International Journal of Data Science and Analytics (2020) 10:359–375
https://doi.org/10.1007/s41060-020-00221-5

REGULAR PAPER

An improved scheme for determining top-revenue itemsets for placement in retail businesses

Parul Chaudhary[1] · Anirban Mondal[2] · Polepalli Krishna Reddy[3]

Received: 8 September 2019 / Accepted: 24 April 2020 / Published online: 29 May 2020

Abstract

Utility mining has been emerging as an important area in data mining. While existing works on utility mining for retail businesses have primarily focused on the problem of finding high-utility itemsets from transactional databases, they implicitly assume that each item occupies only one slot. Here, the slot size of a given item is the number of (integer) slots occupied by that item on the retail store shelves. However, in many real-world scenarios, the number of slots consumed by different items typically varies. Hence, this paper considers that a given item may physically occupy any fixed (integer) number of slots. Thus, we address the problem of efficiently determining the top-utility itemsets when a given number of slots is specified as input. The key contributions of our work are three fold. First, we present an efficient framework to determine the top-utility itemsets for different user-specified number of slots that need to be filled. Second, we propose a novel flexible and efficient index, designated as Slot Type Utility (STU) index, for facilitating quick retrieval of the top-utility itemsets for a given number of slots. Third, we conducted an extensive performance evaluation using both real and synthetic datasets to demonstrate the overall effectiveness of the STU index in quickly retrieving the top-utility itemsets by considering a placement scheme in terms of execution time and utility (net revenue) as compared to recent existing schemes.

Keywords High-utility itemset mining · Top-*k* mining · Retailing · Supermarkets · Product placement · Indexing

1 Introduction

Over the past decade, we have been witnessing the prevalence of several popular medium-to-mega-sized retail stores with relatively huge retail floor space. Walmart Supercenters are examples of medium-sized retail stores, and Macy's Department Store at Herald Square (New York City, USA), Dubai Mall (Dubai), Shinsegae Centum city Department Store (Busan, South Korea) are examples of mega-sized retail stores. Typically, such mega-sized stores have more than a million square feet of retail floor space [1]. In this regard, research efforts are going on to address issues such as scalable supply chain management [2], inventory management [3], stock-out management [4,5] and placement of items (products) [6] for strategically improving the revenue of retail stores. In this paper, we address the problem of item (and itemset) placement for revenue improvement of the retailer.

Incidentally, the placement of items in large retail stores involves placing tens of thousands of items in the typically massive number of slots (of the shelves) of a given store in a way that maximizes the sales revenue of the retailer. Furthermore, item placement decisions also need to reflect the fact that the physical sizes of the items vary in terms of the number of slots occupied by any given item. For example, suppose an over-sized camping equipment C_x occupying 5 slots produces a revenue of \$5000 per day for the retailer. On the other hand, a set of items A to E (e.g., 500-ml Pepsi bottle, small tube of toothpaste, packet of potato chips) occupying 1 slot *each* produces a revenue of \$1200 a day for the retailer i.e., a total revenue of \$6000 per day. Observe that by replacing item C_x from those 5 slots by items A to E, the retailer would increase its revenue by \$1000 a day. In essence, for maximizing the revenue of the retailer, it becomes critical to

✉ Parul Chaudhary
pc230@snu.edu.in

Anirban Mondal
anirban.mondal@ashoka.edu.in

Polepalli Krishna Reddy
pkreddy@iiit.ac.in

[1] Shiv Nadar University, Greater Noida, India

[2] Ashoka University, Sonipat, India

[3] International Institute of Information Technology, Hyderabad, India

consider not only the revenue earned by a given item, but also the number of slots consumed by the item.

Consider the case of a large retail store with multiple aisles, where each aisle contains items that are stocked in the slots of the shelves. Such slots are either *premium* or *non-premium*. Examples of premium slots include slots that are nearer to the eye level or at the shoulder level of the customers or slots that are at the checkout counters for encouraging impulse buys. On the other hand, non-premium slots are those, which lack visibility and/or physical accessibility, e.g., they may be located very high or very low in the shelves. Thus, a given item placed in such premium slots would have a significantly higher probability of sale than if it were to be placed at other non-premium slots. *Hence, in this paper, we address the issue of placement of items only for the premium slots.*

Observe that there would typically be multiple blocks of premium slots of varying sizes (in terms of the number of slots) across the different aisles in a given large retail store. Now if the store manager of a large retail store S were to greedily fill these premium slots with only the most expensive items, the revenue of S would not be maximized and could actually decrease. This is because customers often tend to buy sets of items (i.e., itemsets) together instead of buying just individual items. From the customers' point of view, it is much more convenient for them to have multiple needs met in one location in S instead of walking considerable distances through several different aisles of a large retail store to locate their desired items one-by-one.

Another option for the store manager of S could be to fill the premium slots with frequent itemsets [7–9]. A frequent itemset refers to the set of items that often appear together in the customer transactions, e.g., bread, jam, milk. However, existing approaches on frequent itemset mining [7–9] determine frequent itemsets based on the frequency of purchase (support). However, they do not consider item prices (utility values). Observe that prices can vary significantly across items (e.g., the price of a branded suit or a Rolex watch versus the price of a carton of milk or a few loaves of bread). Since revenue depends upon both the frequency of sales and the prices of items, placing frequent itemsets in the premium slots may fail to maximize the revenue of the retailer primarily because some of the frequent itemsets could have low revenue.

Another alternative for the store manager could be to consider a utility mining approach [10–19] toward the problem of deciding upon item placement in the premium slots. Incidentally, utility mining has been emerging as an important area in data mining. The goal of utility mining is to determine high-utility itemsets from transactional databases. Here, utility can be defined in terms of revenue, profits, interestingness and user convenience, depending upon the application. Utility mining approaches focus on creating representations of high-utility itemsets [10], identifying the minimal high-utility itemsets [11], proposing upper bounds and heuristics for pruning the search space [12,13] and using specialized data structures, such as the utility list [14] and the UP-Tree [16], for reducing candidate itemset generation overheads. However, existing utility mining techniques have not been designed for the application of itemset placement in brick-and-mortar retail stores. Moreover, given a revenue threshold, the number of candidate itemsets generated by existing utility mining techniques typically explodes, thereby significantly increasing the time required for the retrieval of high-revenue itemsets. Furthermore, existing utility mining techniques do not consider that items can vary in terms of physical size, i.e., different items can consume different numbers of physical slots in the retail store.

In this paper, we propose an improved placement approach, which differs from the existing approaches in the following aspects. First, the existing approaches implicitly assume that a given item occupies only one physical slot in the retail store shelves. However, as discussed previously, in real-world retail scenarios, the number of physical slots occupied by different items typically varies. Second, existing approaches are not capable of efficiently indexing, retrieving and placing top-revenue itemsets of varying given slot sizes. Here, the slot size of a given item is the number of (integer) slots occupied by that item on the retail store shelves.

In this paper, for maximizing the revenue of the retailer, we address the problem of *efficiently* determining the top-revenue itemsets when a given number of slots is specified as input. In the proposed approach, we consider that the placement of itemsets in the slots of a given retail store is influenced by a combination of four factors, namely (a) frequency of item sales, (b) price of items, (c) physical size (in terms of the number of slots) occupied by each item and (d) association between items as well as itemset size. Intuitively, if we address each of these factors in isolation, it may not facilitate revenue improvement for the retailer. To address (a) and (b) in tandem, we propose the revenue metric. To address (c) and (d) in tandem, we propose the **Slot Type Utility (STU)** index for flexible retrieval of itemsets with varied slot sizes by considering the proposed *revenue* metric as well as the number of slots occupied by each item of any given itemset.

By analyzing the transactional data, the number of potential itemsets typically explodes. Hence, the identification of candidate itemsets to be selected for placement in the slots of the retail store by considering prices and physical sizes of the itemsets for maximizing the revenue of the retailer becomes an important research challenge. The key issue here is to efficiently prune the search space for quickly identifying itemsets to be selected for placement in the slots of the retail store by considering prices and physical sizes of the itemsets for maximizing the revenue of the retailer. (Note that the downward closure property is not applicable to utility mining). For efficiently pruning the search space, the challenge is about how to index the high-revenue itemsets in a manner such that itemsets of any given pre-specified physical size (in terms of the number of slots) can be quickly retrieved.

The proposed scheme has the following advantages w.r.t. the existing approaches: (a) It considers that items can be of varying physical sizes in terms of the number of slots occupied by the items, (b) It efficiently extracts the potential top-revenue itemsets for placement based on our proposed STU index, which facilitates quick retrieval of the top-revenue itemsets for a given number of slots, and (c) It effectively places the top-λ high-revenue itemsets (using the STU index) in the given number of premium slots of large retail stores for maximizing the revenue of the retailer.

The key contributions of this work are three fold:

1. Given a transactional database over a set of items with the corresponding utility (price) values,
   – we present an efficient framework to determine the top-utility itemsets for different user-specified number of slots that need to be filled.
   – we propose a novel, flexible and efficient index, designated as the STU index, for facilitating quick retrieval of the top-utility itemsets for a given number of slots.
2. We conducted an extensive performance evaluation using both real and synthetic datasets to demonstrate the effectiveness of the STU index in quickly retrieving the top-utility itemsets by considering a placement scheme. The results of our experiments indicate that our proposed scheme indeed significantly outperforms recent existing schemes in terms of execution time and utility (net revenue).

We shall use revenue as an example of a utility measure throughout this paper; hence, we use the terms revenue and utility interchangeably.

We have made a preliminary effort in [20]. This paper significantly extends the work in [20] with the following additions. First, we have included a placement scheme designated as **TIPDS (Top-utility Itemset Placement scheme for Different Slot-sizes)** as an example to demonstrate the efficiency of the STU index. Second, we added a more extensive set of experiments with additional datasets. Moreover, we demonstrated the improvement in retailer revenue by conducting our experiments by means of dividing each of the datasets into two parts, namely training set and test set.

The remainder of this paper is organized as follows. Section 2 discusses related works, while Sect. 3 describes the context of the problem. Section 4 presents the STU indexing scheme. Section 5 discusses the TIPDS placement scheme. Section 6 reports the performance evaluation. Section 7 discusses some of the issues associated with the deployment of our proposed TIPDS placement scheme in the real world. Finally, we conclude in Sect. 8 with directions for future work.

2 Related work

Existing works on association rule mining [7–9] consider the problem of determining frequent itemsets based on support. However, they do not take into account the utility values of the items. Moreover, the existing works exploit the downward closure property [7], which implies that the subset of a frequent itemset should also be frequent. However, the downward closure property does not apply to utility mining. Given that the determination of the optimal high-utility itemsets would be prohibitively expensive due to the overhead of exhaustive search, it becomes a necessity to design approximate approaches for determining high-utility itemsets.

Research efforts on utility mining include the approaches proposed in [10–19,21–26]. The work in [10] presents concise representations of high-utility itemsets. Moreover, it discusses the HUG-Miner and GHUI-Miner algorithms to mine those representations.

The work in [19] discusses the FHM algorithm, which reduces the cost of mining high-utility itemsets based on an analysis of item co-occurrences. In particular, it uses a pruning mechanism, designated as EUCP (Estimated Utility Co-occurrence Pruning), for directly eliminating low-utility extensions and all of their transitive extensions without having to construct their respective utility lists. Observe that FHM extracts candidate itemsets of different slot sizes, as long as the itemsets satisfy the minimum utility threshold criterion. The work in [11] proposes the MinFHM algorithm, which uses pruning strategies in conjunction with several optimizations for efficient extraction of utility patterns. In particular, MinFHM uses the notion of minimal high-utility itemsets (MinHUIs), which are defined as the smallest itemsets that can generate a large amount of profit. Notably, the search and pruning strategy of MinFHM is geared toward mining only MinHUIs as opposed to mining all of the high-utility patterns. Furthermore, the work in [11] also discusses a representation of minimal high-utility itemsets.

The work in [12] presents an algorithm, designated as EFIM. EFIM determines high-utility itemsets by using two upper bounds, namely sub-tree utility and local utility, for pruning the search space. In a similar vein, the proposal in [13] presents an algorithm, designated as EFIM-Closed, for finding closed high-utility itemsets by using pruning strategies in conjunction with upper bounds for utility. Moreover, the work in [14] prunes the number of candidate itemsets by means of a two-phase algorithm for discovering high-utility itemsets.

The work in [16] mines high-utility itemsets by means of an algorithm, which is designated as the Utility Pattern Growth (UP-Growth) algorithm. The UP-Growth algorithm uses the Utility Pattern Tree (UP-Tree) for maintaining information about high-utility itemsets. The proposal in [15] discusses an algorithm, designated as the HUI-Miner algorithm, for determining high-utility itemsets. The HUI-Miner algorithm stores utility values and heuristic information about the itemsets in a specialized data structure called *the utility list*. In particular, the use of this specialized utility list data structure facilitates the HUI-Miner algorithm toward avoiding utility computations for a large number of candidate itemsets and also helps the algorithm in avoiding expensive candidate itemset generation. Moreover, the work in [24] presents an algorithm for discovering multiple high-utility patterns simultaneously. The algorithm uses a data structure, designated as IUData List, which stores information about itemsets of length-1 as well as the positions of those itemsets in the transactions. Furthermore, the work in [21] proposes a top-k high-utility mining method, which uses the Leaf Itemset Utility data structure in conjunction with a threshold raising strategy.

The work in [17] proposes an algorithm for computing the utility values of the itemsets without generating candidates. The algorithm is designed for mining closed high-utility itemsets. Moreover, the work in [23] discusses an algorithm for mining high-utility patterns from incremental databases without generating any candidates by incorporating restructuring and pruning techniques in conjunction with an indexed list-based data structure. Furthermore, the work in [26] discusses an algorithm for finding high-utility patterns without generating candidates by using a high-utility pattern growth approach in conjunction with a lookahead strategy and a linear data structure.

The work in [25] proposes the notion of the *average utility measure* (of an itemset), which is computed as the total utility of a given itemset divided by the number of items in that itemset. Observe how the average utility measure is capable of providing a better reflection of the utility of any given itemset by considering the length of the itemset. Furthermore, the work in [22] mines high average-utility itemsets by means of a level-wise algorithm. In particular, the algorithm uses the notion of least minimum average-utility in conjunction with pruning strategies for mining high average-utility itemsets.

The proposal in [18] looks at the problem of determining the top-K high-utility closed patterns from the perspective of business goals. Thus, its goal is to determine the top-K high-utility closed patterns that are related specifically to a given business goal. To achieve this goal, it presents a pruning strategy, whose aim is to prune away low-utility itemsets.

An important aspect of product placement in retail stores is stock-outs. In this regard, the work in [4] proposed a conceptual model for testing the impact of stock-outs and stock-out policies on item selection for placement in retail stores. The study in [4] also explored methods for suggesting replacement items in case of stock-outs. The work in [5] used a study of more than 71,000 consumers. The study was conducted as a series of 29 studies across 20 countries; its results indicate that the average worldwide out-of-stock rate was found to be 8.3%.

Utility mining has also been applied to sequence mining [27–30]. The work in [27] incorporates the notion of utility into sequential pattern mining and creates a generic framework for high-utility sequence mining. In particular, the USpan algorithm [27] is proposed for mining high-utility sequential patterns. USpan uses the lexicographic quantitative sequence tree for extracting the complete set of high-utility sequences and designs concatenation mechanisms for calculating the utility of a node and its children by means of pruning heuristics. Furthermore, the work in [28] presents a framework for addressing the problem of top-k high-utility sequential pattern mining. In particular, the Top-k high-utility sequence (TUS) algorithm [28] identifies the top-k high-utility sequential patterns without the requirement of setting the value of a minimum utility threshold. Heuristics for raising the threshold and for pruning away unpromising items have also been investigated in [28].

Moreover, the work in [29] discusses high-utility episode mining, which is concerned with identifying episodes of high utility in a sequence of events with quantities and weights. In particular, the work in [29] refines the earlier problem definitions of high-utility episode mining by ensuring that none of the high-utility episodes is missed. The HUE-Span algorithm [29] is proposed to efficiently find all of the patterns. Additionally, the work in [30] introduces a new framework for mining actionable high-utility association rules, which are designated as Combined Utility-Association Rules (CUAR), whose goal is to identify high-utility and strong association of itemset combinations comprising multiple itemsets connected in the form of item/itemset relations.

Notably, the proposed approach is differentiated from the existing works in that it considers the number of slots occupied by each item and the revenue of items/ itemsets. Furthermore, it also addresses the problem of efficiently identifying, retrieving and placing the top-revenue itemsets such that the itemsets cover exactly a given number of fixed slots.

We have made a preliminary effort in [20]. The approach proposed in this paper is refined significantly in terms of approach, presentation and experimental results.

On the issue of itemset placement in retail stores, we have also investigated approaches by considering the issue of improving the long-term revenue sustainability of the retailer based on itemset diversification [31] and the issue of revenue

Table 1 Price and slot size information of items

Item	A	B	C	D	E	F	G	H	I
Revenue (ρ)	7	2	6	1	3	1	5	4	3
Slot size (ω)	3	2	1	2	4	3	2	5	2

maximization of the retailer by considering the variations in the premiumness of the retail slots [32]. Notably, our proposed scheme differs from the works in [31,32] because they consider different research problems with different underlying assumptions as well as context.

3 Context of the problem

Consider a finite set Υ of m items $\{A_1, A_2, A_3, .., A_m\}$. Each item A_j of set Υ is associated with a price ρ_j, a frequency of sales (support) σ_j and a slot size ω_j. Each item of set Υ may be physically different in size. Thus, each item may consume a different number of slots, e.g., on the shelves of a retail store. We shall henceforth refer to the number of slots that are physically occupied by a given item as the ***slot size*** of that item. We assume that all slots (premium or non-premium) are of equal size.

Let us consider Tables 1 and 2 to better understand the context, where Table 1 indicates the respective prices and slot sizes of the items (A to I), while Table 2 depicts a database of five transactions involving these items.

Definition 1 The **net revenue of item** A_j is denoted as (NR_j) and defined as $NR_j = (\rho_j \times \sigma_j)$, where ρ_j is the price of the item and σ_j is the frequency of sales (support) of the item A_j.

Definition 2 The **net revenue of an itemset** Υ in a transaction T is denoted as $NR(\Upsilon, T)$ and computed as product of the prices of all the item in Υ and frequency of sales (support) of itemset Υ in transaction T.

$$NR(\Upsilon, T) = \sum_{j \in \Upsilon \wedge \Upsilon \subseteq T} \rho_j \times \sigma(\Upsilon, T) \tag{1}$$

For example, in Tables 1 and 2, the net revenue of the itemset {A,D} = (7+1)*6 i.e., 48. Similarly, the net revenue of the itemset {A,C,G} = (7+6+5)*2 i.e., 36.

An **s-slot itemset** is defined as an itemset consuming a total of s slots, i.e., an s-slot itemset has a slot size of s. In Tables 1 and 2, 5 slots can be filled by the following combinations: {H}, {C,D}, {A,B}, {A,D}, {A,G}, {A,I}, {F,B}, {F,D}, {F,G}, {F,I}, {C,E}. Hence, all these combinations constitute 5-slot itemsets.

Definition 3 The **net revenue per slot** of a given itemset Υ in a transaction T is denoted as $NR/\omega(\Upsilon, T)$ and defined as the net revenue of the itemset divided by the total number of slots consumed by all the items in the itemset Υ.

$$NR/\omega(\Upsilon, T) = \frac{\mathrm{NR}(\Upsilon, T)}{\sum_{j \in \Upsilon} \omega_j} \tag{2}$$

For example, in Tables 1 and 2, the net revenue of the itemset {A,D} = 48 and the total number of slots consumed by the itemset is (3+2) i.e., 5. Therefore, net revenue per slot of itemset {A,D} = 48/5, i.e., 9.6.

Definition 4 Let N be the number of itemsets and $\mu_{\mathrm{NR}/\omega}$ be the mean value of NR/ω for all N itemsets. Then the threshold net revenue per slot ($\mathrm{TH}_{\mathrm{NR}/\omega}$) = ($\mu_{\mathrm{NR}/\omega}$ + $(\alpha/100) * \mu_{\mathrm{NR}/\omega}$), where $\mu_{\mathrm{NR}/\omega}$ is the mean value of NR/ω across all the itemsets. Here, α ($0 \le \alpha \le 100$) is a parameter, which controls $\mathrm{TH}_{\mathrm{NR}/\omega}$.

In Definition 4, the parameter α is application-dependent. The purpose of the parameter α is to act as a lever to limit the number of items satisfying the revenue per slot threshold criterion so that items with low revenue per slot can be effectively pruned away from the index. For example, if the revenue per slot of the items were to follow a uniform distribution, nearly half of the items would satisfy the revenue threshold criterion, if α were set to zero; several of these items could possibly have low revenue. For example, in Table 2, the mean net revenue per slot, $\mu_{\mathrm{NR}/\omega}$ = (9.6 + 4.5 + 6 + 1.84 + 7.8)/5, i.e., 5.94. Therefore, setting the value of α to 10%, the threshold net revenue per slot $\mathrm{TH}_{\mathrm{NR}/\omega}$ = 5.94 + (10/100)*5.94, i.e., 6.54.

Given a transactional database over a set of items with the corresponding utility (price) values and a number of slots to be filled, this paper addresses the problem of efficiently determining the top-λ high-revenue itemsets. If we set the value of λ to be high, some of the top-λ itemsets would possibly have low revenue. On the other hand, if the value of λ is set too low, we may miss some itemsets with relatively high revenue. As such, the value of λ is essentially application-dependent; hence, we leave the determination of the optimal value of λ to future work.

Normally, it is easy for the user to specify the number of itemsets (patterns) required instead of setting the value of the threshold net revenue per slot. Typically, a retailer wants to know the number of top-λ revenue itemsets, given the slot size. Here, slot size varies from 1 to the size of large itemset. Intuitively, given a large number of items in set Υ, the number of possible combinations of the items satisfying a given slot size would essentially explode. Hence, a naive brute-force approach of generating and examining all possible itemsets of a given slot size for finding the top-λ high-revenue itemsets would be prohibitively expensive. Hence, it becomes imperative to set threshold values for the net revenue per slot.

Table 2 A sample transaction database

TID	Transaction	Slot (ω)	Frequency of sales (σ)	Net revenue (NR)	NR/ω
1	A,D	5	6	48	9.6
2	B,C,I,F	8	3	36	4.5
3	A,C,G	6	2	36	6
4	A,B,C,G,H	13	1	24	1.84
5	A,C,G,I	8	3	63	7.8

Problem statement Consider the retail store, where each item A_j is associated with a price ρ_j and a slot size ω_j. Given a set D of user purchase transactions on a finite set Υ of m items and N premium slots of the retail store, the problem is to develop an approach to maximize the total revenue of the retailer by *efficiently* determining and placing the top-λ high-revenue itemsets in the given N premium slots.

4 STU index

In this section, we first discuss the basic idea of the STU index and then describe the index. Moreover, we present an example and the algorithm for building the index.

4.1 Basic idea

Our proposed STU indexing scheme aims at efficiently determining the top-utility itemsets of any given slot size, given that the individual slot sizes of the items may vary. The basic idea of the STU index is to store only the top-λ high-revenue itemsets for each different slot size instead of storing all the itemsets of a given slot size. The sth level of the STU index corresponds to itemsets having a slot size of s. The index is built in a level-wise manner starting from the lowest level, which corresponds to itemsets of slot size 1. Then 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.

The issue is to extract top-λ itemsets from the transactional database given the value of λ, level number and revenue threshold value of the level number. It can be observed that we employ $\mathrm{TH}_{\mathrm{NR}/\omega}$ in addition to value of λ for the given level because of the higher value of λ, and there is a possibility of getting non-potential itemsets. To filter nonpotential top-λ itemsets, we employ $\mathrm{TH}_{\mathrm{NR}/\omega}$.

By maintaining only the top-λ itemsets, the STU index restricts the number of candidate itemsets that need to be examined for building the next higher level of the index. This improves the efficiency of computation for building the index because a lower number of candidate itemsets need to be examined. This also reduces index storage costs since the number of itemsets being maintained at each level of the index is upper limited by the value of λ. Furthermore, the STU index also facilitates quick retrieval of high-revenue itemsets of any given slot size. This is because the top-λ high-revenue itemsets are maintained in the index for different slot sizes.

4.2 Description of the STU index

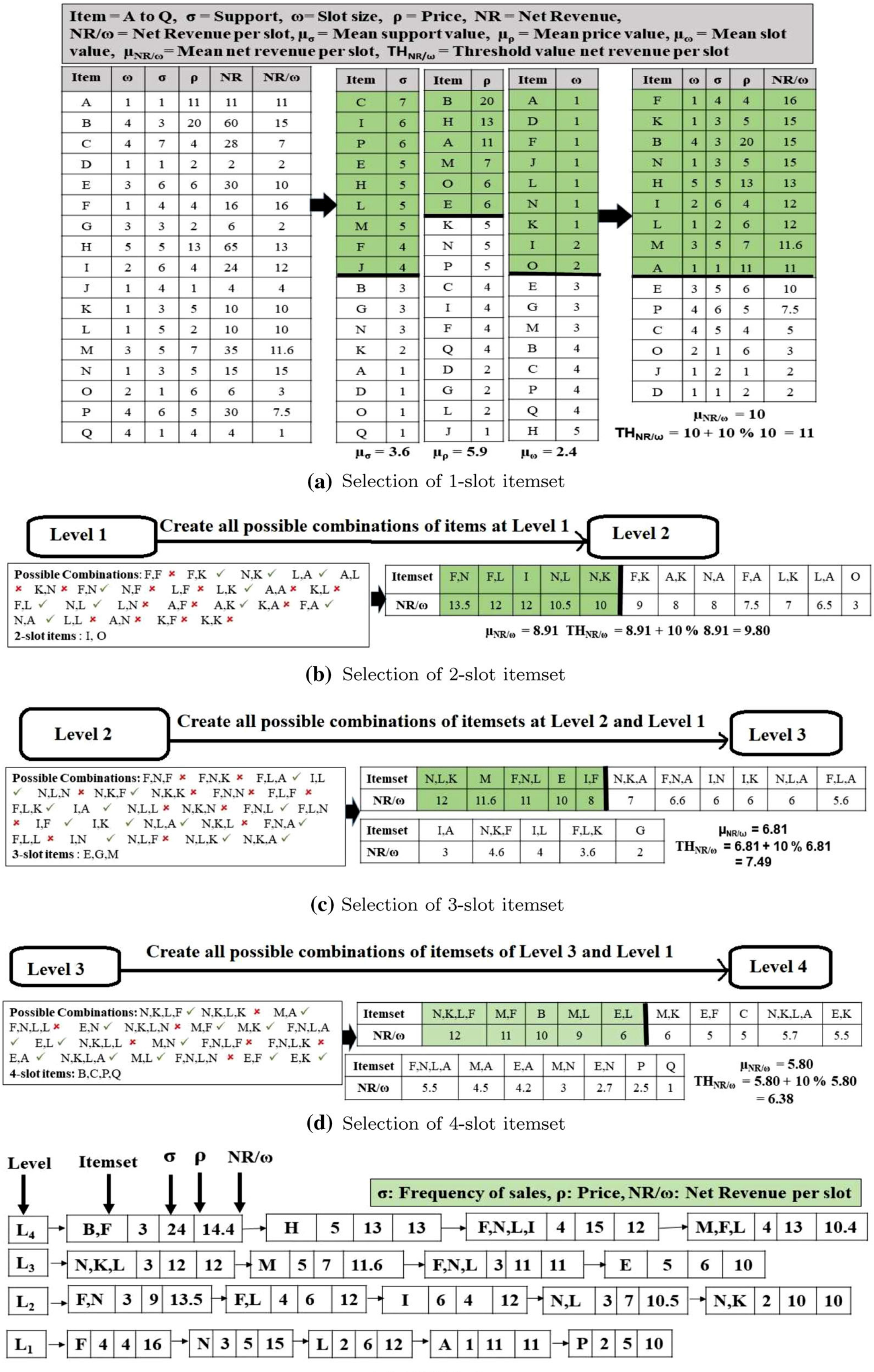
The STU index is essentially a multi-level index, where each level corresponds to a given slot size s. Corresponding to each level, the STU index stores the top-λ high-revenue itemsets of the slot size associated with that level.

Each level in the STU index corresponds to a hash bucket. Thus, for indexing itemsets of N different slot sizes, the index would contain N hash buckets, i.e., one hash bucket per slot size. Hence, when a query Q tries to find the top-utility itemsets of a given slot size s, Q is able to traverse quickly to the sth hash bucket as opposed to having to traverse through all the hash buckets corresponding to $s = \{1, 2, \ldots, s-1\}$.

Now, for each level i in the STU index, the corresponding hash bucket contains a pointer to a linked list of the top-λ high-revenue itemsets of slot size s. The entries of the linked list are of the form (itemset, σ, ρ, NR/ω), where itemset refers to the given itemset under consideration. Here, σ is the support (frequency of sales) of itemset, while ρ refers to the total price of all the items in itemset. NR$/\omega$ is the net revenue per slot, as discussed earlier in Sect. 3 (see Definition 3). The entries in the linked list are sorted in descending order of their values of NR$/\omega$.

4.3 Illustrative example for building the STU index

Figure 1 depicts an illustrative example for the creation of the STU index. Figure 1a indicates 17 items, namely A to Q, with their respective values of slot size ω, support σ, price ρ, net revenue NR and net revenue per slot (NR/ω). For example, in Fig. 1a, the values of ω, σ, ρ, NR and NR$/\omega$ for items B and G are 4, 3, 20, 60, 15 and 3, 3, 2, 6, 2, respectively. In this example for simplicity, we set $\lambda = 5$ in this example, i.e., for each slot size, there would be at most 5 top-revenue per slot itemsets. Since the revenue threshold value $\mathrm{TH}_{\mathrm{NR}/\omega} = 11$, only the items $\{F, K, B, N, H, I, L, M, A\}$ are selected. Since $\lambda = 5$ in this example, only the top-5 of these items (in terms of NR$/\omega$) are inserted into level L_1 of the index.



Item = A to Q, σ = Support, ω = Slot size, ρ = Price, NR = Net Revenue, NR/ω = Net Revenue per slot, μ_σ = Mean support value, μ_ρ = Mean price value, μ_ω = Mean slot value, $\mu_{NR/\omega}$ = Mean net revenue per slot, $TH_{NR/\omega}$ = Threshold value net revenue per slot

Item	ω	σ	ρ	NR	NR/ω
A	1	1	11	11	11
B	4	3	20	60	15
C	4	7	4	28	7
D	1	1	2	2	2
E	3	6	6	30	10
F	1	4	4	16	16
G	3	3	2	6	2
H	5	5	13	65	13
I	2	6	4	24	12
J	1	4	1	4	4
K	1	3	5	10	10
L	1	5	2	10	10
M	3	5	7	35	11.6
N	1	3	5	15	15
O	2	1	6	6	3
P	4	6	5	30	7.5
Q	4	1	4	4	1

Item	σ
C	7
I	6
P	6
E	5
H	5
L	5
M	5
F	4
J	4
B	3
G	3
N	3
K	2
A	1
D	1
O	1
Q	1

$\mu_\sigma = 3.6$

Item	ρ
B	20
H	13
A	11
M	7
O	6
E	6
K	5
N	5
P	5
C	4
I	4
F	4
Q	4
D	2
G	2
L	2
J	1

$\mu_\rho = 5.9$

Item	ω
A	1
D	1
F	1
J	1
L	1
N	1
K	1
I	2
O	2
E	3
G	3
M	3
B	4
C	4
P	4
Q	4
H	5

$\mu_\omega = 2.4$

Item	ω	σ	ρ	NR/ω
F	1	4	4	16
K	1	3	5	15
B	4	3	20	15
N	1	3	5	15
H	5	5	13	13
I	2	6	4	12
L	1	2	6	12
M	3	5	7	11.6
A	1	1	11	11
E	3	5	6	10
P	4	6	5	7.5
C	4	5	4	5
O	2	1	6	3
J	1	2	1	2
D	1	1	2	2

$\mu_{NR/\omega} = 10$

$TH_{NR/\omega} = 10 + 10\ \%\ 10 = 11$

(a) Selection of 1-slot itemset

Level 1 → Create all possible combinations of items at Level 1 → Level 2

Possible Combinations: F,F ✗ F,K ✓ N,K ✓ L,A ✓ A,L ✗ K,N ✗ F,N ✓ N,F ✗ L,F ✗ L,K ✓ A,A ✗ K,L ✗ F,L ✓ N,L ✓ L,N ✗ A,F ✗ A,K ✓ K,A ✗ F,A ✓ N,A ✓ L,L ✗ A,N ✗ K,F ✗ K,K ✗
2-slot items : I, O

Itemset	F,N	F,L	I	N,L	N,K	F,K	A,K	N,A	F,A	L,K	L,A	O
NR/ω	13.5	12	12	10.5	10	9	8	8	7.5	7	6.5	3

$\mu_{NR/\omega} = 8.91$ $TH_{NR/\omega} = 8.91 + 10\ \%\ 8.91 = 9.80$

(b) Selection of 2-slot itemset

Level 2 → Create all possible combinations of itemsets at Level 2 and Level 1 → Level 3

Possible Combinations: F,N,F ✗ F,N,K ✗ F,L,A ✓ I,L ✓ N,L,N ✗ N,K,F ✓ N,K,K ✗ F,N,N ✗ F,L,F ✗ F,L,K ✓ I,A ✓ N,L,L ✗ N,K,N ✗ F,N,L ✓ F,L,N ✗ I,F ✓ I,K ✓ N,L,A ✓ N,K,L ✗ F,N,A ✓ F,L,L ✗ I,N ✓ N,L,F ✗ N,L,K ✓ N,K,A ✓
3-slot items : E,G,M

Itemset	N,L,K	M	F,N,L	E	I,F	N,K,A	F,N,A	I,N	I,K	N,L,A	F,L,A
NR/ω	12	11.6	11	10	8	7	6.6	6	6	6	5.6

Itemset	I,A	N,K,F	I,L	F,L,K	G
NR/ω	3	4.6	4	3.6	2

$\mu_{NR/\omega} = 6.81$
$TH_{NR/\omega} = 6.81 + 10\ \%\ 6.81 = 7.49$

(c) Selection of 3-slot itemset

Level 3 → Create all possible combinations of itemsets of Level 3 and Level 1 → Level 4

Possible Combinations: N,K,L,F ✓ N,K,L,K ✗ M,A ✓ F,N,L,L ✗ E,N ✓ N,K,L,N ✗ M,F ✓ M,K ✓ F,N,L,A ✓ E,L ✓ N,K,L,L ✗ M,N ✓ F,N,L,F ✗ F,N,L,K ✗ E,A ✓ N,K,L,A ✓ M,L ✓ F,N,L,N ✗ E,F ✓ E,K ✓
4-slot items: B,C,P,Q

Itemset	N,K,L,F	M,F	B	M,L	E,L	M,K	E,F	C	N,K,L,A	E,K
NR/ω	12	11	10	9	6	6	5	5	5.7	5.5

Itemset	F,N,L,A	M,A	E,A	M,N	E,N	P	Q
NR/ω	5.5	4.5	4.2	3	2.7	2.5	1

$\mu_{NR/\omega} = 5.80$
$TH_{NR/\omega} = 5.80 + 10\ \%\ 5.80 = 6.38$

(d) Selection of 4-slot itemset

Level | Itemset | σ | ρ | NR/ω

σ: Frequency of sales, ρ: Price, NR/ω: Net Revenue per slot

L_4 → B,F 3 24 14.4 → H 5 13 13 → F,N,L,I 4 15 12 → M,F,L 4 13 10.4
L_3 → N,K,L 3 12 12 → M 5 7 11.6 → F,N,L 3 11 11 → E 5 6 10
L_2 → F,N 3 9 13.5 → F,L 4 6 12 → I 6 4 12 → N,L 3 7 10.5 → N,K 2 10 10
L_1 → F 4 4 16 → N 3 5 15 → L 2 6 12 → A 1 11 11 → P 2 5 10

(e) STU Index

Fig. 1 Illustrative example for building the STU index

Figure 1b indicates all the possible combinations of itemsets of slot size 2 based on the items in level L_1 of the index and the items with slot size of 2. Since the ordering of the items in a given itemset does not matter, we remove the duplicates. The itemsets marked with a tick symbol are the candidate itemsets, while the itemsets marked with the cross symbol are duplicates; hence, they are not candidate itemsets. For example, $\{F, N\}$ is marked with a tick symbol; hence, it is a candidate itemset. However, $\{N, F\}$ is marked with a cross symbol; hence, it is not a candidate itemset. In Fig. 1c, observe how only the itemsets with revenue per slot equal to or above the value of $\text{TH}_{\text{NR}/\omega}$, (i.e., 9.80), are selected to be inserted into level L_2 of the index.

In Fig. 1c, notice how the different combinations of itemsets of slot size 3 are created by combining the itemsets from L_1 and L_2 of the index. Since itemsets, such as $\{F, N, F\}$ and $\{F, N, N\}$, are itemsets of slot size 2, they are not considered as candidate itemsets for building the level L_2 of the index; hence, they are marked with a cross symbol. Similarly, Fig. 1d indicates how the itemsets of slot size 4 are selected into level L_4 of the index. Finally, Fig. 1e depicts the corresponding STU index.

4.4 Creation of the STU index

Using the intuition from the illustrative example in Fig. 1, we will now discuss the creation of the STU index. First, for slot size of 1, we select only those items, which have a slot size of 1 and whose net revenue per slot is either equal to or above a given threshold value. Notably, the purpose of this threshold value is to ensure that items with low revenue per slot (or itemsets, in case of higher slot sizes), are efficiently pruned away from the index. Then, we sort the selected items in descending order of their values of revenue per slot and insert the top-λ items into level 1 of the index. Next, from level 1 of the index, we list all the combinations of the items of slot size 1 and we also list the individual items with slot size of 2. Among these items/itemsets, we select only those, whose revenue per slot is equal to or exceeds a specific revenue per slot threshold due to the rationale explained earlier. Among these itemsets, the top-λ high-revenue itemsets are now inserted into level 2 of the index.

Then, for creating itemsets of slot size 3, we list all of the possible combinations of the items in level 1 of the STU index and the itemsets in level 2 of the index. Additionally, individual items with slot size of 3 are also listed. Among these itemsets of slot size 3, we select only the top-λ high-revenue itemsets, whose revenue either equals or exceeds a given threshold; these selected itemsets are then inserted into level 3 of the index. This process is continued till the maximum level of the index has been populated with the entries of itemsets corresponding to the slot size at that level.

Now let us make a few important observations about the STU index creation algorithm. First, when we build the higher levels of the index, only the top-λ high-revenue itemsets in the lower levels of the index are considered, thereby implicitly restricting the number of candidate itemsets corresponding to each different slot size. Intuitively, we can understand that this prevents the explosion in the total number of itemsets that need to be examined for building the next higher levels of the index. Second, for creating higher levels of the STU index, we could examine possibly multiple lower levels of the index. For example, for building level 6 of the index (i.e., the level of the index containing itemsets with a slot size of 6), we could possibly combine itemsets of slot sizes 1 and 5, itemsets of slot sizes 2 and 4, and so on. Similarly, to build level 9 of the index, we could possibly combine itemsets of slot sizes (1, 8), (2, 7), (3, 6), (4, 5) and so on.

Algorithm 1 depicts the algorithm to build STU index. The index is built in a level-wise manner starting from the first level of the index, i.e., for itemsets of slot size 1. To build the level 1 of the index for 1-slot size items, observe how the algorithm scans the database to compute the frequency of sales and the net revenue of each 1-slot size item and build a triple of the form $< k, f, r >$, where k is the item, f is the frequency of sales of k and r is the net revenue of k for each 1-slot size item (see Lines 1–6). Moreover, it appends all triples of 1-slot size items into $ListItemset$. Then, the algorithm computes total revenue (TR), which is the sum of the net revenue of all 1-slot size item entries in $ListItemset$ and sets a threshold value as discussed in Sect. 3 (see Definition 4) (see Lines 7–8). Only those top-λ 1-slot size items, whose net revenue is greater than or equal to threshold value, are inserted in the descending order of net revenue into the level one of the index (see Lines 9–13).

The higher levels of the STU index are built one-by-one. For building the second level of the index, the algorithm examines all the possible combinations of 2-slot-sized itemsets from level 1 (see Lines 15–17). For building any given higher level L of the index, the algorithm examines all the possible combinations of L-slot-sized itemsets from level 1 and level L-1 (see Lines 19–20). The remaining process to compute the frequency of sales and the net revenue of itemset remains the same as discussed in the building of level 1.

5 TIPDS (Top-utility Itemset Placement scheme for Different Slot-sizes)

It is well-known in the retail industry that strategic placement of items in the retail store slots has a significant impact on sales [33]. If we arbitrarily (or randomly) assign and place the items in the slots, it may fail to maximize the revenue of the retailer. Thus, there is an opportunity to improve the

Algorithm 1 Build_STU_Index( )

```
Input: (i) $I$: set of $m$ distinct items (ii) p[$m$]: price values of $m$ items
    (iii) s[$m$]: number of slots required for each of $m$ item (iv) $D$: a
    transactional database on $I$ (v) $maxL$: maximum level of STU
    index (vi) $\lambda$: number of itemsets at each level of STU index (vii)
    $\alpha$: revenue threshold control parameter
Output: STU Index
Parameter: $STU[maxL, \lambda]$: array of structures of type $< iset, f, nr >$;
    $ListItemset$: array of structures of type $< iset, f, nr >$ /* $iset$ is the itemset; $f$ is the frequency
    of $iset$; $nr$ is the net revenue of $iset$; L: level; t1: temporary
    variable */
// Building the first level of STU index
L=1;
for each item $i$ of $I$ do
    insert $< iset = i, f = 0, nr = 0 >$ into $ListItemset$.
for each item $k$ of each transaction $T \in D$ do
    if $s[k] == 1$ then
        insert $< iset = k, f = f + 1, nr = (nr + p[k]/s[k]) >$ into
        $ListItemset$
$TR$ = Total revenue of all entries in $ListItemset$
Threshold = $[(TR/m) + (\alpha/100 * (TR/m))]$;
for each $< k, f, nr >$ of $ListItemset$ do
    t1=0
    if $nr[k] \geq$ Threshold then
        insert $< k, f, nr >$ into STU[L,t1]; t1++
Sort STU[L,t1] in descending order of $t1$ and keep only the top-$\lambda$
  items.
  // Building second & other levels of STU
  index
for ($L$=2 to $maxL$) do
    if ($L = 2$) then
        Compute all 2-slot-sized combinations by combining top-$\lambda$
        1-slot-sized items from STU[1,$\lambda$] and insert into $ListItemset$
        with the corresponding $f = 0$ and $nr = 0$
        Identify all 2-slot-sized individual items and append them to
        $ListItemset$ with the corresponding $f = 0$ and $nr = 0$;
    if ($L > 2$) then
        Compute all $L$-slot-sized combinations by combining top-$\lambda$
        $(L - 1)$-slot-sized items from STU[$L - 1$,$\lambda$] and 1-slot-sized
        items from STU[1,$\lambda$] and insert into $ListItemset$ with the
        corresponding $f = 0$ and $nr = 0$.
        Identify all L-slot-sized individual items and append them to
        $ListItemset$ with the corresponding $f = 0$ and $nr = 0$.
    for each transaction $T \in D$ do
        Compute all combinations of L-slot sized items/itemsets from
        T and store in $Tset$.
    for each itemset $k \in Tset$ do
        Extract corresponding $< k, f, nr >$ from $ListItemset$
        $p[k] \leftarrow$ sum of price of all items in itemset $k$
        insert $< iset = k, f = f + 1, nr = (nr + p[k]/s[k]) >$ into
        $ListItemset$
    $TR$ = Total revenue of all L-slot sized itemsets $P$ in list−itemset

    Threshold = $[(TR/P) + (\alpha/100 * (TR/P))]$
    t1=0;
    for each itemset $x$ in $ListItemset$ do
        if NR[x] $\geq$ Threshold then
            insert $< x, nr >$ into STU[L,t1]; t1++
    Sort STU[L,t1] in descending order of $t1$ and keep only top-$\lambda$
    itemsets
return STU[$maxL, \lambda$]
```

revenue of the retailer by considering user purchase patterns in conjunction with a methodology for placing items/itemsets in the premium slots.

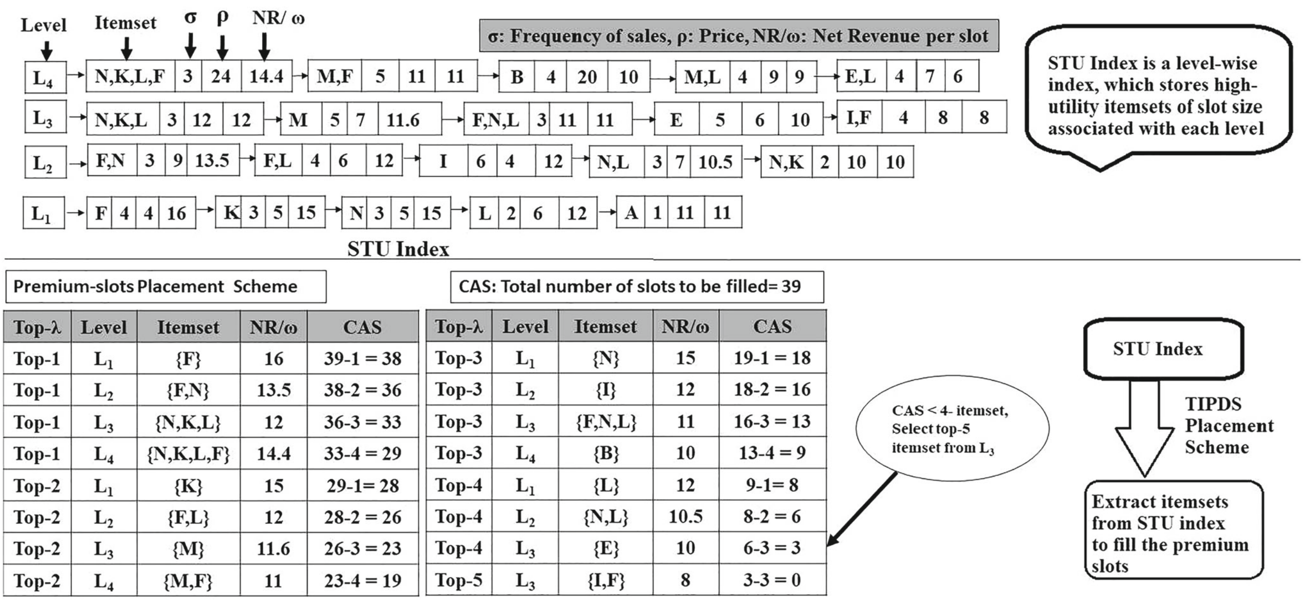
We propose an efficient placement approach Top-utility Itemset Placement scheme for Different Slot-sizes (TIPDS). TIPDS is designed to extract and place the top-λ high-revenue itemsets from STU index in the given number of premium slots of large retail stores for revenue improvement. As discussed earlier in Sect. 4, recall that the STU index is a multi-level index, which stores top-λ high-revenue itemsets of the slot sizes associated with that level.

Our proposed scheme works as follows. We extract top-1 itemsets from each level of the index and place the itemsets in the given premium slots based on the net revenue per slot until we ran out of slot instances. First, we extract top-1 itemset of size s=1 from the level 1 of the STU index (Sect. 4) and place the item in the given premium slot. Then we move to the next level and extract the top-1 highest revenue itemset of size s=2 from level 2 and place the itemset in the premium slots. We progressively fill up the remaining premium slots by extracting the top-1 highest revenue itemset from each level and placing it in the available premium slots. Once we have reached the maximum level of the STU index and still have the premium slots left that needed to be filled, we do a round-robin and circle back to extract the next top-2 highest revenue 1-itemset from level 1, and then we move to the next level, extract the next top-2 highest revenue 2-itemset from that level and so on.

Algorithm 2 Top-utility Itemset Placement scheme for Different Slot-sizes (TIPDS)

```
Input: (i) $I$: set of $m$ distinct items (ii) p[$m$]: price values of $m$ items
    (iii) s[$m$]: number of slots required for each of $m$ item (iv) $D$: a
    transactional database on $I$ (v) $maxL$: maximum level of STU
    index (vi) $\lambda$: number of itemsets at each level of STU index
    (vii) $\alpha$: revenue threshold control parameter (viii) N: number of
    premium slots
Output: Placement of the items/itemsets in N premium slots
1: STU[maxL][$\lambda$] $\leftarrow$ Build_STU_Index( )
2: z=N;
3: while ($z \neq 0$) do
4:     for ($i$=1 to $maxL$) do
5:         for ($j$=1 to $\lambda$) do
6:             extract triple $< k, f, nr >$ at STU[i][j]
7:             place items of itemset $k$ into the premium slots
8:             z = z - i;
```

Algorithm 2 depicts our proposed TIPDS scheme. In Line 1, we invoke Algorithm 1 for building STU index. The premium slots are filled with the items from each level of STU index until all of the slots have been exhausted. First, we extract the top-1 itemset from each level of STU index. Since each level represents the size of the itemset, we place the extracted itemset into the premium slots and update the value


Level
Itemset
σ
ρ
NR/ ω
σ: Frequency of sales, ρ: Price, NR/ω: Net Revenue per slot
L4 → N,K,L,F 3 24 14.4 → M,F 5 11 11 → B 4 20 10 → M,L 4 9 9 → E,L 4 7 6
L3 → N,K,L 3 12 12 → M 5 7 11.6 → F,N,L 3 11 11 → E 5 6 10 → I,F 4 8 8
L2 → F,N 3 9 13.5 → F,L 4 6 12 → I 6 4 12 → N,L 3 7 10.5 → N,K 2 10 10
L1 → F 4 4 16 → K 3 5 15 → N 3 5 15 → L 2 6 12 → A 1 11 11
STU Index is a level-wise index, which stores high-utility itemsets of slot size associated with each level
STU Index
CAS < 4- itemset, Select top-5 itemset from L3
STU Index
TIPDS Placement Scheme
Extract itemsets from STU index to fill the premium slots

Premium-slots Placement Scheme

Top-λ	Level	Itemset	NR/ω	CAS
Top-1	L_1	{F}	16	39-1 = 38
Top-1	L_2	{F,N}	13.5	38-2 = 36
Top-1	L_3	{N,K,L}	12	36-3 = 33
Top-1	L_4	{N,K,L,F}	14.4	33-4 = 29
Top-2	L_1	{K}	15	29-1= 28
Top-2	L_2	{F,L}	12	28-2 = 26
Top-2	L_3	{M}	11.6	26-3 = 23
Top-2	L_4	{M,F}	11	23-4 = 19

CAS: Total number of slots to be filled= 39

Top-λ	Level	Itemset	NR/ω	CAS
Top-3	L_1	{N}	15	19-1 = 18
Top-3	L_2	{I}	12	18-2 = 16
Top-3	L_3	{F,N,L}	11	16-3 = 13
Top-3	L_4	{B}	10	13-4 = 9
Top-4	L_1	{L}	12	9-1= 8
Top-4	L_2	{N,L}	10.5	8-2 = 6
Top-4	L_3	{E}	10	6-3 = 3
Top-5	L_3	{I,F}	8	3-3 = 0

Fig. 2 Illustrative example of TIPDS Scheme

of the premium slots left to be filled after the placement. The approach allocates items in a round-robin manner from level 1 to $maxL$ until all of the premium slots have been filled up (see Lines 3–8).

5.1 Illustrative example

Figure 2 depicts an illustrative example for our proposed TIPDS scheme. For the sake of clarity, this example considers the total number of the premium slots to be 39. To fill the premium slot instances, we extracted itemsets from the STU index. Starting from level 1 to the maximum level of the index, we keep filling up the slots as follows. First, we extract 1-itemset from level 1, i.e., {F} and place it. Next, we update the remaining available slots as (39-1), i.e., 38. Then, we extract the top-1 itemset from level 2, i.e., {F, N} and place it. Now, we update the remaining available slots as (38-2), i.e., 36. To fill the remaining number of the premium slots, we extract the top-1 itemset from level 3 of the STU index, i.e., {N, K, L} and place the itemset in the remaining slots. Now, the available slots are (36-3) i.e., 33.

Now, we extract the top-1 itemset from level 4 of the STU index, i.e., {N, K, L, F}, and place the itemset in the remaining slots. Then, we update the available slots are (33-4), i.e., 29. We have reached the maximum level of index, and still there are 29 slots that are needed to be filled. So, we start extracting top-2 itemsets from each level to fill the remaining slots. First, we extract 2-itemset from level 1, i.e., {K}, and place it. Next, we update the remaining available slots as (29-1), i.e., 28. Then, we extract the top-2 itemset from level 2, i.e., {F, L}, and place it. Now, we update the remaining available slots as (28-2), i.e., 26. To fill the remaining slots, we extract the top-2 itemset from level 3 of the STU index, i.e., {M}, and place the itemset in the remaining slots. Now, the available slots are (26-3), i.e., 23. Now, we extract the top-2 itemset from level 4 of the STU index, i.e., {M, F}, and place the itemset in the remaining slots. Then, we update the available slots are (23-4), i.e., 19.

Next, we extract the top-3 itemset from level 1 of the STU index, i.e., {N}, and place the itemset in the remaining slots. Then, we update the available slots as (19-1), i.e., 18. To fill the remaining slots, we extract the top-3 itemset from level 2 of the STU index, i.e., {I}, and place the itemset in the remaining slots. Now, the available slots are (18-2), i.e., 16. Now, we extract the top-3 itemset from level 3 of the STU index, i.e., {F, N, L}, and place the itemset in the remaining slots. Then, we update the available slots are (16-3), i.e., 13. Next, we extract the top-3 itemset from level 4, i.e., {B}, and place it. Now, we update the remaining available slots as (13-4), i.e., 9. We have reached the maximum level of index, and still there are 9 slots that needed to be filled. So, we start extracting top-4 itemsets from each level to fill the remaining slots. First, we extract the top-4 itemset from level 1 of the STU index, i.e., {L}, and place the itemset in the remaining slots. Then we update the available slots are (9-1), i.e., 8. Next, we extract the top-4 itemset from level 2, i.e., {N, L}, and place it. Now, we update the remaining available slots as (8-2), i.e., 6.

To fill the remaining slots, we extract the top-4 itemset from level 3 of the STU index, i.e., {E}, and place the itemset in the remaining slots. Now, the available slots are (6-3), i.e., 3. The remaining slots are less than the itemset size of the next level, i.e., 3 $<$ level 4-itemset. Hence, we extract the top-5 itemset from level 3 of the STU index, i.e., {I, F}, and place the itemset in the remaining slots. Then we update the available slots are (3-3), i.e., 0. Now, there are no slots left. Hence, the algorithm TIPDS terminates.

6 Performance evaluation

This section reports the performance evaluation by comparing the proposed STU index and placement scheme w.r.t the existing schemes HUI-Miner [14], FHM [19] and MinFHM [11]. We have implemented all of the schemes in Java. All of our experiments were performed on a 64-bit Core i5 processor running Windows 7 with 8 GB memory.

For our experiments, we conducted the experiments using a synthetic dataset as well as two real datasets. The real datasets, namely Retail and Chainstore, were obtained from the SPMF open-source data mining library [34]. Using the IBM data generator, we generated a synthetic dataset, $T20I10N17K|D|2,000K$. The parameters of the synthetic dataset are as follows: T (the average size of the transactions) is 20; I (the average size of potential maximal itemsets) is 10; N (the number of distinct items) is 17K; $|D|$ (the total number of transactions) is 2,000K. For the sake of brevity, we shall henceforth refer to this dataset as the **Synthetic dataset**.

The Retail dataset is an anonymous retail market basket data from an anonymous Belgian retail store, and the Chainstore dataset is the data of customer transactions from a retail store, obtained and transformed from NU-Mine Bench. Table 3 summarizes the number of items and the number of transactions associated with each of these datasets.

Table 3 Statistical information about datasets

Dataset	No. of items	No. of transactions
Retail	16,470	88,162
Chainstore	46,086	1,112,949
Synthetic dataset	17,000	2,000,000

Incidentally, both the Retail dataset and the Synthetic dataset do not provide utility (price) values. Hence, for each of these datasets, we generated the price of the items in the range [0.01, 1.0] using zipf distribution. The Chainstore dataset contains utility values; hence, we have used those utility values in our experiments.

Notably, the physical space consumed by the items can vary in terms of the slot size, (i.e., the number of slots) occupied by a given item. Each item in the dataset is randomly assigned a value of slot size between 1 and 10 as follows. The slot size range is divided into five buckets. We divided the physical size range into five categories, i.e., [1, 2], [3, 4], [5, 6], [7, 8] and [9, 10]. For generating the physical size for each item, we randomly select one of these categories and assign a number of that category. Table 4 summarizes the parameters of the performance study.

Table 4 Parameters for the performance evaluation

Parameter	Default	Variations
Revenue threshold (α)	20	
Queried slots (s)	4	2, 6, 8, 10,12,14,16
Top high-utility itemsets (λ) (10^3)	6	2, 4, 8, 10
Slots to be filled (S)	6000	
Zipf factor (Z_P)	0.7	

To the best of our knowledge, no placement scheme has been proposed in the literature for itemset placement in retail stores. However, several utility mining approaches have been proposed in the literature for extracting high-utility itemsets. Hence, for performance comparison purposes, we used the state-of-the-art FHM algorithm [19], MinFHM algorithm [11] and the HUI-Miner algorithm [14] for the extraction and identification of high-utility itemsets. Although none of these three schemes is a placement scheme, we adapted these schemes for comparing with our proposed TIPDS scheme. The next few paragraphs describe how we adapted these schemes.

Recall from Sect. 2 that the HUI-Miner algorithm [14] stores utility values and heuristic information about the itemsets in a specialized data structure called *the utility list*, which facilitates it in avoiding utility computations for a large number of candidate itemsets as well as in avoiding expensive candidate itemset generation. From the *utility lists*, we extracted all of the itemsets corresponding to each of the respective slot sizes. For example, we extracted the itemsets corresponding to slot size $s = 1$; then, we extracted the itemsets corresponding to slot size $s = 2$ and so on all the way up to the pre-defined maximum slot size supported by our system, thereby segregating itemsets of different slot sizes.

As discussed in Sect. 2, recall that the FHM algorithm [19] uses a strategy based on the analysis of item co-occurrences to reduce the computational cost of mining high-utility itemsets. Observe that FHM extracts candidate itemsets of different slot sizes, as long as the itemsets satisfy the minimum utility threshold criterion. Hence, we extracted all of the itemsets corresponding to each of the respective slot sizes, thereby segregating itemsets of different slot sizes.

Recall from Sect. 2 that the MinFHM scheme [11] defines the notion of minimal high-utility itemsets (MinHUIs) as the smallest itemsets that can generate a large amount of profit. First, we use the MinFHM scheme to generate all of the itemsets consuming different slot sizes. Second, from these generated itemsets, we extracted all of the itemsets corresponding to each slot size, thereby segregating itemsets of different slot sizes.

In essence, for each of the above three schemes, namely HUI-Miner, FHM and MinFHM, observe that we have basically segregated itemsets of different slot sizes, but note that they are not sorted in descending order of revenue. Now we adapt these three schemes toward itemset placement as follows. For each slot size, starting from $s = 1$, we randomly select an itemset from among all of the itemsets of slot size 1. Similarly, for $s = 2$, we randomly select an itemset from among all of the itemsets of slot size 2. In this manner, we continue iterating up to the maximum pre-specified slot size supported by our system. Once we have completed the iteration for the maximum pre-specified slot size, we repeat the process for itemsets of slot size 1, followed by itemsets of slot size 2 and so on in a round-robin manner until all of the desired slots have been filled. For the sake of convenience, we shall henceforth refer to the above three placement schemes as **HUI-Miner**, **FHM** and **MinFHM**, respectively.

We conducted our experiments by dividing the dataset of transactions into two parts, namely training set (containing 70% of the transactions) and test set (containing the remaining 30% of the transactions). We evaluate the performance on the test set. Our performance metrics are index build time (IBT), execution time (ET), the number of patterns generated (N_P) and total revenue (TR). IBT is the time required for building the index. ET is the execution time for the determination of the placement of itemsets in the slots for the training set. N_P is the number of patterns (itemsets) that a given scheme needs to examine for answering a specific query.

TR is the total revenue of the retailer for the test set. During the training phase, we have extracted the set S of itemsets and built the STU index. In the test phase, we consider each transaction having set X of items with the revenue threshold above 50% of the net revenue of the largest itemset Y of S such that Y is a subset of X. The revenue of Y is added to the total revenue. The justification is as follows. In this work, we are evaluating the impact of placing itemsets as opposed to that of placing individual items. We have to create a mechanism to consider the impact of itemsets and ignore the impact of buying individual items. In fact, our way of evaluating the total revenue is strict and conservative because as we can intuitively understand, the actual values of total revenue would be higher if we relax the 50% threshold criteria. In this work, the value of 50% is an example of a threshold, which we have used to investigate the impact of placing itemsets. As a part of future work, we plan to study the effect of varying the value of this threshold.

6.1 Performance of index creation

We performed an experiment to study the performance of index creation. Figure 3 depicts the results for performance of index creation. The results in Fig, 3a–c represent Retail, Chainstore and Synthetic dataset, respectively. The results in Fig. 3a indicate that our proposed scheme TIPDS incurs significantly lower index build time (IBT) than that of HUI-Miner, FHM, MinFHM because TIPDS exploits STU index, which considers only the top-λ itemsets of a given slot size for building the index at each level. On the other hand, HUI-Miner, FHM and MinFHM need to generate all of the itemsets across different itemset sizes (k).

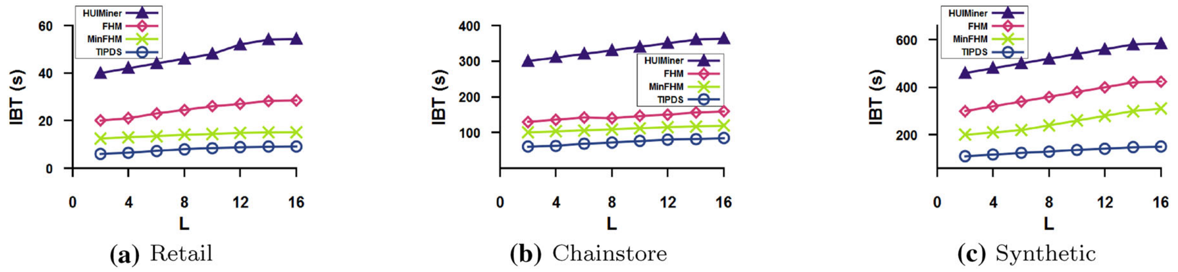
HUI-Miner incurs higher index build time as compared to that of FHM and MinFHM because it builds utility list corresponding to each pattern and larger patterns are obtained by performing the join operation of utility lists of smaller patterns. On the other hand, in case of FHM, IBT remains lower as compared to that of HUI-Miner. This is because it eliminates low-utility itemsets without performing join operations to extract high-utility itemsets using the utility list data structure. Next, MinFHM extends FHM for HUI mining by exploiting pruning strategies to mine only MinHUIs as oppose to all of HUIs. Hence, it incurs lower IBT as compared to that of HUI-Miner and FHM. After all of the itemsets have been generated, segregating the itemsets across the different levels of the index based on slot size constitutes a relatively minor overhead.

In contrast, our proposed scheme TIPDS builds STU index in a level-by-level manner. Hence, its index build time increases as the number L of levels in the index increases. Furthermore, as the value of L increases, a considerably higher number of itemset combinations qualify w.r.t. the revenue per slot threshold criterion of STU. However, beyond $L = 14$, all the schemes exhibit a saturation effect because based on typical purchase transactions, customers do not typically purchase a large number of items together. Hence building the index beyond a certain level is not useful.

The results in Fig. 3b, c follow similar trends as those of Fig. 3a; the actual values are higher in case of the results in Fig. 3b, c since the Chainstore and Synthetic dataset has a significantly higher number of items as well as a higher number of transactions as compared to those of the Retail dataset.

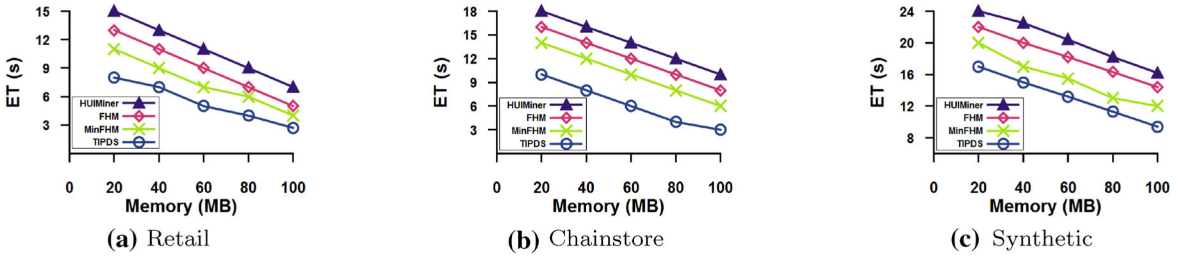
6.2 Memory allocation

Figure 4a–c depicts the effect of variations in memory allocation for the Retail dataset, Chainstore dataset and Synthetic dataset, respectively. Since the STU index, Utility list (MinFHM), Utility list (FHM) and Utility list (HUI-Miner) do not fit into main memory, additional disk I/Os are required. In this experiment, the size of main memory is controlled while running the algorithms. The generated candidate itemsets are stored in the main memory and disk for all the schemes. The result in Fig. 4a indicates that TIPDS incurs significantly lower execution time (ET) as compared to those of HUI-Miner, FHM and MinFHM on varying the size of the



(a) Retail **(b)** Chainstore **(c)** Synthetic

Fig. 3 Performance of index creation



(a) Retail **(b)** Chainstore **(c)** Synthetic

Fig. 4 Memory allocation

main memory because it generates less candidate itemsets as compared to HUI-Miner, FHM and MinFHM. Hence, it results in lower number of disk I/O. On the other hand, HUI-Miner performs the join operation to find larger patterns by joining utility lists of smaller patterns, which return more number of pattern as a result and increases the number of disk I/Os.

The result in Fig. 4a indicates that ET decreases for all the schemes with increase in the value of main memory. This occurs because as the value of main memory increases, more number of candidate itemsets can fit in the main memory. Hence, a lower number of disk I/Os result in lower ET.

Notably, the results in Fig. 4b, c follow similar trends as those of Fig. 4a; the actual values are higher in case of the results in Fig. 4b, c due to the larger sizes of the Chainstore dataset and Synthetic dataset w.r.t. the Retail dataset in terms of the number of items and the number of transactions.

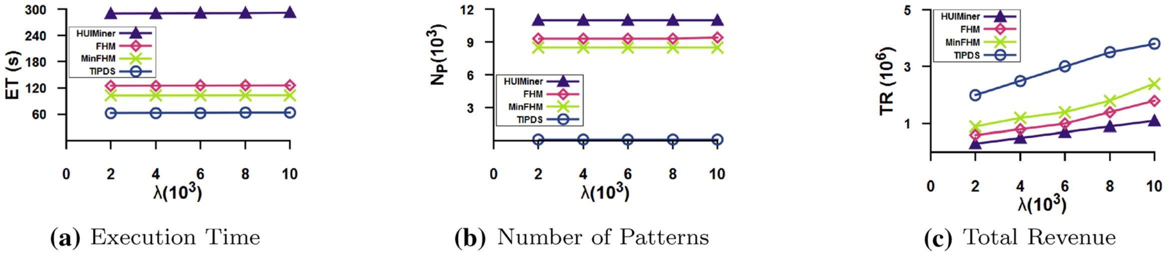
6.3 Effect of variations in λ

Figures 5, 6 and 7 depict the effect of variations in λ for the Retail dataset, Chainstore dataset and Synthetic dataset, respectively. The results in Fig. 5a, b indicate that TIPDS incurs significantly lower execution time (ET) and it has to examine a considerably lower number of patterns (itemsets) as compared to those of HUI-Miner, FHM and MinFHM. This occurs because when a query comes in to determine the top-λ itemsets of slot size s, TIPDS just needs to traverse to the level in the STU index corresponding to s. Then it simply needs to retrieve the λ itemsets from the linked list entries at that level of the index. In contrast, FHM and MinFHM need to first generate all of the itemsets of the queried slot size s. Then it needs to extract all the itemsets of slot size s. Finally, it has to randomly select any λ itemsets from these extracted itemsets as the query result. On the other hand, HUI-Miner extracts all the itemsets from utility list of the queried slot size s and select λ itemsets as the query result.

As the value of λ increases, ET remains comparable for FHM and MinFHM since they incur their predominant cost in generating all the itemsets of varied slot sizes and then extracting the itemsets of the queried slot sizes. Consequently, the random selection of any λ itemsets of the queried slot size from the extracted itemsets represents a relatively minor overhead.

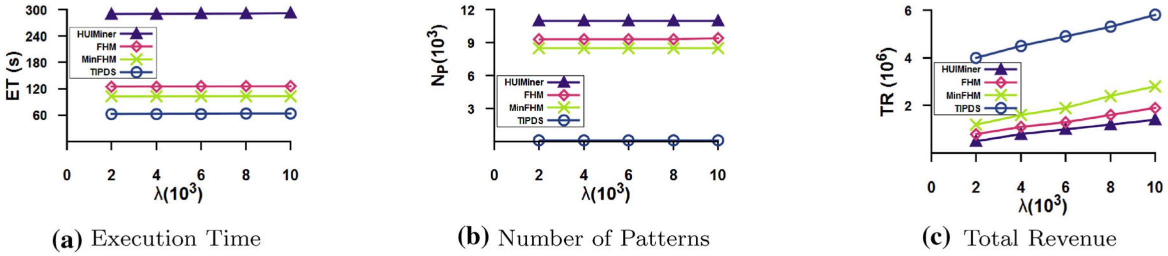
The results in Fig. 5c indicate that all of the schemes exhibit higher values of total revenue (TR) with increase in the value of λ. This occurs because as the value of λ increases, more itemsets are retrieved as the query result for both of the schemes; an increased number of retrieved itemsets implies more total revenue. TIPDS shows significantly higher values of TR as compared to that of HUI-Miner, FHM and MinFHM because it is able to directly select the top-λ high-revenue itemsets from its index. On the other hand, HUI-Miner, FHM and MinFHM randomly select λ itemsets from their respective utility lists. HUI-Miner exhibits lower value of TR because it mines the high-utility itemsets by joining smaller patterns and only compares the utility value of joining patterns to mine high-utility itemsets.

Notably, the results in Figs. 6 and 7 follow similar trends as those of Fig. 5. The actual values are higher in case of the results in Figs. 6 and 7 due to the larger size of the Chainstore



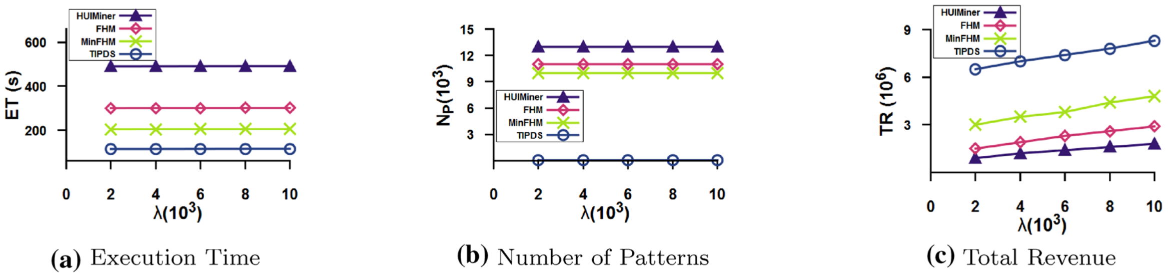
(a) Execution Time **(b)** Number of Patterns **(c)** Total Revenue

Fig. 5 Effect of variations in λ (Retail dataset)



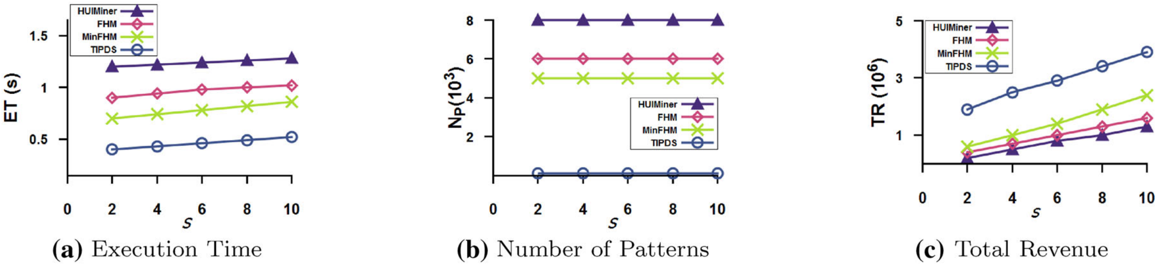
(a) Execution Time **(b)** Number of Patterns **(c)** Total Revenue

Fig. 6 Effect of variations in λ (Chainstore dataset)



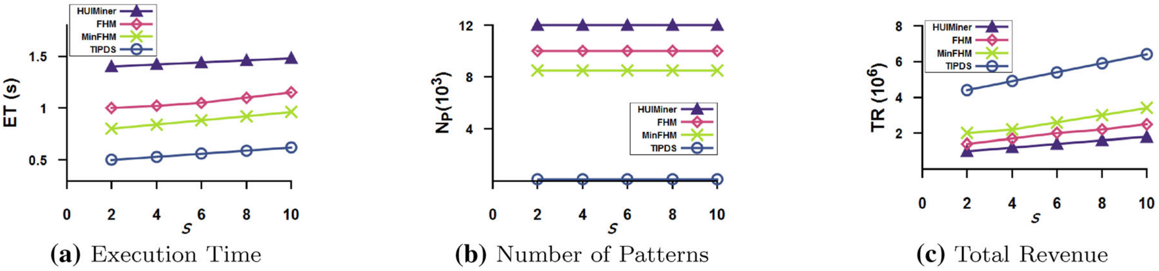
(a) Execution Time **(b)** Number of Patterns **(c)** Total Revenue

Fig. 7 Effect of variations in λ (Synthetic dataset)



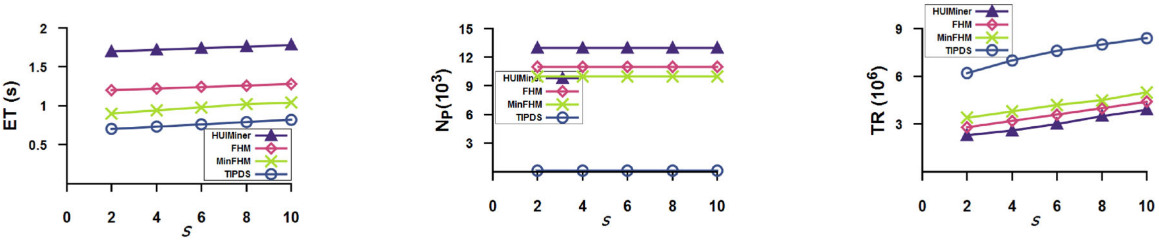
(a) Execution Time **(b)** Number of Patterns **(c)** Total Revenue

Fig. 8 Effect of variations in *s* (Retail dataset)



(a) Execution Time **(b)** Number of Patterns **(c)** Total Revenue

Fig. 9 Effect of variations in *s* (Chainstore dataset)



(a) Execution Time (b) Number of Patterns (c) Total Revenue

Fig. 10 Effect of variations in s (Synthetic dataset)

dataset and Synthetic dataset w.r.t. the Retail dataset in terms of the number of items and the number of transactions.

6.4 Effect of variations in slot size *s*

Figures 8, 9 and 10 depict the results for the Retail dataset, the Chainstore dataset and the Synthetic dataset, respectively, when we vary the queried slot size s.

The results in Fig. 8a, b indicate that TIPDS outperforms HUI-Miner, FHM and MinFHM in terms of ET and N_P due to the reasons explained earlier for the results of Fig. 5a, b, respectively. Observe that both ET and N_P increased albeit slightly for all the schemes. This is because as the value of s increases, more slots need to be filled, thereby necessitating a slightly higher number of patterns to be examined. However, this increase in both ET and N_P is only slight for TIPDS because of its efficient indexing mechanism, which maintains the top-λ high-utility itemsets. Thus, TIPDS only needs to examine the top-λ itemsets from its STU index level that corresponds to a given queried slot size; the slight increase arises from the traversal of more linked list entries at that level of the index in response to an increase in the queried slot sizes. On the other hand, the predominant cost for HUI-Miner, FHM and MinFHM arises from the candidate itemset generation. HUI-Miner, FHM and MinFHM have to generate all of the candidate itemsets of varied slot sizes first and then extract the itemsets of the queried slot size s.

The results in Fig. 8c indicate that the total revenue TR increases for all of the schemes with increase in s. This occurs because as the value of s 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. TIPDS provides higher TR than that of HUI-Miner, FHM and MinFHM since TIPDS selects the top-λ high-revenue itemsets from STU index, while HUI-Miner, FHM and MinFHM randomly select any λ itemsets from their utility lists.

Observe that the results in Figs. 9 and 10 follow similar trends as those of Fig. 8; the difference in the actual values of the performance metrics arises due to the respective dataset sizes.

7 Discussion

In this section, we discuss some of the issues associated with the deployment of our proposed TIPDS placement scheme in the real world.

For performing a real-world deployment of the system, we observe that different retailers may have different ways for placing items and itemsets. For example, in a supermarket, the retailer may usually place similar items together as opposed to placing the items of a given itemset together. Moreover, some retailers may place a given item at only one location in the retail store, while others may decide to place a given item at multiple locations in the retail store. In fact, observe that our proposed TIPDS scheme allows duplication, i.e., a given item may belong to multiple itemsets and can be placed at different locations in the retail store.

We believe that placing a given item in more than one location (by associating the item with different itemsets) may increase the revenue of the retailer. In fact, many retailers nowadays have a business strategy of consciously placing the same item at different locations by associating that item with multiple itemsets. Notably, our proposed TIPDS scheme is only the basis for a recommendation system for decision-making regarding itemset placement. Hence, if there is a significant amount of duplication, the retailer may not wish to consider the recommendation of the decision-support system. We would like to investigate this important issue as a part of our future work.

Although retailers may have their own ways for placing items and itemsets, we observe that the retail business is highly competitive and retailers are always looking for new and innovative ways to boost sales revenue, and appropriate itemset placement is one of the ways. Our proposed TIPDS scheme essentially acts as a basis for a decision-support system to recommend the retailer toward effective itemset placement geared toward revenue maximization. One of the strategies that a retailer may adopt is to follow its usual placement scheme for a certain percentage of slots and use the recommendations provided by our proposed placement scheme for the remaining percentage of slots on an experimental basis. In essence, for revenue improvement in the highly competitive retail business, retailers would wish to try out various placement schemes on an experimental basis, and our proposed scheme may be considered as one of the efforts in this direction. As a part of our future work, we also plan to investigate different placement schemes with the primary objective of revenue maximization for the retailer.

8 Conclusion

Utility mining has been emerging as an important area in data mining. There are efforts in the literature to develop algorithms for extracting high-utility itemsets from large transactional databases. Utility mining approaches have also been extended to improve the performance of item placement in retail stores. However, they implicitly assume that each item physically occupies only one slot. This is in contrast with many real-world scenarios, where the number of slots consumed by different items typically varies. Hence, in this paper, we have considered that a given item can occupy any fixed (integer) number of slots. In this regard, we have addressed the problem of *efficiently* determining and placing the top-utility itemsets when a given number of slots is specified as input.

The key contributions of our work include an efficient framework to determine the top-utility itemsets for different queried slot sizes. Moreover, we have proposed the novel, flexible and efficient index, designated as Slot Type Utility (STU) index, for facilitating quick retrieval of the top-utility itemsets for a given number of slots. Furthermore, we have an extensive performance evaluation using both real and synthetic datasets to demonstrate the effectiveness of the STU index by considering our proposed placement scheme called TIPDS.

In the near future, we plan to investigate different placement schemes for maximizing the revenue of the retailer. Additionally, we plan to study the issue of duplication in the placement of a given item at multiple locations in the retail store. Furthermore, as a part of our future work, we shall investigate how to modify TIPDS for utility sequence mining.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Largest retail stores, https://www.thebalancesmb.com/largest-retail-stores-2892923
2. Bhattacharjee, S., Ramesh, R.: A multi-period profit maximizing model for retail supply chain management: an integration of demand and supply-side mechanisms. Eur. J. Oper. Res. **122**(3), 584–601 (2000)
3. Caro, F., Gallien, J.: Inventory management of a fast-fashion retail network. Oper. Res. **58**(2), 257–273 (2010)
4. Breugelmans, E., Campo, K., Gijsbrechts, E.: Opportunities for active stock-out management in online stores: the impact of the stock-out policy on online stock-out reactions. J. Retail. **82**(3), 215–228 (2006)
5. Corsten, D., Gruen, T.: Desperately seeking shelf availability: an examination of the extent, the causes, and the efforts to address retail out-of-stocks. Int. J. Retail. Distrib. Manag. **31**(12), 605–617 (2003)
6. Ahn, K.I.: Effective product assignment based on association rule mining in retail. Expert Syst. Appl. **39**(16), 12551–12556 (2012)
7. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proc. VLDB. **1215**, 487–499 (1994)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of ACM SIGMOD. vol. 29, pp. 1–12. ACM (2000)
9. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Proceedings of ICDT. pp. 398–416. Springer (1999)
10. Fournier-Viger, P., Wu, C.W., Tseng, V.S.: Novel concise representations of high utility itemsets using generator patterns. In: Proceedings of ADMA. pp. 30–43. Springer (2014)
11. Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S., Faghihi, U.: Mining minimal high-utility itemsets. In: Proceedings of DEXA. pp. 88–101. Springer (2016)
12. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.: EFIM: a highly efficient algorithm for high-utility itemset mining. In: Proceedings of MICAI. pp. 530–546. Springer (2015)
13. Fournier-Viger, P., Zida, S., Lin, J.C.W., Wu, C.W., Tseng, V.S.: EFIM-Closed: fast and memory efficient discovery of closed high-utility itemsets. In: Proceedings of MICAI. pp. 199–213. Springer (2016)
14. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of CIKM. pp. 55–64. ACM (2012)
15. Liu, Y., Liao, W.K., Choudhary, A.: A fast high utility itemsets mining algorithm. In: Proceedings of Workshop on Utility-Based Data Mining. pp. 90–99. ACM (2005)
16. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: UP-Growth: an efficient algorithm for high utility itemset mining. In: Proceedings of ACM SIGKDD. pp. 253–262. ACM (2010)
17. Tseng, V.S., Wu, C.W., Fournier-Viger, P., Philip, S.Y.: Efficient algorithms for mining the concise and lossless representation of high utility itemsets. IEEE Trans. Knowl. Data Eng. **27**(3), 726–739 (2015)
18. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: Proceedings of ICDM. pp. 19–26. IEEE (2003)
19. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proceedings of ISMIS. pp. 83–92. Springer (2014)
20. Chaudhary, P., Mondal, A., Reddy, P.K.: A flexible and efficient indexing scheme for placement of top-utility itemsets for different slot sizes. In: Proceedings of BDA. pp. 257–277. Springer (2017)
21. Krishnamoorthy, S.: Mining top-*k* high utility itemsets with effective threshold raising strategies. Expert Syst. Appl. **117**, 148–165 (2019)
22. Lin, J.C.W., Li, T., Fournier-Viger, P., Zhang, J., Guo, X.: Mining of high average-utility patterns with item-level thresholds. J. Internet Technol. **20**(1), 187–194 (2019)
23. Yun, U., Nam, H., Lee, G., Yoon, E.: Efficient approach for incremental high utility pattern mining with indexed list structure. Future Gen. Comput. Syst. **95**, 221–239 (2019)

24. Jaysawal, B.P., Huang, J.W.: DMHUPS: discovering multiple high utility patterns simultaneously. Knowl. Inf. Syst. **59**(2), 337–359 (2019)
25. Hong, T.P., Lee, C.H., Wang, S.L.: Effective utility mining with the measure of average utility. Expert Syst. Appl. **38**(7), 8259–8265 (2011)
26. Liu, J., Wang, K., Fung, B.C.: Mining high utility patterns in one phase without generating candidates. IEEE Trans. Knowl. Data Eng. **28**(5), 1245–1257 (2015)
27. Yin, J., Zheng, Z., Cao, L.: Uspan: an efficient algorithm for mining high utility sequential patterns. In: Proceedings of KDD. pp. 660–668 (2012)
28. Yin, J., Zheng, Z., Cao, L., Song, Y., Wei, W.: Efficiently mining top-k high utility sequential patterns. In: Proceedings of ICDM. pp. 1259–1264. IEEE (2013)
29. Fournier-Viger, P., Yang, P., Lin, J.C.W., Yun, U.: Hue-span: fast high utility episode mining. In: Proceedings of ICADMA. pp. 169–184. Springer (2019)
30. Shao, J., Yin, J., Liu, W., Cao, L.: Mining actionable combined patterns of high utility and frequency. In: Proceedings of DSAA. pp. 1–10. IEEE (2015)
31. Chaudhary, P., Mondal, A., Reddy, P.K.: A diversification-aware itemset placement framework for long-term sustainability of retail businesses. In: Proceedings of DEXA. pp. 103–118. Springer (2018)
32. Chaudhary, P., Mondal, A., Reddy, P.K.: An efficient premiumness and utility-based itemset placement scheme for retail stores. In: Proceedings of DEXA. pp. 287–303. Springer (2019)
33. Hart, C.: The retail accordion and assortment strategies: an exploratory study. Int. Rev. Retail. Distrib. Consumer Res. **9**(2), 111–126 (1999)
34. SPMF: A Java Open-Source Data Mining Library, http://www.philippe-fournier-viger.com/spmf/datasets

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.