Improving Product Placement in Retail with Generalized High-Utility Itemsets

by

Chinmay Bapna, P Krishna Reddy, Anirban Mondal

in

IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)
: 1
-10

sydney, Australia

Report No: IIIT/TR/2020/-1

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2020

Improving Product Placement in Retail with Generalized High-Utility Itemsets

Chinmay Bapna*, Polepalli Krishna Reddy*
International Institute of Information Technology Hyderabad, India
*chinmay.bpn@gmail.com, *pkreddy@iiit.ac.in

Anirban Mondal*
Ashoka University, Haryana, India
*anirban.mondal@ashoka.edu.in

***Abstract*—Product placement in retail has a significant impact on the sales revenue of retailers. Hence, research efforts are being made to improve retailer revenue using high-utility pattern mining based product placement approaches. However, none of these existing approaches has explored generalized high-utility itemset mining for determining product placement in retail. The knowledge of generalized high-utility itemsets extracted from user purchase transactional database in conjunction with a product taxonomy can provide new insights about customer purchase behaviour. This work proposes the generalized utility itemset (GUI) index for retrieving generalized high-utility (revenue) itemsets. We also present a framework, which leverages the GUI index towards retail product placement to improve revenue. Our performance study using real datasets shows the effectiveness of our proposed scheme w.r.t. two existing schemes.**

***Index Terms*—utility mining, generalized association rules, retail management, product taxonomy**

I. Introduction

Product placement on the shelf space in retail stores has a significant impact on the sales revenue of retailers [1]–[3]. The past few decades have witnessed the prevalence and popularity of medium-scale stores and the rising trends of mega-sized retail stores (e.g., Dubai Mall and Walmart Supercenters [4]) with huge retail floor space. Such retail stores generally have a *huge number of slots* across the different aisles of the store space. Thus, it becomes critical to *strategically* place products in the huge number of slots of the retail store in a way that *improves* the sales revenue of the retailer.

The shelf space of a retail store can be segregated into *premium* and *non-premium* slots. Premium slots are those, which are easily visible and physically accessible to the customers e.g. slots which are closer to the eye or shoulder level of the customers. Those slots, which are located very high or very low in the shelves, are examples of non-premium slots. In this work, we focus on product placement only for the premium slots for improving revenue. We assume that each item is physically of the same size i.e., each item consumes equal space on the retail store's shelves. We shall use revenue as an example of a utility measure throughout this paper; hence, we use the terms *revenue* and *utility* interchangeably.

Research efforts are being made to propose pattern mining and utility mining techniques towards enhancing product assortment [5], placing the items strategically [3], [5], designing shop layout [6], indexing high-utility itemsets for facilitating improved product placement [7]–[9] and exploiting product categories to improve retailer revenue [10], [11]. Notably, approaches have also been proposed in [12]–[14] for extracting knowledge of generalized association rules from a given transactional database in conjunction with a taxonomy of items. However, *none* of the existing approaches incorporate knowledge of *high-utility generalized itemsets* in retail.

Given a taxonomy of items and user purchase transactions, it is possible to extract interesting associations among the generalized items at different levels of the taxonomy. For example, typically, there are a relatively few user purchase transactions that include a specific brand of *soap*, *toothbrush* and *toothpaste* in a super market. However, there would be a much larger number of transactions that include the *generalized* itemset of $\{soap, toothbrush, toothpaste\}$.

There is a scope to improve the revenue of the retailer by extending the utility mining paradigm to *generalized itemsets* for improving product placement. For example, consider items a, b, c, d, e, and f. Also, let generalized item of a, b and c be X and d, e, and f be Y. Suppose we extract a generalized pattern with itemsets X and Y with support say ψ. It means that the ψ percentage of the transactions purchases one of the items of X and one of the items of Y. If we follow the existing utility-based placement methods based on the utility patterns at the item level, we have to place itemsets $\{a, d\}$, $\{a, e\}$, $\{a, f\}$, $\{b, d\}$, $\{b, e\}$, $\{b, f\}$, $\{c, d\}$, $\{c, e\}$, $\{c, f\}$ in the given slots of retail store to cover all corresponding transactions. On the other hand, if we place the items based on the knowledge of generalized utility pattern $\{X, Y\}$, it is sufficient to place items $\{a, b, c, d, e, f\}$ at one place. It can be noted that the approach with generalized itemsets provides scope to cover a larger number of user transactions within a relatively lower number of slots in a comprehensive manner.

In this paper, we extend utility mining and generalized itemset mining to propose an improved approach for product placement in retail to improve retailer revenue. Incidentally, generalized itemsets are virtual in the sense that they cannot be directly placed in the slots of the given retail store i.e., we need to identify the specific actual items that pertain to such generalized itemsets. Given a pattern of generalized items, it may not be effective in improving retailer revenue if we place all of the corresponding actual itemsets corresponding to the generalized itemset. This is because many of these actual itemsets would be likely to have low utility (revenue). Referring to our earlier example of the *generalized* itemset of {*soap*, *toothbrush*, *toothpaste*}, there could be some brands of soaps, toothbrushes and toothpastes, which generate higher sales revenue for the retailer e.g., due to higher popularity (frequency of sales) or higher pricing. Here, the issue is to identify the potential items to be placed for a given generalized pattern, while ensuring that different brands of soaps, toothbrushes and toothpastes are placed in the slots of the retail store, thereby providing customers with purchase flexibility based on their brand preferences for improving the revenue of the retailer.

Our proposed framework includes an indexing scheme for extracting and storing high-revenue generalized itemsets and a product placement approach. Given a user purchase transactions database and a product taxonomy, we propose a utility-based framework to extract high-revenue generalized itemsets. For a given level of the taxonomy, we build an index, designated as the **Generalized Utility Itemset (GUI) index**, to identify and store high-revenue generalized itemsets. Furthermore, we propose a product placement framework, which exploits the GUI index, for placing items in the slots of the retail store to maximize retailer revenue.

The key contributions of this work are three-fold:

1) We propose GUI, which is an efficient index for facilitating retrieval of generalized high-revenue itemsets.
2) We propose a framework for leveraging the GUI index towards retail product placement to improve revenue.
3) We conducted a performance evaluation using real datasets to demonstrate the effectiveness of our proposed scheme w.r.t. two existing schemes.

The remainder of this paper is organized as follows. Section II discusses related works and background, while Section III describes the proposed problem framework. Section IV presents the GUI indexing scheme and product placement framework. Section V reports the performance evaluation. Finally, Section VI concludes the paper.

II. RELATED WORK AND BACKGROUND

This section discusses existing works and background.

A. RELATED WORK

Association rule mining approaches [15]–[17] use the *downward closure property* [17] for finding frequent itemsets primarily based on support. However, they do not consider the utility of the items. Notably, the downward closure property does not apply to utility mining. Since a brute-force approach for extracting high-utility itemsets would be prohibitively expensive, utility mining approaches have been proposed.

A representation of high-utility itemsets has been discussed in [18], [19], while the notion of MinHUIs (minimal high-utility itemsets) has been proposed in [20]. MinHUIs are the smallest itemsets that generate high utility [20]. The HUG-Miner and GHUI-Miner algorithms [21] extract high-utility itemsets by mining concise representations in conjunction with pruning.

Observe that in practice, taxonomies (*is-a* hierarchies) typically exist over the items. Hence, association rules can be extracted in a generalized form by using taxonomies, thereby extracting knowledge in a compact form. Generalized association rules depict the semantic relationships between the items and analyze such correlations at higher abstraction levels [12]–[14]. Multi-level association rules apply apriori-like frequent pattern mining strategies for finding frequent itemsets at a given level of abstraction [13]. However, they do not consider the utility of the items.

The case for generalized (cross-category) itemsets in retail has been well-documented in the literature [22]–[24]. The work in [22] found that optimal in-store category adjacencies could increase cross-category sales. The work in [23] indicated the effects of aisle and display placements on significantly improving cross-category brand sales. Furthermore, the work in [24] found that cross-category effects go beyond substitutes and complements in influencing consumer demand.

The works in [5]–[10] discuss product placement and shelf space allocation in the retail domain. A *microeconomic model* for product selection (PROFSET) has been proposed in [25]. It applies association rule mining to mine frequent itemsets and introduces the concept of *gross-margin* for determining optimal product assortments in retail stores. It also integrates the discovery of frequent itemsets with both quantitative and qualitative (domain knowledge) criteria. Moreover, it generalizes this approach by introducing category-based product allocation in [10]. The work in [5] proposed a multi-level association rule mining-based framework for exploring the relationships between products as well as between product categories in retail stores. The works in [7], [9] focused on placing itemsets in the slots of the retail stores when the physical item sizes may vary. The work in [8] addressed the problem of determining the placement of the itemsets in different types of slots with varied premiumness.

Notably, *none* of the existing approaches incorporate knowledge of *high-utility generalized itemsets* in retail. This limits their applicability in building practical systems to place products in retail stores for maximizing the revenue of the retailer.

B. BACKGROUND

The kUI index [26] provides flexibility to the retailer by serving as a guide towards strategic placement of high-utility itemsets in the premium slots of the retail store. kUI is a multi-level index, where each level concerns a given itemset size. At the k^{th} level, the kUI index stores the top-λ high-revenue itemsets of itemset size k. Each level of the kUI index corresponds to a hash bucket. For indexing itemsets of N different sizes, the index has N hash buckets i.e., one hash bucket per itemset size. Hence, given a query for finding the top-λ high-revenue itemsets of a given size k, one can traverse quickly to the k^{th} hash bucket instead of having to traverse all of the hash buckets corresponding to k = {1, 2,..., k-1}.

For each level k in the kUI index, the corresponding hash bucket contains a pointer to a linked list of the top-λ itemsets of size k. The entries of the linked list are of the form (*itemset*, σ, ρ, NR), where σ refers to the frequency of item sales, ρ refers to the item prices and NR refers to the net revenue of the itemset. NR is computed as the product of frequency of sales

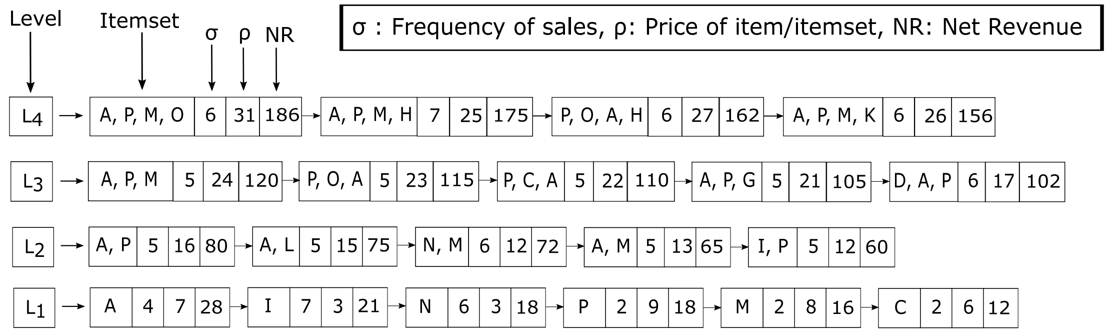


Fig. 1: Illustrative example of the kUI index

(σ) and price (ρ) of items in *itemset*. The entries in the linked list are sorted in descending order of NR for facilitating quick retrieval of high-revenue itemsets of a given size. Figure 1 depicts an illustrative example of the kUI index. Observe how the itemsets (e.g., {I}, {O}) of size 1 correspond to level 1 of the index, the itemsets of size 2 (e.g., {A, L}, {N, M}) correspond to level 2 of the index and so on. Notice how the itemsets are sorted in descending order of NR.

Observe that in practice, the number N of products in large retail stores can run into tens of thousands. The kUI index generates the top-λ high revenue items from a considerably large number of items i.e., $\lambda << N$. Consequently, it considers *only* these top-λ items to subsequently build higher levels of the index by generating itemsets from among these items. Hence, the items, which are outside of the top-λ high revenue items (as well as their corresponding associations), are essentially ignored and therefore not *covered* by the kUI index, thereby possibly resulting in missed opportunities for improving the revenue of the retailer. On the other hand, we cannot keep increasing the value of λ to cover a significant percentage of the items because the computational overheads of finding all of the associations (e.g., 2-itemsets, 3-itemsets etc.) would become prohibitively expensive beyond a certain point.

III. Proposed Framework of the Problem

Consider a set D of user purchase transactions on a finite set Υ of m items, where each transaction comprises a set of items from set Υ. Each item $i \epsilon \Upsilon$ is associated with a price ρ_i and a frequency of sales σ_i. Let $\mathcal{T}$ be a taxonomy (i.e., a *is-a* hierarchy), which aggregates items in Υ into higher-level concepts that represent the semantic *is-a* relationship between the related items. These higher-level concepts will henceforth be designated as *generalized items*.

Generalized items can be further aggregated to other generalized items at higher abstraction levels. We assume the taxonomy to be a well-balanced tree with the items in Υ as the leaf nodes. We assign the leaf-level nodes of a taxonomy as **taxonomy level** zero, the nodes on the level immediately above it as *taxonomy level* one, the next higher level of nodes as *taxonomy level* two and so on. Observe that in a balanced taxonomy, the nodes on the same level of the taxonomy tree would have the same *taxonomy level* values. Our focus is on finding combinations of generalized items lying on the same level in the taxonomy, since they precisely constitute information at specific abstraction levels. We shall now discuss some terminology for a better understanding of the context.

Definition 1 (Net Revenue of an item): We define the *net revenue* NR_k of a specific item k as the product of its price ρ_k and its frequency σ_k of sales i.e., $NR_k = (\rho_k * \sigma_k)$.

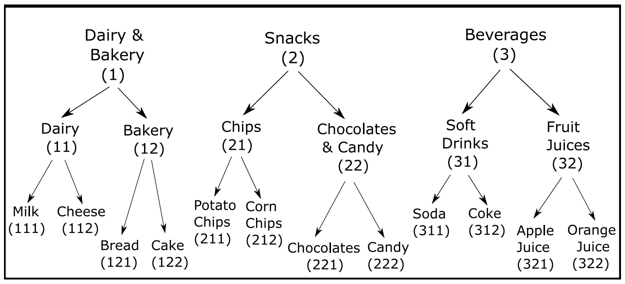
Now we define the notion of net revenue of a generalized item after defining the notion of cover set.

Definition 2 (Cover Set): Let G be the set of generalized items in $\mathcal{T}$. Given a generalized item $g \epsilon$ G, **cover set (CSet)** of g, where CSet(g) $\subseteq \Upsilon$, is the set of leaf-level items covered by g. Thus, the *cover set* maps generalized items in the taxonomy to their corresponding leaf-level items.

Definition 3 (Net Revenue of a generalized item): Given a taxonomy $\mathcal{T}$ and a generalized item $g \epsilon$ G, *net revenue* NR_g of g is the product of its frequency of sales and the mean price of items in CSet(g) i.e., $NR_g = ceil(\frac{\sum_{j \epsilon CSet(g)} \rho_j}{|CSet|}) * \sigma_g$.

Observe that the value of NR_g computed above is approximate because we are using the average of the prices of the items in the corresponding cover set of a given generalized item as opposed to considering the product of the prices and frequencies of sales of each individual item. This approximation in the computation of NR_g is justifiable because in practice, item prices in the same taxonomy category do not typically vary to any significant extent in a given retail store. In the absence of this approximation, we would have to maintain individual item prices and item frequencies of sales for each level of our proposed GUI index, thereby resulting in increased storage and computational overheads.

Figure 2a depicts a sample taxonomy *Retail* of items. The items in the taxonomy have been encoded using a simple encoding scheme described in Table I. Figure 2b presents a sample transactional database of encoded items of the taxonomy. We construct generalized transactions corresponding to a specific taxonomy level by replacing each item in the original transaction by its respective ancestor at that taxonomy level. Some examples of generalized transactional databases of *taxonomy levels* one and two are depicted in Figures 2b(ii) and 2b(iii) respectively, which have been generalized from the transactions in Figure 2b(i).



(a) Sample Taxonomy *Retail*

TID	Transaction	σ	ρ	NR
1	111,322,112	3	6	18
2	321,212	2	5	10
3	212,112,321,311	1	9	9
4	121,111,322	2	4	8
5	211,312	2	5	10

(i) Transactions of taxonomy level-0

TID	Transaction	σ	ρ	NR
1	11:2,32:1	3	6	18
2	32:1,21:1	3	5	15
3	21:1,11:1,32:1,31:1	1	9	9
4	12:1,11:1,32:1	2	7	14
5	21:1,31:1	6	5	30

(ii) Transactions of taxonomy level-1

TID	Transaction	σ	ρ	NR
1	1:2,3:1	5	6	30
2	3:1,2:1	9	5	45
3	2:1,1:1,3:2	1	9	9
4	1:2,3:1	5	6	30
5	2:1,3:1	9	5	45

(iii) Transactions of taxonomy level-2

(b) Transactional Database

Fig. 2: Taxonomy and corresponding Transactional Database

TABLE I: Encoding of nodes in Product Taxonomy.

Item	Code	σ	ρ	NR
Dairy & Bakery	1	21	2	42
Snacks	2	12	3	36
Beverages	3	27	2	54
Dairy	11	13	2	26
Bakery	12	8	3	24
Chips	21	8	3	24
Chocolates & Candy	22	4	3	12
Soft Drinks	31	12	2	24
Fruit Juices	32	15	2	30
Milk	111	9	1	9

Item	Code	σ	ρ	NR
Cheese	112	4	3	12
Bread	121	6	1	6
Cake	122	2	4	8
Potato Chips	211	2	3	6
Corn Chips	212	6	3	18
Chocolates	221	2	5	10
Candy	222	2	1	2
Soda	311	3	1	3
Coke	312	9	2	18
Apple Juice	321	7	2	14
Orange Juice	322	8	2	16

For instance, consider the *taxonomy level-1* transaction X = {11:2, 32:1} i.e., {*dairy*:2, *fruit juices*:1} in Figure 2b(ii). Observe that the cover sets of the generalized items of *taxonomy level* one, *dairy* and *fruit juices* are CSet(*dairy*) = {*milk*, *cheese*} and CSet(*fruit juices*) = {*apple juice*, *orange juice*} respectively. Notice that price of the itemset $\rho_X = ceil(\frac{\rho_{milk} + \rho_{cheese}}{2}) * 2 + ceil(\frac{\rho_{apple\ juice} + \rho_{orange\ juice}}{2})$ is equal to 6. Its frequency of sales σ_X is equal to 3, hence its net revenue $NR_X(\rho_X * \sigma_X)$ equals 18. Similarly, for the *taxonomy level-2* transaction from Figure 2b(iii), Z={2:1, 1:1, 3:2} i.e., {*snacks*:1, *dairy & bakery*:1, *beverages*:2}, price of the itemset ρ_Z is equal to 9, its frequency of sales σ_Z is equal to 1, thus its net revenue $NR_Z(\rho_Z * \sigma_Z)$ equals 9.

Problem statement: This paper addresses the problem of product placement in retail with generalized high-revenue itemsets for improving the revenue of the retailer.

IV. Proposed Scheme

This section presents our proposed scheme.

A. Overview

Consider the existing kUI-based utility mining approach [26] for itemset placement in retail stores. Given that the total number of items in a large-scale retail store can be large, the approach considers only the top-λ high revenue items and creates 2-itemsets, 3-itemsets and so on up to n-itemsets by associating them. However, the items, which are outside of the top-λ high revenue items (as well as their corresponding associations), are essentially ignored and therefore not *covered* by the placement algorithm, thereby possibly resulting in missed opportunities for improving the revenue of the retailer. On the other hand, we cannot keep increasing the value of λ to cover a significant percentage of the items because the computational overheads of finding all of the associations (e.g., 2-itemsets, 3-itemsets etc.) would become prohibitively expensive beyond a certain point. Here, by considering generalized itemsets, we can essentially *cover* a much larger percentage of the items without incurring significant computational overheads.

The following issues are to be resolved to develop item placement framework based on generalized high-utility itemsets. First, given a transactional database and generalized items at the given taxonomy level, the issue is to build a k-level generalized utility itemset (GUI) index and identify the potential set of items for each generalized itemset. The second issue is to develop a placement algorithm, given GUI index of a given taxonomy level and the number of premium slots.

Given a set of items Υ, transactional database over Υ and taxonomy $\mathcal{T}$ over Υ, the GUI index for each taxonomy level tl, GUI(tl), is built from high-revenue generalized itemsets based on the existing generalized itemset extraction algorithm [12], [13]. GUI(tl) is a multi-level index, where the j^{th} level of GUI(tl) corresponds to j-sized generalized itemsets. The index is built in a level-wise manner starting from the lowest level, which corresponds to generalized items of size 1. Then

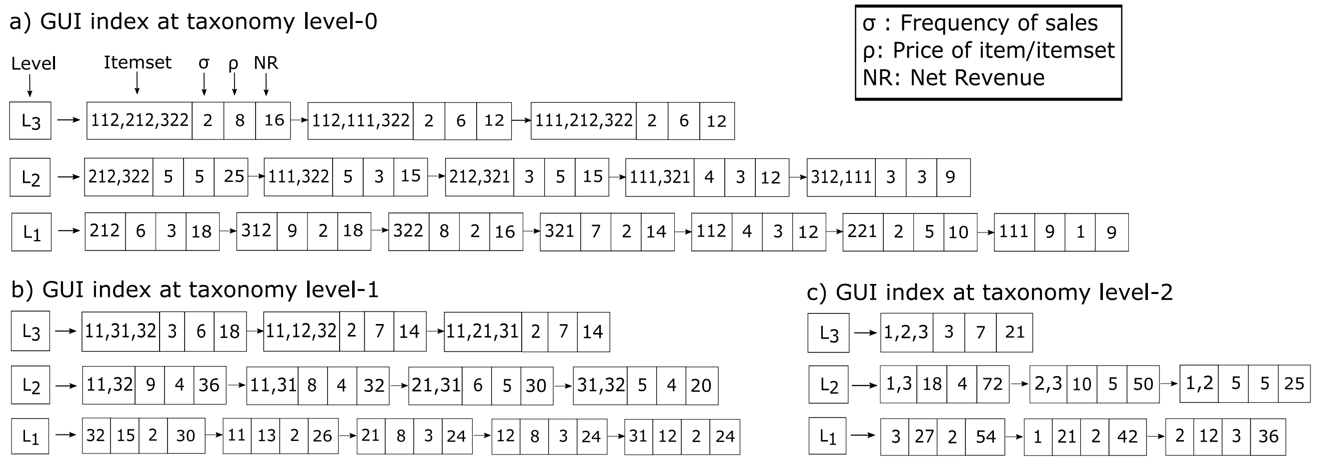


Fig. 3: Illustrative Example of the GUI Index

the next higher levels of the index are built progressively one-by-one by considering only the top-λ high-revenue itemsets at the lower levels.

Normally, given a generalized item g in GUI(tl), the cover set CSet(g) contains several items. If we place all leaf-level items of g, it will not be effective as the net revenue of some of the items may be low. Hence, the issue is to identify the potential items from CSet(g) with high net revenue for placement. For each CSet(g), we maintain a revenue contribution list, RCL(g), which stores potential items based on their mean net revenue.

Given the number of premium slots, our proposed placement framework identifies high-revenue generalized itemsets from each level of GUI(tl) for allocation of premium slots in a round-robin manner. It progressively allocates generalized itemsets of different sizes to premium slots until all such slots are exhausted. Subsequently, in the slots allocated to each generalized item g, we iteratively place high revenue items from RCL(g) such that, each item in RCL(g) gets slots proportionally based on their net revenue values.

B. GUI index and RCL

1) Building the GUI index: For a given taxonomy and taxonomy level tl, the GUI index extracts high-revenue generalized itemsets of varying sizes from its corresponding generalized transactional database of a given taxonomy level tl. GUI index constitutes a multi-level index, where each level concerns a given itemset size for generalized items of given taxonomy level. Thus, for each level k of the GUI index, the corresponding hash bucket contains a pointer to a linked list of the top-λ itemsets of size k. The entries of the linked list are of the form (*itemset*, σ, ρ, NR), where *itemset* refers to a generalized itemset, σ refers to its frequency of sales, ρ is its price and NR is the net revenue of the concerned itemset. Moreover, the entries are arranged in descending order of NR.

The GUI index for a given taxonomy level tl is built from a generalized transactional database, which is built by replacing each item in the given transactions by their corresponding generalized ancestor items in the taxonomy on tl. The index is built in a level-wise manner starting from the lowest level, which corresponds to itemsets of size 1. First, for level 1 of the GUI index, we identify top-λ generalized itemsets of size 1 based on their net revenue. Next, we build the level 2 of the index from top-λ high-revenue itemsets of size 2 selected from all the combinations of the items of level 1. Similarly, the higher levels of GUI index are computed from joining the corresponding lower level items (itemsets). This process is continued till the maximum level of the index has been populated with itemsets of their respective sizes at that level. Furthermore, for the purpose of appropriately representing generalized items/categories of the GUI index on the shelf space of a retail store, we will shortly discuss the concept of revenue contribution lists.

Figure 3 depicts an illustrative example of the GUI index, built on the items described in the taxonomy shown in Figure 2. Recall that Figure 2b portrays the transactional databases based on items of the taxonomy. GUI index of taxonomy level-2 is built from the transactions of taxonomy level-2 such as those in Figure 2b(iii). Similarly, GUI index of taxonomy level-1 is built from transactions of taxonomy level-1 and so on. Observe that the itemsets of size one for each taxonomy level of the GUI index (e.g., {212}, {32}, {3}) correspond to level one of the corresponding GUI index, the itemsets of size two (e.g., {212, 322}, {11, 32}, {1, 3}) correspond to level two of the index and so on. Notice how for each level, the entries show a linked list of itemsets with their prices, frequency of sales and net revenue values. Observe that the itemsets are sorted in descending order of NR.

2) Revenue Contribution List: Given a taxonomy $\mathcal{T}$, the **selection threshold (ST_{NR})** of a generalized item g is computed as $ST_{NR}(g) = (\mu_{NR}(g) + (\beta/100) * \mu_{NR}(g)))$, where $\mu_{NR}(g)$ is the *mean value* of NR across all the items in CSet($g, \mathcal{T}$). Here, β is a parameter which controls the threshold ST_{NR}, and its value lies between 0 and 100. The parameter β is application-dependent and it acts as a *lever* to limit the number of items satisfying the selection threshold criterion.

The **revenue contribution list (RCL(g))** refers to a list of tuples, with each tuple comprising of an item j in CSet($g, \mathcal{T}$) with its **revenue coefficient RC_j** such that its net revenue equals or exceeds the selection threshold ($NR_j \geq ST_{NR}(g)$). The tuple is of the form $<j, RC_j>$. The *revenue coefficient* of an item j in CSet($g, \mathcal{T}$) is computed as the ratio of NR_j with the sum of NR across all the items selected in RCL(g). Note that RCL(g) is sorted in descending order of RC.

Consider a category CT that consists of eight items in its cover set. The items and their net revenue values are given as follows: (A, 12), (B, 20), (C, 5), (D, 13), (E, 2), (F, 6), (G, 10) and (H, 4). The selection threshold $ST_{NR}(CT) = (72/8) * (1 + 30/100)$ (assuming $\beta = 30$) is equal to 11.7. Thus, the revenue contribution list of CT i.e., RCL(CT) is $\{<B, 20/45>, <D, 13/45>, <A, 12/45>\}$. Let us assume that CT is assigned 6 premium slots. First, we place the topmost item in RCL(CT), 3 slots are allocated to B (*ceil*(6*(20/45))). Since, 3 slots are still left (6-3), for the next item D, (*ceil*(6*(13/45))) 2 slots are allocated. Finally, item A will be allocated the remaining 1 slot. Hence, slots allocated to the items B, D and A are 3, 2 and 1 respectively. Observe how the number of slots allocated to a given item is proportional to the relative revenue contribution of that item. Note that for the cases when two or more items have the same RC values, we resolve ties arbitrarily through random allocation of slots.

C. Revenue-based Generalized Itemset Placement (RGIP)

We propose a framework, designated as **Revenue-based Generalized Itemset Placement (RGIP)**, which identifies high-revenue generalized itemsets from the GUI index for determining product placement in the premium slots. Our proposed scheme works as follows. Premium slots are allocated to high-revenue generalized itemsets, in a round-robin manner, based on itemset sizes. This is subsequently followed by mapping slots allocated to generalized items with their corresponding high revenue leaf-level items in the product taxonomy.

Algorithm 1 depicts our proposed RGIP scheme. It takes as input, the number of premium slots to be allocated, N; taxonomy level, t; GUI index of taxonomy level t, GU; number of levels in GU, $maxL$; number of itemsets on each level of GU, λ; list of pointers to linked lists on each level of GU, $pList$; and the list of revenue contribution lists, LR, which for every node r, in the product taxonomy, stores RCL(r), which consists of a sorted list of tuples having items with their corresponding RC values. The algorithm works as follows. In Lines 1-2, GU and LR have been declared. In Lines 3-7, we perform the necessary initializations.

In Lines 8-21, the algorithm allocates N premium slots using the GUI index. In Lines 22-35, the algorithm places items in allocated slots. We pick generalized itemsets from GU and store the number of slots allocated to generalized items in an array, $temp$. The variable s_left represents the remaining slots to be allocated over the course of this procedure. It is initialized as N. In every iteration, we select the itemset K, at

Algorithm 1: Revenue-based Generalized Itemset Placement (RGIP)

Inputs: (a) N: Number of premium slots (b) t: Taxonomy Level (c) GU: GUI Index on taxonomy level t (d) $maxL$: Number of levels in GU (e) λ: Number of itemsets on each level of GU (f) $pList$: List of pointers to linked lists on each level of GU (g) LR: List of revenue contribution lists

Output: S: Placement of itemsets in N premium slots

Variables: (a) S: Slot Array (b) s_left: Total slots left (c) $clev$: Current level (d) $temp$: Array of items (e) p: Pointer to level of GU (f) K: A generalized itemset (g) $alloc$: Number of slots allocated to a generalized item (h) $assign$: Number of slots assigned to a leaf-level item

```
1  Scan GUI index as a 2D array $GU[L, num[L]]$
2  Scan $LR$ as a 3D array $LR[L_1, L_2, num(L_1), num(L_2)]$
3  $S \leftarrow []$
4  $s\_left \leftarrow N$
5  $clev \leftarrow 2$
6  foreach item $j$ in $LR$ do
7      $temp[j] \leftarrow 0$
8  while $s\_left \neq 0$ do
9      if $clev > maxL$ or $s\_left - clev < 2$ then
10         $clev \leftarrow 2$
11         if $s\_left \leq maxL$ then
12             $clev \leftarrow s\_left$
13     $p \leftarrow pList[clev]$
14     if $p == NULL$ then
15         break
16     $K \leftarrow GU[clev][p]$
17     $pList[clev] \leftarrow pList[clev] \rightarrow next$
18     foreach item $j$ in $K$ do
19         $temp[j] \leftarrow temp[j] + 1$
20         $s\_left \leftarrow s\_left - 1$
21     $clev \leftarrow clev + 1$
22 if $t > 0$ then
23     foreach item $j$ in $temp$ do
24         $alloc \leftarrow temp[j]$
25         foreach leaf $l$ in $LR[j]$ do
26             $assign \leftarrow ceil(LR[j][l] * temp[j])$
27             if $alloc \geq assign$ then
28                 $S[l] \leftarrow assign$
29                 $alloc \leftarrow alloc - assign$
30             else
31                 $S[l] \leftarrow alloc$
32                 break
33 else
34     $S \leftarrow temp$
35 return $S$
```

the current pointer of the linked list in GU and level (itemset-size) $clev$. For every generalized item j in K, we increment the count of slots allocated to j in $temp$ by 1. After every allocation we update the remaining slots.

We extract the top-revenue 2-sized generalized itemset from GU, update $temp$ and increment $clev$ by 1. Then we extract the top-revenue 3-sized itemset, allocate slots to it, and so on. Thus, we progressively allocate slots to itemsets by incrementing $clev$. Upon reaching the topmost level of GU, we do a round-robin and circle back to the next top-revenue 2-sized itemset, and repeat this process until all of the slots have been allocated.

Subsequently, we progressively iterate on all the nodes in $temp$ and begin to place items in the slot array, S. For the case when $t > 0$, for every node j in $temp$, we select items from the revenue contribution list of j, $LR[j]$, and place them in slots allocated to j, in proportion of their revenue coefficient RC, till all such slots are exhausted. For the other case when t is equal to 0 (leaf-level of taxonomy), $temp$ itself represents the slot placement.

V. PERFORMANCE EVALUATION

This section reports the performance evaluation, which was done on a 64-bit i3 core processor running Ubuntu 16.04 with 4GB memory. We have implemented our proposed RGIP scheme as well as the reference schemes in Python.

We use two real datasets, namely the *instacart retail dataset* (designated as **DSET1**) [27] and the *classical R "groceries" market basket analysis dataset* (designated as **DSET2**) [28], as summarized in Table II. DSET1 has a pre-defined product taxonomy with three levels, the number of items at levels 0, 1 and 2 being 49688, 134 and 21 respectively. For DSET2, we used the product taxonomy (concept hierarchy) available from Tesco to generate a three-level taxonomy. The number of items at levels 0, 1 and 2 were 169, 57 and 10 respectively. Notably, the taxonomy of each dataset can be represented in the form of well-balanced three-level trees. Since neither of these datasets provides utility values, we generated the prices of the items for each dataset as follows. Given an item, we generated its price in the range of [0.01-1.0] as follows. We considered five price ranges i.e., [0.01-0.20], [0.21-0.40], [0.41-0.60], [0.61-0.80] and [0.81-1.0]. For assigning prices to items, we *randomly* picked one of these price ranges and generated a random number within that price range.

We split each dataset into a *training set* and a *test set* with 70% and 30% of the transactions respectively. For all schemes, we place the itemsets based on the experiments run on the training set and evaluate their performance on the test set.

As reference, we adapted the Generalized PROFSET model [10], [25], which we have discussed in Section II-A. Consider a transactional database with utility information and a user-defined minimum support min_sup as input. Generalized PROFSET outputs a set of frequent itemsets with high gross-margin (utility) having support no less than that of min_sup. By scanning the training set, the algorithm generates frequent itemsets of different sizes. It then estimates the gross-margin of each frequent itemset using the algorithm in [10]. This is followed by a user-defined category-based allocation of items from the frequent itemsets such that the sum of gross margin of all the selected frequent itemsets is maximized.

We adapted the Generalized PROFSET model as follows. First, using the procedure defined above, we generate all the frequent itemsets having support more than min_sup. We compute the gross margin for each frequent itemset and select

TABLE II: Statistical Information about Datasets.

Dataset	No. of Items	No. of Transactions
DSET1	49,688	3,214,874
DSET2	169	9,835

itemsets with gross margin greater than a minimum utility threshold min_util across different itemset sizes (k). Since the number of frequent itemsets to be analyzed is typically very large, we set a threshold on the gross-margin to prune the itemsets having low gross-margin. This is followed by a random allocation of the total premium slots to different product categories. From the itemsets extracted from the training set, we first place a 2-sized itemset onto the premium slots, followed by placing a 3-sized itemset and thus, we do a round-robin selection of itemsets based on itemset-size for placement, until all of the available category slots have been exhausted. We shall henceforth refer to this scheme as **GPF**.

We also compare our proposed scheme with the kUI index [26] (see Section II-B). We apply the **RGIP** scheme discussed in Algorithm 1 to place the high-revenue itemsets in the premium slots from kUI index and GUI index, and refer to the schemes as **kUIP** and **GUIP** respectively. Recall that kUI index is a multi-level index where the j^{th} level of the index corresponds to j-sized high-revenue itemsets. For kUIP, we extract the top-revenue 2-sized itemset from kUI index and assign 2 slots to it. Then we extract the top-revenue 3-sized itemset and assign 3 slots to it and thus, we do a round-robin selection of itemsets based on itemset-size for placement, until all of the available slots have been allocated.

For the kUIP and GUIP schemes, the itemsets are extracted from the training set and the respective index (i.e., kUI/GUI) and RCL for GUIP is built offline. For the GPF scheme, we performed an offline extraction of all of the itemsets from the training set.

Our performance metrics include **execution time (ET)** and **total revenue (TR)**. For the kUIP and GUIP schemes, ET is the sum of the time required for the identification of itemsets from the respective index and the time required for placement in the slots, until all of the slots have been exhausted. In case of the GPF scheme, ET is the sum of the time required for the identification of itemsets from the extracted itemsets of the training set and the time required for placement in the slots, until all of the slots have been exhausted. Moreover, TR is the total revenue earned by the retailer based on the results of the test set. To compute TR, we iterate through each transaction t in the test set and add to TR only the prices of those items (in t), which have already been placed in the premium slots during the training phase.

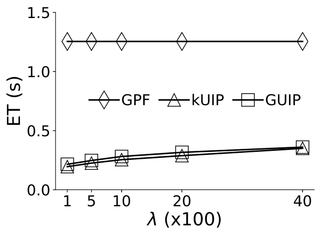
Table III summarizes the parameters of our performance evaluation. We specified the values of these parameters based on our understanding of the application environment through the results of our preliminary experiments.

A. *Effect of variations in* $\boldsymbol{\lambda}$

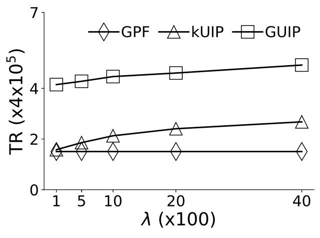
Figure 4 depicts the effect of variations in λ for the datasets DSET1 and DSET2 respectively. Being independent of λ, ET and TR for the GPF scheme remain constant with varying

TABLE III: Parameters of Performance Evaluation

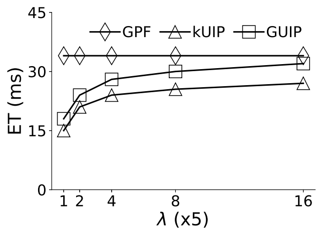
Parameters	DSET1		DSET2	
	Default	Variations	Default	Variations
Top high-utility items (λ)	1000	100, 500, 2000, 4000	20	5, 10, 40, 80
Number of Premium Slots (N)	1000	100, 500, 2000, 4000	20	5, 10, 40, 80
Taxonomy Level (TL)	1	0, 2	1	0, 2
Selection threshold ($\beta(\%)$)	30	10, 20, 40, 50	30	10, 20, 40, 50
Itemset Size/Levels (k)	10	-	8	-
Revenue threshold ($\alpha(\%)$)	30	-	30	-



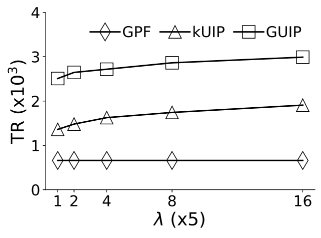
(a) ET (DSET1)



(b) TR (DSET1)

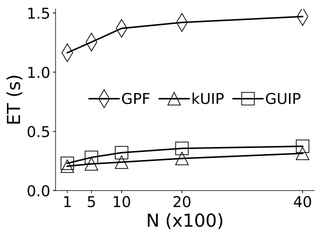


(c) ET (DSET2)

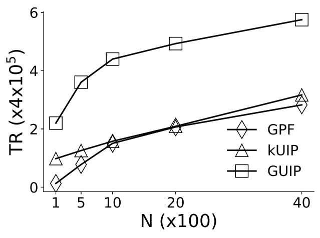


(d) TR (DSET2)

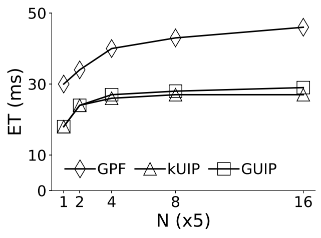
Fig. 4: Effect of variations in λ (DSET1 & DSET2)



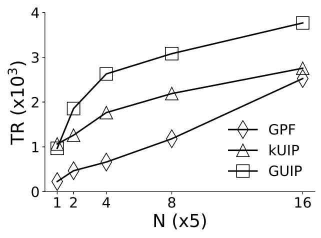
(a) ET (DSET1)



(b) TR (DSET1)

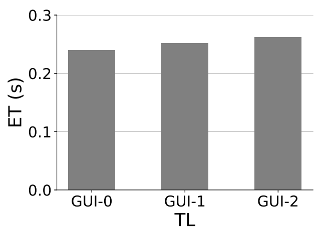


(c) ET (DSET2)

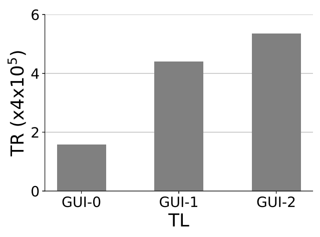


(d) TR (DSET2)

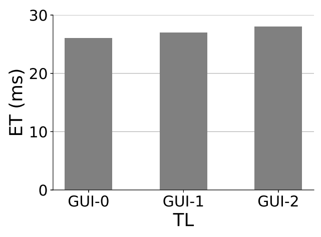
Fig. 5: Effect of variations in total number of premium slots (DSET1 & DSET2)



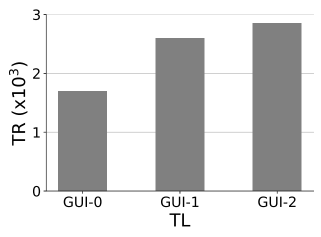
(a) ET (DSET1)



(b) TR (DSET1)

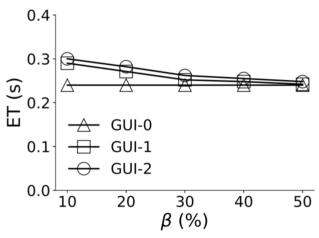


(c) ET (DSET2)

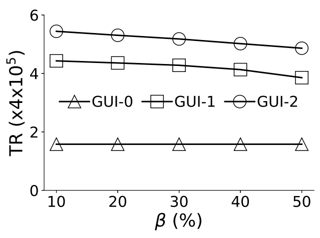


(d) TR (DSET2)

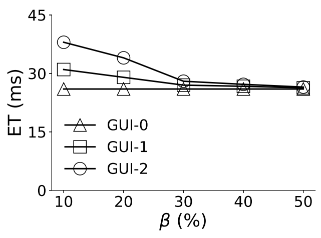
Fig. 6: Effect of variations in taxonomy levels (DSET1 & DSET2)



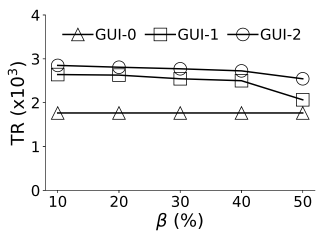
(a) ET (DSET1)



(b) TR (DSET1)



(c) ET (DSET2)



(d) TR (DSET2)

Fig. 7: Effect of variations in selection threshold with varying taxonomy levels (DSET1 & DSET2)

TABLE IV: Summary of Notations of the Approaches

Notation	Description	Citation
GPF	Generalized PROFSET	[10]
kUIP	k-Utility Itemset Placement	[26]
GUIP	Generalized Utility Itemset Placement	Algorithm 1

λ. GPF scheme examines all the patterns that exceed a pre-specified minimum support and minimum gross-margin. It incurs a high execution time for search and retrieval of items from a large number of itemsets for placement in category slots. The kUIP and GUIP schemes outperform GPF in terms of ET since they only examine the top-λ high-revenue itemsets of different sizes (by exploiting the kUI index and GUI index respectively). Moreover, as λ increases, the number of patterns to be examined increases for both kUIP and GUIP, which contributes to the increase in ET. Furthermore, GUIP exhibits slightly higher values of ET than kUIP since it involves an additional step of mapping its selected generalized items to their corresponding leaf-level items in the taxonomy.

The results in Figure 4b indicate that the GUIP scheme provides higher TR than both the GPF and kUIP schemes. GPF is outperformed by GUIP and marginally falls behind kUIP in terms of TR. GPF scheme imposes a restriction on both the support and gross-margin of itemsets and it ignores itemsets with high support & low gross-margin or low support & high gross-margin. Moreover, a randomized selection of itemsets for placement in an arbitrary number of category slots further inhibits the revenue of the model. On each level, kUI index stores only the **top-λ** high-revenue itemsets. Hence, the items, which are outside of the top-λ high revenue items, are essentially ignored and therefore not *covered* by the kUI index, thereby resulting in missed opportunities which is indicated by its TR curve. However, with increasing values of λ, kUIP scheme considers more high-revenue itemsets and thus shows an increase in TR. GUIP considers top-λ generalized itemsets on each level. Since generalized items on higher levels of the taxonomy are already small in number, lower values of λ do not hinder the performance of GUIP. Furthermore, it places high revenue leaf-level items for each selected generalized item, which provides GUIP a larger outlook over the items enabling discovery of distinct itemsets, which are unbeknownst to approaches that restrict their focus only on the leaf-level items of the taxonomy. The results in Figures 4c and 4d follow similar trends as those of Figures 4a and 4b; the difference in the actual values of the performance metrics arises due to the respective dataset sizes.

B. Effect of variations in N

Figure 5 depicts the results for the datasets DSET1 and DSET2 respectively when we vary the number of premium slots (N). GUIP and kUIP outperform GPF in terms of ET due to the reasons explained earlier for Figure 4a. A detailed investigation of the experimental logs revealed that ET increased for all of the schemes. This is because as the value of N increases, more slots need to be filled, thereby necessitating a slightly higher number of patterns to be examined. However, this increase in ET is only slight for kUIP and GUIP because of their efficient indexing mechanism, which maintains the top-λ high-revenue itemsets. On the other hand, the predominant cost for GPF arises from examining a large number of itemsets for extraction of itemsets of varied sizes and placing them in their corresponding category slots.

The results in Figure 5b indicate that the total revenue TR increases for all the schemes with increase in N. This occurs because as the value of N increases, more slots need to be filled up. Hence, more items would be used to fill up an increased number of slots, thereby resulting in more revenue. GUIP provides higher TR than that of kUIP and GPF for the same reasons as explained for the results in 4b. Observe that the results in Figures 5c and 5d follow similar trends as those of Figures 5a and 5b; the difference in the actual values of the performance metrics arises due to the respective dataset sizes.

C. Effect of variations in TL

Figure 6 depicts the performance of GUIP scheme for the datasets DSET1 and DSET2 respectively when we vary the number of taxonomy levels (TL) in the GUI index. The results in Figure 6a indicate that ET increases with increase in TL. Here, GUI-2 refers to GUIP scheme exploiting GUI index of taxonomy level-2, GUI-1 corresponds to GUIP with GUI index of taxonomy level-1, and so on.

As we go up the levels in the taxonomy tree $\mathcal{T}$, the total number of nodes on each level decrease by a large extent and each such node g, maps to a much larger number of items in its cover set i.e., CSet(g). As discussed in Section IV-C, after allocating premium slots to generalized items, GUIP maps each generalized item g, to leaf-level items in its revenue contribution list RCL(g), for placement in its allocated slots. With increasing TL, size of CSet(g) increases, which further results in increase in execution time for decoding generalized items to their respective leaf-level counterparts in RCL(g). Thus, GUI-0 outperforms GUI-1, which outperforms GUI-2 in terms of ET albeit slightly.

The results in Figure 6b indicate that GUI-2 outperforms both GUI-0 and GUI-1 in terms of TR. This is largely because generalizing a large number of items to a concise set of categories enables us to perform a more efficient analysis for selecting high-revenue itemsets in contrast to analyzing only a small fraction of the total items. Hence, for higher values of TL, more items with good revenue garnering potential are selected, which leads to increased total revenue. Observe that the results in Figures 6c and 6d follow similar trends as those of Figures 6a and 6b; the difference in the actual values of the performance metrics arises due to the respective dataset sizes.

D. Effect of variations in β

Figure 7 depicts the performance of GUIP scheme for the datasets DSET1 and DSET2 respectively with respect to the different abstraction levels in the taxonomy, when we vary the parameter β which controls the selection threshold. As discussed in Section IV-B2 for a generalized item g, the selection threshold is computed as $(\mu_{NR}(g) + (\beta/100) * \mu_{NR}(g)))$, where $\mu_{NR}(g)$ is the mean value of net revenue across all the items in the cover set of g. Since the selection threshold limits the number of leaf-level items to be placed in slots with respect to every selected generalized item in GUIP, for GUI-0 ($TL = 0$), which only considers items on the leaf-level, values of ET and TR are independent of β.

The results in Figure 7a indicate that ET decreases with increase in the value of β. Moreover, GUI-0 outperforms both GUI-1 and GUI-2 in terms of ET for the same reasons as explained for the results in Figure 6a. Increase in the value of β implies decrease in the number of items selected from the cover set for every generalized item in the taxonomy, and hence leads to decrease in ET.

The results in Figure 7b indicate that GUI-2 outperforms both GUI-0 and GUI-1 in terms of TR for the same reasons as explained for the results in Figure 6b. Increase in the value of β implies decrease in the number of items selected from the cover set for every generalized item in the taxonomy and with lesser number of items placed in premium slots, the total revenue also decreases. Observe that the results in Figures 7c and 7d follow similar trends as those of Figures 7a and 7b; the difference in the actual values of the performance metrics arises due to the respective dataset sizes.

VI. Conclusion

Appropriate placement of products in a given retail store is critical to improving the revenue of the retailer. In this regard, we have proposed a generalized high-utility (revenue) mining approach for retail product placement to improve retailer revenue. Our performance evaluation with two real datasets demonstrates the effectiveness of the proposed scheme in terms of total revenue and execution time w.r.t. two existing schemes. In the near future, we plan to examine the memory usage of our proposed scheme w.r.t. existing schemes. Furthermore, we plan to investigate the cost-effective integration of our proposed scheme into existing retail business systems.

References

[1] M.-H. Yang and W.-C. Chen, “A study on shelf space allocation and management,” *International Journal of Production Economics*, vol. 60, pp. 309–317, 1999.

[2] M.-H. Yang, “An efficient algorithm to allocate shelf space,” *European Journal of Operational Research*, vol. 131, no. 1, pp. 107–118, 2001.

[3] Y.-L. Chen, J.-M. Chen, and C.-W. Tung, “A data mining approach for retail knowledge discovery with consideration of the effect of shelf-space adjacency on sales,” *Decision Support Systems*, vol. 42, no. 3, pp. 1503–1520, 2006.

[4] B. Farfan, *Largest retail stores*, 2019. [Online]. Available: https://www.thebalancesmb.com/largest-retail-stores-2892923

[5] M.-C. Chen and C.-P. Lin, “A data mining approach to product assortment and shelf space allocation,” *Expert Systems with Applications*, vol. 32, no. 4, pp. 976–986, 2007.

[6] S. Altuntas, “A novel approach based on utility mining for store layout: a case study in a supermarket,” *Industrial Management & Data Systems*, vol. 117, no. 2, pp. 304–319, 2017.

[7] P. Chaudhary, A. Mondal, and P. K. Reddy, “A flexible and efficient indexing scheme for placement of top-utility itemsets for different slot sizes,” in *Proc. BDA*. Springer, 2017, pp. 257–277.

[8] P. Chaudhary, A. Mondal, and P. K. Reddy, “An efficient premiumness and utility-based itemset placement scheme for retail stores,” in *Proc. DEXA*. Springer, 2019, pp. 287–303.

[9] P. Chaudhary, A. Mondal, and P. K. Reddy, “An improved scheme for determining top-revenue itemsets for placement in retail businesses,” *International Journal of Data Science and Analytics*, vol. 10, pp. 359–375, 2020.

[10] T. Brijs, B. Goethals, G. Swinnen, K. Vanhoof, and G. Wets, “A data mining framework for optimal product selection in retail supermarket data: The Generalized PROFSET Model,” in *Proc. SIGKDD*. ACM, 2000, pp. 300–304.

[11] I. Cil, “Consumption universes based supermarket layout through association rule mining and multidimensional scaling,” *Expert Systems with Applications*, vol. 39, no. 10, pp. 8611–8625, 2012.

[12] R. Srikant and R. Agrawal, “Mining generalized association rules,” in *Proc. VLDB*. Morgan Kaufmann, 1995, pp. 407–419.

[13] J. Han and Y. Fu, “Discovery of multiple-level association rules from large databases,” in *Proc. VLDB*, 1995, pp. 420–431.

[14] S. Brin, R. Motwani, and C. Silverstein, “Beyond market baskets: Generalizing association rules to correlations,” in *Proc. ACM SIGMOD*. ACM, 1997, pp. 265–276.

[15] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proc. VLDB*, vol. 1215, 1994, pp. 487–499.

[16] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in *Proc. ACM SIGMOD*. ACM, 2000, pp. 1–12.

[17] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules,” in *Proc. ICDT*. Springer, 1999, pp. 398–416.

[18] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, “UP-Growth: An efficient algorithm for high utility itemset mining,” in *Proc. KDD*. ACM, 2010, pp. 253–262.

[19] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, “FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning,” in *Proc. ISMIS*. Springer, 2014, pp. 83–92.

[20] P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, V. S. Tseng, and U. Faghihi, “Mining minimal high-utility itemsets,” in *Proc. DEXA*. Springer, 2016, pp. 88–101.

[21] P. Fournier-Viger, C.-W. Wu, and V. S. Tseng, “Novel concise representations of high utility itemsets using generator patterns,” in *Proc. ADMA*. Springer, 2014, pp. 30–43.

[22] A. Fares, “The effects of in-store category adjacencies on consumer purchase behavior,” in *Proc. EMS*. SSRN, 2019.

[23] R. Bezawada, S. Balachander, P. Kannan, and V. Shankar, “Cross-category effects of aisle and display placements: a spatial modeling approach and insights,” *Journal of Marketing*, vol. 73, no. 3, pp. 99–117, 2009.

[24] S. Gelper, I. Wilms, and C. Croux, “Identifying demand effects in a large network of product categories,” *Journal of Retailing*, vol. 92, no. 1, pp. 25–39, 2016.

[25] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets, “Using association rules for product assortment decisions: A case study,” in *Proc. SIGKDD*. ACM, 1999, pp. 254–260.

[26] P. Chaudhary, A. Mondal, and P. K. Reddy, “A diversification-aware itemset placement framework for long-term sustainability of retail businesses,” in *Proc. DEXA*. Springer, 2018, pp. 103–118.

[27] Instacart, *Instacart Market-basket analysis dataset*. [Online]. Available: https://www.kaggle.com/c/instacart-market-basket-analysis/data

[28] M. Hahsler, K. Hornik, and T. Reutterer, “Implications of probabilistic data modeling for mining association rules,” in *Proc. Conference of the Gesellschaft fr Klassifikation*. Springer-Verlag, 2005, pp. 598–605.

[29] C. Bapna, *Source Code*. [Online]. Available: https://github.com/chinmay-bpn/Retail-GUI-Index.git