\mathcal{B}}^s$ for each $s \in \mathcal{S}$ are also dynamically generated. The sets $\tilde{\mathcal{S}}_b$ and $\tilde{\mathcal{B}}^s$ are analogous to the sets $\overline{\mathcal{S}}_b$ and $\overline{\mathcal{B}}^s$ described in the New MIP Formulation section. The **Populate** procedure chooses pairs (b,s) that will always satisfy the following properties:

$$b \in \tilde{\mathcal{B}}^s \Leftrightarrow s \in \tilde{\mathcal{S}}_b$$
$$\tilde{\mathcal{B}} = \bigcup_{s \in \mathcal{S}} \tilde{\mathcal{B}}^s. \tag{35}$$

The goal is that $|\tilde{\mathcal{B}}_s| \geq \hat{m}$ and $|\tilde{\mathcal{S}}_b| \geq \overline{m}$ at the end of the procedure. The construction of the sets $\tilde{\mathcal{B}}$, $\tilde{\mathcal{B}}$, and $\tilde{\mathcal{S}}_b$ is accomplished using the following steps:

1. First, we include any pairs (b,s) involved in the linear-programming solution obtained in the **Relax** step. Mathematically, initialize $\tilde{\mathcal{B}}_s$ and $\tilde{\mathcal{S}}_b$ to include any box $b \in \hat{\mathcal{B}}$ and $s \in \mathcal{S}$ such that the optimal value $w_{sb} > 0$ or w_{sb} is basic in that LP solution.

2. We now add eligible subscribers to the patterns chosen in the previous step to achieve our goal of a minimum number $\overline{m}$ of subscribers per pattern. For each box $b \in \tilde{\mathcal{B}}$, such that $|\tilde{\mathcal{S}}_b| < \overline{m}$, choose a subscriber $s \in \hat{\mathcal{S}}_b$ with $|\hat{\mathcal{B}}_s|$ being minimal, and update the sets $\tilde{\mathcal{B}}_s$ and $\tilde{\mathcal{S}}_b$ accordingly.

3. The next step is to ensure that for any product with a desired minimum number of subscribers, there are at least five patterns using that product. Based on the set $\hat{\mathcal{B}}$, we denote the set of boxes containing product $p \in \mathcal{P}$ as $\hat{\mathcal{B}}_p$. Based on the set $\hat{\mathcal{B}}$, define $\hat{\mathcal{B}}^s$ as the set of boxes eligible for subscriber $s \in \mathcal{S}$. For any $p \in \mathcal{P}$ with $\phi_p^{\ell} > 0$ such that $\hat{\mathcal{B}}_p \cap \tilde{\mathcal{B}} = \emptyset$, choose $b \in \tilde{\mathcal{B}} \backslash \hat{\mathcal{B}}_p$ such that $|\hat{\mathcal{S}}_b|$ is maximum, and add b and the subscriber $s \in \mathcal{S}$, which generated b during the pattern-generation process, to the sets $\tilde{\mathcal{B}}_s$ and $\tilde{\mathcal{S}}_b$ accordingly.

4. We now want to achieve the goal of having $\hat{m}$ patterns per subscriber. For each $s \in \mathcal{S}$ such that $|\tilde{\mathcal{B}}_s| < \hat{m}$, choose $b \in \hat{\mathcal{B}}^s$ such that $b \notin \tilde{\mathcal{B}}_s$ and $|\tilde{\mathcal{S}}_b|$ is largest. Update the sets $\tilde{\mathcal{B}}_s$ and $\tilde{\mathcal{S}}_b$ accordingly. The concept here is to find new (b,s) pairs that will affect the greatest number of subscribers.

The above steps now define a new restricted master problem based on the sets $\tilde{\mathcal{B}}_s$ and $\tilde{\mathcal{S}}_b$, which will be solved in the **Subscribe** step. By defining $\overline{\mathcal{B}}^s = \tilde{\mathcal{B}}^s$ and $\overline{\mathcal{S}}_b = \tilde{\mathcal{S}}_b$, the restricted master problem described above is well defined.

Subscribe

As we describe above, the beginning of the **Subscribe** step is to solve the MIP form of the restricted master problem created using the **Populate** step with the Objective Function (25). Often, a solution is found at the root node of the Gurobi MIP branch-and-bound procedure. If after limited branching was done, the objective value is smaller than $|\mathcal{S}|$, we then add additional box/subscriber pairs (b,s) to the MIP restricted master problem. The first set of additional patterns is determined by adding all combinations that were present after the **Relax** step—that is, all the patterns and their eligible subscribers that were generated to solve the linear-programming relaxation of the master problem. Using the solution from the first MIP solve, the new MIP with these additional patterns is solved, with the same stopping criteria of achieving the objective target $|\mathcal{S}|$ or doing limited branching. We then try to generate

additional patterns for the MIP restricted master problem by solving the pattern-generation subproblem, using the Objective Function (30). Because dual values are not available, we determine surrogate dual values based on the slack values v_p from the last MIP feasible solution corresponding to Constraint (1r). The Objective Function (30) is then defined by setting

$$\overline{\pi}_p = \frac{1}{v_p + 1} \quad \forall p \in \mathcal{P}. \tag{36}$$

Because the subproblem is a maximization problem, this will emphasize generating patterns that use products that are used most often and are most constrained, again creating combinatorial diversity.

During this phase, we use one other feature to terminate the MIP optimization early. If the solver can prove that the upper bound of the objective function is less than $|\mathcal{S}|$, we immediately terminate the MIP solution process and proceed to generate additional patterns.

In our tests, the restricted master MIP required multiple solves only 28% of the time. That is, the patterns and associated subscribers generated during the **Populate** phase were sufficient for the MIP optimizer to find a solution with an objective function value of $|\mathcal{S}|$, which meant that a solution was found where each subscriber could be assigned a valid box without the need to generate additional patterns.

MIP Parallelization

Because we had access to a machine with more than 24 cores, we tested a Gurobi MIP feature called concurrent optimization for MIP. The random seed used by the Gurobi algorithms can affect MIP performance; therefore, we set up six concurrent parallel solves, each using a different random seed, and four threads to solve each MIP restricted master problem. This increased the chance that one of the Gurobi heuristics would find good solutions at the root node, as opposed to some cores not being used until branching began. As we show in the Computational Results and Impact section, this option provided better performance than using all 24 cores 50% of the time. In contrast, the MIP subproblems were solved by using all cores. However, the majority of those subproblems were solved extremely quickly, thus demonstrating that pattern generation by solving mixed-integer programs was not a factor in overall run times.

Computational Results and Impact

All computational results were performed on a computer with AMD Opteron 6328 processors with a clock speed of 3.2 Ghz and two megabytes of cache. Our computational results are based on using 24 cores of this machine, which had eight processors and four cores per processor. Gurobi 9.0.2 was used as the MIP solver.

During the project, Birchbox provided a few test sets that we used to compare the performance of the RIP algorithm to using Gurobi to solve the original formulation of the problem. Table 2 presents the results using the concurrent optimization feature we describe above. One problem (190.998) did not solve to optimality after 48 hours of computation using 24 threads, while a second problem (192.998) solved to optimality after 17 hours. A third problem (198.999) was easy to solve with the existing formulation and allowed us to test and refine the RIP concept. For the other problems, we did not want to waste the time to use the solver designed for the original formulation, because Birchbox personnel indicated that most of those problems would take more than one day to solve, and their computational resources needed to be used for production operations.

In Tables 2 and 3, we show the cumulative time for each phase using both the **Hard** constraint and the **Soft** constraint techniques described above, highlighting which method (hard or soft) was fastest. The last column includes the times from the existing formulation on three of the problems

Table 3 shows the results if all 24 cores are used to solve the restricted master problem as a MIP (i.e., disabling the Gurobi feature for concurrent optimization).

When comparing the two tables, the concurrent option improved results 50% of the time. In both cases, adding the Hard Constraint (28) gave the best overall

Table 2. Using Concurrent Optimization, Solutions Times in Hours, Minutes, and Seconds (HH:MM:SS) Are Presented

	End of Relax		End of Subscribe		Before starting product		Total time		
Problem	Hard	Soft	Hard	Soft	Hard	Soft	Hard	Soft	Old time
190.998	0:01:23	0:01:26	0:02:37	0:02:32	0:02:40	0:18:49	0:24:50	1:14:24	48:00:00
192.998	0:02:32	0:03:21	0:03:58	0:05:27	0:04:02	0:05:54	0:05:44	0:07:55	17:05:00
194.399.brunel2	0:04:24	0:04:03	0:22:26	0:40:20	0:22:32	1:13:52	0:22:33	1:13:58	
195.995	0:01:49	0:01:49	0:03:54	0:03:30	0:03:59	0:15:52	0:16:19	2:38:44	
198.999	0:01:03	0:00:59	0:03:08	0:02:05	0:03:13	0:04:11	0:03:13	0:04:19	0:03:49
206.600	0:02:40	0:01:32	1:24:45	0:28:02	1:25:01	1:30:44	1:25:03	1:30:53	
207.998	0:01:45	0:01:45	0:19:09	0:02:42	0:19:13	1:10:28	0:44:42	1:11:25	

Notes. Times represent cumulative time to that point in the RIP algorithm. Shaded squares indicate which method was fastest.

Table 3. Without Concurrent Optimization, Solutions Times in Hours, Minutes, and Seconds (HH:MM:SS) Are Presented

	End of Relax		End of Subscribe		Before starting product		Total time		
Problem	Hard	Soft	Hard	Soft	Hard	Soft	Hard	Soft	Old time
190.998	0:01:25	0:01:20	0:03:20	0:03:13	0:03:23	0:04:37	0:19:15	1:38:29	48:00:00
192.998	0:02:37	0:03:08	0:04:22	0:07:55	0:04:25	0:08:22	0:06:12	0:11:16	17:05:00
194.399.brunel2	0:04:01	0:04:04	0:17:28	0:54:05	0:17:34	3:18:14	0:17:35	3:18:19	
195.995	0:01:42	0:01:38	0:03:31	0:03:31	0:03:35	0:04:23	0:11:29	0:17:29	
198.999	0:01:03	0:01:00	0:03:20	0:02:17	0:03:25	0:04:35	0:03:25	0:04:42	0:03:49
206.600	0:02:41	0:01:31	1:26:08	0:16:33	1:26:24	1:06:45	1:26:25	1:06:53	
207.998	0:01:50	0:01:39	0:22:24	0:02:59	0:22:32	0:55:46	0:49:23	0:57:15	

Note. Shaded squares indicate which method was fastest.

performance. As we mention above, the **Soft** option is valuable to Birchbox to determine whether its source data will support assigning every subscriber to a box, independent of the limit on the number of boxes needed.

Impact

For Birchbox, the RIP algorithm has opened up new opportunities for delighting its customers with customized beauty and grooming experiences. With the original formulation, most solves took multiple days, which placed enormous pressure on Birchbox staff to deliver new solutions each month to meet production deadlines. During the COVID-19 pandemic, Birchbox observed an additional benefit. Its suppliers have experienced the same supply chain challenges that manufacturers across the globe have seen. The RIP algorithm allowed Birchbox to quickly replan its boxes, which would have been impossible using the original model. David Bendes, Vice President of Global Business Technology at Birchbox, said, "The increase in speed and flexibility from the new model impacts every piece of our business across multiple teams. Now we can strategically invest that time in everything from building more personal customer experiences to optimizing our production process. With this new model, Birchbox has truly entered a new operating universe" (Lustig and Randall 2020).

The RIP algorithm includes the following innovative ideas for solving a difficult mixed-integer program:

- The use of a MIP solver to generate patterns in a column-generation framework
- A restricted master problem that increases in size in both the number of constraints and number of variables as new patterns are generated
- Modification of dual values in a linear-programming phase to improve performance
- Surrogate dual values in a mixed-integer programming phase to generate new columns for the restricted master problem
- Leveraging a variety of features of a powerful mixed-integer programming solver to improve performance.

We have also applied the RIP algorithm to a set of 24 public-domain nurse-scheduling problems available at http://www.schedulingbenchmarks.org/nrp/. The RIP algorithm found better solutions and/or improved lower bounds on three of the instances as compared with results obtained by other researchers, while also solving to optimality all the instances with known provable optimal solutions in under an hour on a laptop. This substantially improves the recent results reported by Strandmark et al. (2020) and demonstrates that the RIP algorithm is transportable to other applications.

We believe that many applications can benefit from using a powerful mixed-integer programming solver like Gurobi in an iterative solving algorithm such as RIP. Our hope is that more optimization specialists will develop new algorithms in this same spirit. In addition, the power of mixed-integer programming can only be realized by properly formulating the problems at hand. We have observed that sometimes a MIP approach is rejected without an understanding of the interactions of proper formulations with the underlying solver. We conclude by saying: "Formulation matters!"

References

Achterberg T, Wunderling R (2013) Mixed integer programming: Analyzing 12 years of progress. Jünger M, Reinelt G, eds. *Facets of Combinatorial Optimization* (Springer, Berlin), 449–481.

Achterberg T, Bixby RE, Gu Z, Rothberg E, Weninger D (2019) Presolve reductions in mixed integer programming. *INFORMS J. Comput.* 26(4):825–847.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Bixby RE (2012) A brief history of linear and mixed-integer programming computation. Grötschel M, ed. *Optimization Stories* (Deutsche Mathematiker-Vereinigung, Bielefeld, Germany), 107–121.

Bixby RE, Gregory JW, Lustig IJ, Marsten RE, Shanno DF (1992) Very large-scale linear programming: A case study in combining interior point and simplex methods. *Oper. Res.* 40(5):885–897.

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.

Desaulniers G, Desrosiers J, Solomon M, eds. (2005) *Column Generation* (Springer, New York).

Fenyo K, Mitchell A (2019) Sizing up the subscription e-commerce market: 2018 update. Accessed November 24,

2020, https://get.fuelbymckinsey.com/article/sizing-up-the-subscription-e-commerce-market/.

Fischetti M, Lodi A (2011) Heuristics in mixed integer programming. Cochran J, Cox LA Jr, Keskinocak P, Kharoufeh J, Smith JC, eds. *Wiley Encyclopedia of Operations Research and Management Science* (John Wiley & Sons, Hoboken, NJ).

Gilmore P, Gomory R (1961) A linear programming approach to the cutting stock problem. *Oper. Res.* 9(6):849–859.

Gurobi (2020) The fastest solver. Accessed June 30, 2019, http://www.gurobi.com/.

Liberti L (2012) Symmetry in mathematical programming. Lee J, Leyffer S, eds. *Mixed Integer Nonlinear Programming, The IMA Volumes in Mathematics and its Applications*, vol. 154 (Springer, New York).

Lustig I, Randall P (2020) Optimization powers a new operating universe at Birchbox. Accessed January 30, 2020, https://princetonoptimization.com/blog/blog/optimization-powers-new-operating-universe-birchbox.

Marchand H, Martin A, Weismantel R, Wolsey L (2002) Cutting planes in integer and mixed integer programming. *Discrete Appl. Math.* 123(1-3):397–446.

Panko R (2019) Subscription box services statistics 2019: The most subscribed-to brands. Accessed November 24, 2020, https://clutch.co/logistics/resources/subscription-box-service-statistics.

Sadykov R, Vanderbeck F, Pessoa A, Tahiri I, Uchoa E (2019) Primal heuristics for branch and price: The assets of diving methods. *INFORMS J. Comput.* 31(2):251–267.

Strandmark P, Qu Y, Curtois T (2020) First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem. *Comput. Oper. Res.* 120(August):104945.

Vanderbeck F (2005) Implementing mixed integer column generation. Desaulniers G, Desrosiers J, Solomon M, eds. *Column Generation* (Springer, New York), 331–358.

Irvin Lustig is the optimization principal at Princeton Consultants, where he leads the sales and development of Princeton's optimization-based solutions across a wide variety of industries. Prior to joining Princeton, he was at IBM, as a result of IBM's acquisition of ILOG. He is one of the original developers of IBM ILOG CPLEX. He earned a PhD in Operations Research from Stanford University, University, where his thesis was supervised by George Dantzig, the "father of linear programming."

Patricia Randall is a director at Princeton Consultants, where she leads the design, development, and implementation of large-scale, high-impact systems that help businesses optimize their decision making at the strategic, tactical, and operational levels. She earned a PhD in industrial engineering from Clemson University, with a focus on metaheuristics to optimize complex machine scheduling problems.

Robert Randall is a senior optimization specialist at Princeton Consultants, where he develops custom production-optimization models and decision-support systems in areas such as fractional airline scheduling, less-than-truckload (LTL) flow plan optimization, rail car allocation, bulk chemical trailer assignment, and patient scheduling. He earned a PhD in industrial engineering from Clemson University, with a focus on metaheuristic design and development.